My first Project
Mathematics 502
California State Univ. Fullerton, Fall 2007

Nasser M. Abbasi

December 15, 2007 Compiled on October 14, 2025 at 5:18pm [public]

Contents

0.1 Introduction

Project handout given to us by Professor Jamshidian. The following describes the project.

Project 1: Generating pseudo random numbers from various distributions

Instructions: Submit a hard copy of your project, including the code. Send a softcopy of your code by e-mail with subject line "code for Project 1 – Your name". Please choose filenames that clearly indicate which program belongs to which (part of a) problem [e.g. "prob3bc.R" for problem 3 section c]. I will accept code in Matlab, R, or Mathematica. However, I will provide help only to those that use R, just to encourage you to use R. In fact the project is written with references to the R language, but you can use equivalent commends in Matlab or Mathematica.

1. `[10 points]` The distribution function for the exponential distribution with parameter λ is given by

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & \text{Otherwise.} \end{cases}$$

a. Write an `R` program that uses F^{-1} and the uniform random number generator in `R` (`runif`) to generate n numbers from the exponential distribution with λ. Use `set.seed(`*your birthday MMDDYY*`)`.

b. Generate $n = 10000$ values from the exponential random variable with parameter $\lambda = 2$, using your program (don't give me the numbers generated!). Use the `hist` command in R to plot the *density* histogram of the *relative frequencies* for the generated data. Overlay this histogram by the graph of the density of the exponential random variable with parameter $\lambda = 2$. The commands `dexp` and `lines`, or `curves` can be used. Select an appropriate number of bins for your histogram. Briefly comment on the relationship between the histogram and the density curve.

2. `[20 points]` Let $\phi_0(x)$ and $\phi_1(x)$ denote the density functions for two normal random variables with $\sigma = 1$, and respective means 0 and 3. The density for a mixture distribution of these is defined as

$$f(x) = \gamma\phi_0(x) + (1 - \gamma)\phi_1(x) \quad -\infty < x < \infty.$$

for a given admixture parameter $0 < \gamma < 1$.

a. Write an `R` program, using the normal random generator `rnorm` and the uniform random number generator `runif`, to generate n random numbers from this mixture. The inputs to your program should be n, and γ. Generate two sets of 10,000 data points; one with $\gamma = .75$, and another with $\gamma = .25$.

b. For each of the data sets generated, graph the density histogram and superimpose it by the density $f(x)$ defined in the equation above. Make sure to choose the number of bins for the histogram appropriately. In each case explain why the shape of the density that you obtain is expected.

3. [30 points] Problem 42 on page 111 of your text gives the pdf for the double exponential density with parameter λ. It also suggests a method to generate random numbers from the double exponential family using two random variables W and T, described in the problem.

a. Write a program that generates random numbers from the double exponential family. The input to the program should be the parameter λ, and the number of random numbers to be generated, n. The output should be n pseudo random numbers from the double exponential with parameter λ. You are only allowed to use `runif` in your program for random number generation.

b. Write a program that uses the Accept/Reject algorithm efficiently to generate n observations from the standard normal density $N(0, 1)$, using random numbers that are generated from the uniform(0,1) and the double exponential random variate with parameter $\lambda = 1$ [your program in part (a)]. Your program should also count and report the proportion of values that are rejected. Give the density histogram of $n = 10,000$ numbers generated from your program and superimpose it by the standard normal pdf.

The project contains 3 problems. To make it easier to run each problem (due to the use of Dynamic UI in the project), I implemented each one in a separate Mathematica notebook.

Below are the links to each problem. There are 3 links to each problem, one for the PDF file report, second for the HTML version of the report, and one for the Mathematica notebook itself to run the code.

In all 3 cases, to run the Mathematica code, please do the following: Download the Mathematica notebook. Open it using Mathematica. Click on the Evaluation menu option at the top, and then select Evaluate Notebook. This will run all the code. Scroll to the bottom of the notebook and the GUI should be UP and ready to use.

0.2 Report and code links

0.2.1 problem 1

Project one. Problem one. Mathematics 502 Probability and Statistics

Nasser Abbasi, September 26,2007. California State University, Fullerton

1. [10 points] The distribution function for the exponential distribution with parameter λ is given by

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & \text{Otherwise.} \end{cases}$$

a. Write an R program that uses F^{-1} and the uniform random number generator in R (runif) to generate n numbers from the exponential distribution with λ. Use set.seed(*your birthday MMDDYY*).

b. Generate $n = 10000$ values from the exponential random variable with parameter $\lambda = 2$, using your program (don't give me the numbers generated!). Use the hist command in R to plot the *density* histogram of the *relative frequencies* for the generated data. Overlay this histogram by the graph of the density of the exponential random variable with parameter $\lambda = 2$. The commands dexp and lines, or curves can be used. Select an appropriate number of bins for your histogram. Briefly comment on the relationship between the histogram and the density curve.

Problem 1 part (a)

The CDF given is defined as $F(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & o.w. \end{cases}$ To find F^{-1} we need to solve for x in the equation $1 - e^{-\lambda x}$ for $x \geq 0$ Hence we write

$$\begin{aligned} y &= 1 - e^{-\lambda x} \\ e^{-\lambda x} &= 1 - y \\ -\lambda x &= \ln(1 - y) \\ x &= \frac{-1}{\lambda} \ln(1 - y) \end{aligned}$$

Therefore

$$F^{-1}(y) = \frac{-1}{\lambda} \ln(1 - y)$$

Now to generate random numbers which belongs to an exponential distribution, we will now generate random numbers from $U(0, 1)$ and for each such number generated, we will apply the above function F^{-1} on it, and the result will be a random number which belongs to the exponential distribution. For example, if $\lambda = 2$ and a uniform random number is say 0.4, then we evaluate $F^{-1}(0.4) = \frac{-1}{2} \ln(1 - 0.4) = 0.25541$

And so this is the idea to implement. We need to first seed the uniform random number generator before we start.

Algorithm

Input: λ: parameter, n: number of random numbers to generate

output: a list of n random numbers from the probability density function $\sim F(x)$ given above.

1. Seed the uniform random number generator with (010101).
2. initialize the array d of size n which will contain the list of random numbers generated below.

This loop below is just an algorithmic view. In actual code, a 'vector' operation Table[] in used for speed.

3. For i in 1..n LOOP

Generate y_k which is a random generated from uniform distribution using the build in function RandomReal[0,1]

$d[i] = F^{-1}(y_k)$ using input λ.
END LOOP

4. Find histogram of d. Select an appropriate number of bins. Let f_a be the histogram found.

5. Now find the relative frequency f_r by dividing set f_a by the number of observations n. Hence histogram now is $f_r = \frac{f_a}{n}$

6. Now scale the histogram such that it is density. Total area is 1. Do this by finding total area under histogram, and divide each bin count by this area.

7. Plot the histogram and the exponential distribution $\lambda e^{-\lambda x}$ on the same plot.

Code Implementation

Define the function F^{-1}which was derived earlier. This is the inverse of the CDF of the exponential density function $\lambda e^{-\lambda x}$

In[156]:=
```
Remove["Global`*"];
gDebug = False;
```

In[158]:=
```
inverseCDFofExponentialDistribution[λ_, n_] := Module[{}, -1/λ Log[1 - n]]
```

Function below is called to generate N random numbers using the above F^{-1} function (User needs to seed before calling

In[159]:=
```
getRandomNumbersFromExponential[λ_, nRandomVariables_] := Module[{i},
  Table[ inverseCDFofExponentialDistribution[λ, RandomReal[]], {i, nRandomVariables}]]
```

Problem 1 part(b)

Generate n = 10000 for $\lambda = 2$ and overlay with relative frequency, use appropriate number of bins. See appendix for the function postProcessForPartOne[] which generate the plots. Removed below to reduce code clutter in the main report.

This function makes a histogram which is scaled to be used to overlay density plots, or other functions.
Input: originalData: this is an array of numbers which represents the data to bin
nBins: number of bins
output: the histogram itself but scaled such that area is ONE

In[160]:=
```
Needs["BarCharts`"]
nmaMakeDensityHistogram[originalData_, nBins_] :=
 Module[{freq, binSize, from, to, scaleFactor, j, a, currentArea},
  to = Max[originalData];
  from = Min[originalData];
  binSize = (to - from) / nBins;
  freq = BinCounts[originalData, binSize];
  currentArea = Sum[binSize * freq[[i]], {i, nBins}];
  freq = freq / currentArea;
  a = from;
  Table[{a + (j - 1) * binSize, freq[[j]], binSize}, {j, 1, nBins}]
 ]
```

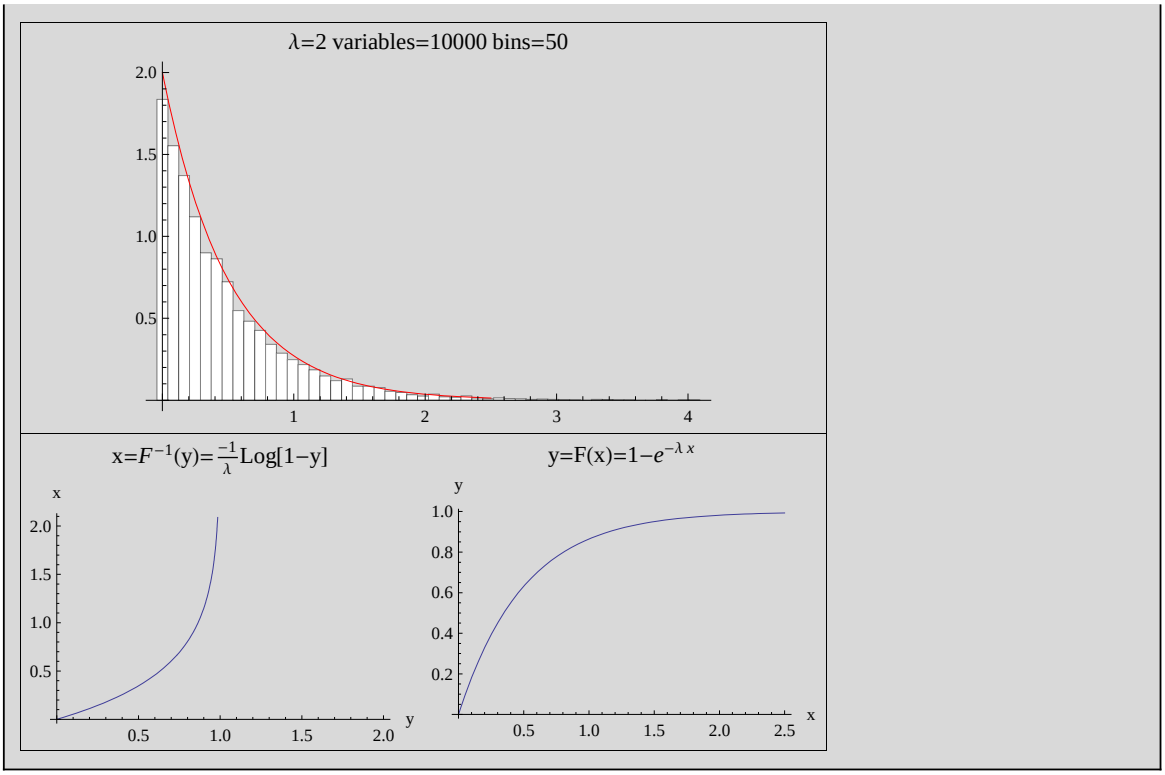
This function to overlay the histogram and the PDF. It is used by the simulation program as well (that is why it is a little larger than needed)

In[162]:=
```
postProcessForPartOne[randomNumbers_, nBins_, λ_, nRandomVariables_, fromX_, toX_] :=
 Module[{frequency, relativeFrequency, p1, p2,
   x, factor, max, imSize = 300, dx, p, pCDF, pinvCDF, gz, g},

  (*find dx which is bin size, needed by Mathematica BinCount function*)
  dx = (toX - fromX) / nBins;

  (*generate frequency count using the above bin size*)
  frequency = BinCounts[randomNumbers, {fromX, toX, dx}];

  (*now normalize by total number of observation to obtain the relative frequencies*)
  relativeFrequency = N[frequency / nRandomVariables];

  (*Now divide by scale factorλ, to scale it *)
  max = Max[relativeFrequency];
  factor = λ / max;
  relativeFrequency = relativeFrequency * factor;

  gz = nmaMakeDensityHistogram[randomNumbers, nBins];
  p1 = GeneralizedBarChart[gz, BarStyle → White,
    ImageSize → imSize, PlotLabel → "λ=" <> ToString[λ] <> " variables=" <>
      ToString[nRandomVariables] <> " bins=" <> ToString[nBins]];

  p2 = Plot[PDF[ExponentialDistribution[λ], x], {x, fromX, toX}
    , PlotRange → All, Frame → True, PlotStyle → {Red(*,Thick*)}, ImageSize → imSize];

  p = Show[{p1, p2}];

  pinvCDF = Plot[inverseCDFofExponentialDistribution[λ, y], {y, 0, λ}
    , PlotLabel → "x=F^-1(y)=-1/λ Log[1-y]", ImageSize → 200, AxesLabel → {"y", "x"}];

  pCDF = Plot[1 - Exp[-λ x], {x, fromX, toX}
    , PlotLabel → "y=F(x)=1-e^-λx", ImageSize → 200, AxesLabel → {"x", "y"}];

  g = Grid[{{pinvCDF, pCDF}}];
  Grid[ { {p}, {g}}, Alignment → {{Center}, {Center}}, Frame → All]
 ]
```

now generate the needed outout for N = 10000

In[163]:=
```
nBins = 50; λ = 2; numberOfVariables = 10 000; fromX = 0; toX = 2.5;
postProcessForPartOne[getRandomNumbersFromExponential[λ, numberOfVariables],
 nBins, λ, numberOfVariables, fromX, toX]
```

Out[164]=



Comment and analysis

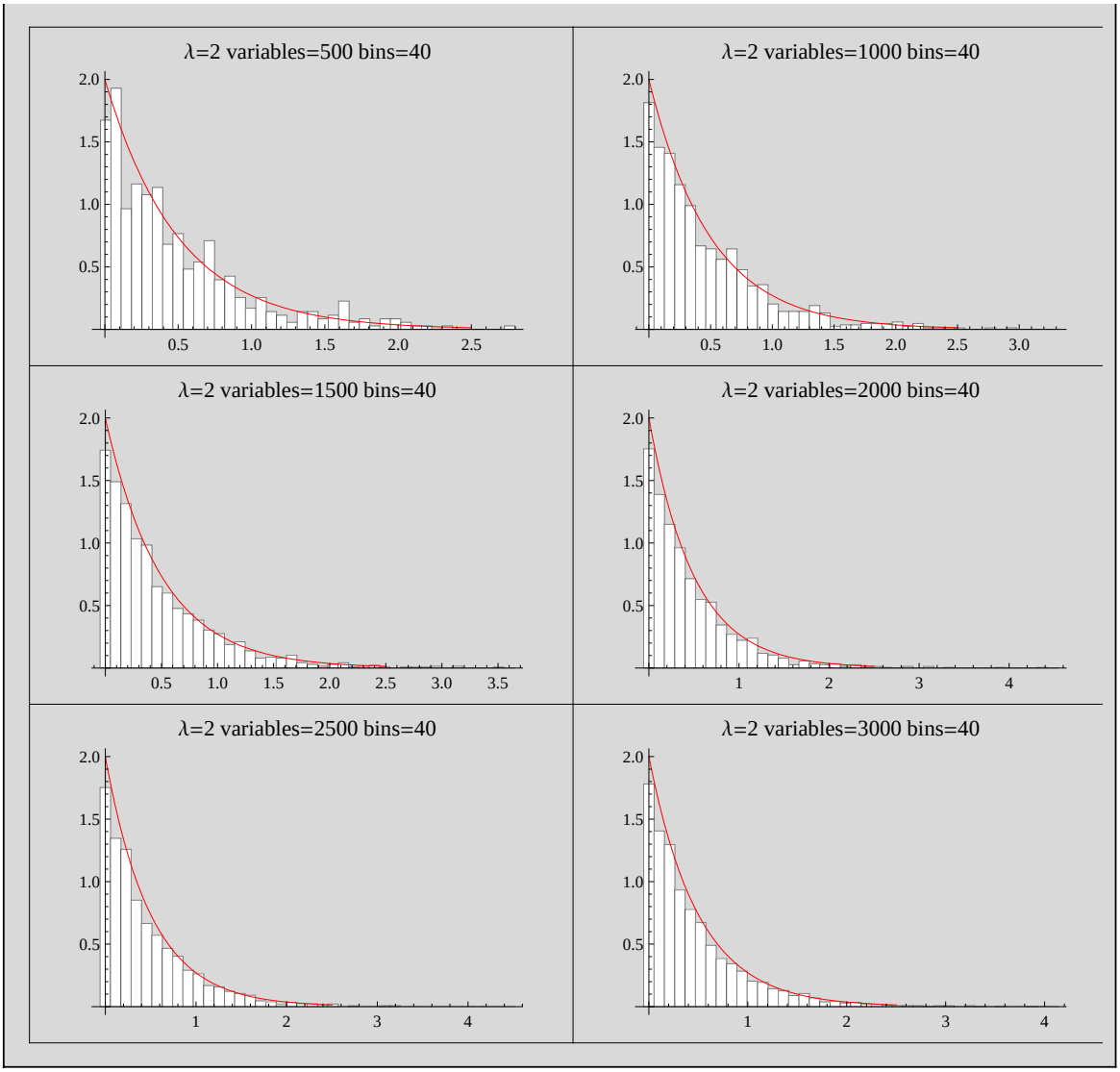
Below I show snap shots of few plots of the density overlaid with the histogram for different values of n which is the number of random variables.

We see from the plots below, that for a fixed number of bins, fixed λ, that as more random variables are generated, the histogram overlaid on top of the actual PDF becomes closer and closer to the PDF curve. The error between the histogram and the PDF curve becomes smaller the larger the number of random variables used. This indicates that this method of finding random numbers for density function will converge to the density function. We need to select an appropriate bin size to see this more clearly. The smaller the bin size the more clear this will become (but too small a bin size will make the histogram itself not too clear).

Please see appendix for additional GUI based simulation for this part of the project.

In[165]:=
```
nBins = 40; λ = 2; fromX = 0; toX = 2.5;
SeedRandom[010 101];
p = Table[postProcessForPartOne[getRandomNumbersFromExponential[λ, nRandomVariables],
    nBins, λ, nRandomVariables, fromX, toX], {nRandomVariables, 500, 6 * 500, 500}];
GraphicsGrid[{ {First[p[[1, 1, 1]]], First[p[[2, 1, 1]]]}, {First[p[[3, 1, 1]]],
         First[p[[4, 1, 1]]]}, {First[p[[5, 1, 1]]], First[p[[6, 1, 1]]]}},
 Frame → All, ImageSize → 600]
```

Out[168]=



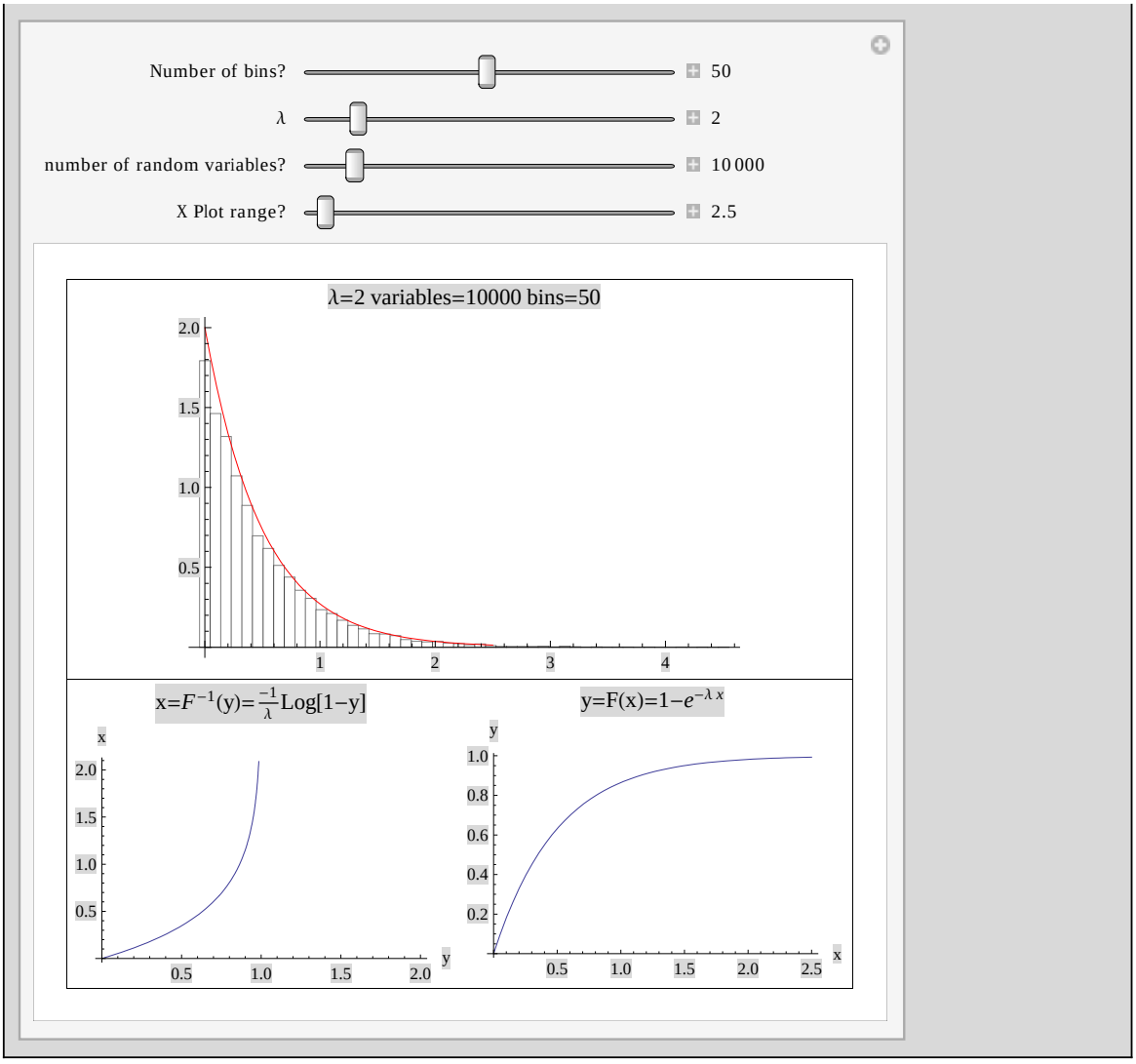
Problem 1 simulation

Define function which accepts a list of random variables from exponential distribution, and λ and generates a plot of the histogram overlaid by the exponential density plot.

In[169]:=

```
m = Manipulate[(SeedRandom[010 101];
    postProcessForPartOne[getRandomNumbersFromExponential[λ, n], nBins, λ, n, 0, maxX]),
  {{nBins, 50, "Number of bins?"}, 1, 100, 1, ContinuousAction → True,
   Appearance → "Labeled"},
  {{λ, 2, "λ"}, 1, 10, .01, ContinuousAction → True, Appearance → "Labeled"},
  {{n, 10 000, "number of random variables?"},
   10, 100 000, ContinuousAction → True, Appearance → "Labeled"},
  {{maxX, 2.5, "X Plot range?"}, 1, 100, 1, ContinuousAction → True, Appearance → "Labeled"},
  AutorunSequencing → {{1, 15}, {2, 20}, {3, 15}}
 ]
```

Out[169]=



0.2.2 problem 2

Project one. Problem 2. Mathematics 502 Probability and Statistics

Nasser Abbasi, September 26,2007. California State University, Fullerton

2. [20 points] Let $\phi_0(x)$ and $\phi_1(x)$ denote the density functions for two normal random variables with $\sigma = 1$, and respective means 0 and 3. The density for a mixture distribution of these is defined as

$$f(x) = \gamma\phi_0(x) + (1-\gamma)\phi_1(x) \quad -\infty < x < \infty.$$

for a given admixture parameter $0 < \gamma < 1$.

a. Write an R program, using the normal random generator **rnorm** and the uniform random number generator **runif**, to generate n random numbers from this mixture. The inputs to your program should be n, and γ. Generate two sets of 10,000 data points; one with $\gamma = .75$, and another with $\gamma = .25$.

b. For each of the data sets generated, graph the density histogram and superimpose it by the density $f(x)$ defined in the equation above. Make sure to choose the number of bins for the histogram appropriately. In each case explain why the shape of the density that you obtain is expected.

Part 2 (a)

The mixed distribution is

$$f(x) = \gamma\phi_0(x) + (1-\gamma)\,\phi_1(x) \quad -\infty \le x \le \infty$$

Where ϕ_0 is the density function for normal distribution with σ=1,μ=0, and ϕ_1 is the density function for normal distribution with σ=1,μ=3

We need to generate random numbers from the above density function.

The following is the idea of how to solve this problem. Let us consider the case for γ=75% .Generate a r.v. from a uniform distribution, which will be between [0, 1]. Let this number be called ζ. If $\zeta < .76$ then now we will generate a random number from the above $\phi_0(x)$ normal distribution otherwise we will generate a random number from $\phi_1(x)$ distribution. Hence this is the algorithm

Algorithm

Input: γ,σ_0, μ_0, σ_1, μ_1, n (where n is number of random number to generate for mixture $f(x)$
output: n random numbers that belong to mixture $f(x)$

seed the random number generator (010101)
Initialize array d of the size of the random numbers generated

For i in 1..n LOOP
  ζ = generate a r.v. from $U[0, 1]$
  IF $\zeta<\gamma$ THEN
    d[i]= generate a random number from $\phi_0 \sim N(\sigma_0, \mu_0)$
  ELSE
    d[i]= generate a random number from $\phi_1 \sim N(\sigma_1, \mu_1)$
  ENDIF
END LOOP

output d which will now contain n random variables drawn from the above probability density function.

Function to implement the mixture random variable algorithm

In[190]:=
```
Remove["Global`*"];
gDebug = False;
```

In[192]:=
```
(*this function below imlements the above algorithm*)
processPartTwo[γ_, μ0_, σ0_, μ1_, σ1_, n_] := Module[{d = {}, k, ζ},
  ζ = Table[RandomReal[], {k, 1, n}];
  d = Table[ If[ζ[[k]] < γ, RandomReal[NormalDistribution[μ0, σ0]],
     RandomReal[NormalDistribution[μ1, σ1]]], {k, 1, n}]
 ]
```

Now generate 2 sets of numbers each 10000 long, one for γ = .75 and the second for γ = .25

In[193]:=
```
μ0 = 0; σ0 = 1; μ1 = 3; σ1 = 1; n = 10 000;
SeedRandom[010 101];
γ = .75;  setA = processPartTwo[γ, μ0, σ0, μ1, σ1, n];
γ = .25;  setB = processPartTwo[γ, μ0, σ0, μ1, σ1, n];
```

Part 2 (b)

Generate 2 plots, one for γ = .75 and one for γ=.25 for number of random variables=10000 generated in part (a) overlaid by histogram.
First define the mixture density function (the true density). Please see appendix for the code that overlays the histogram and the mixture function called postprocessPartTwo[]. Moved below to the appendix to reduce code clutter in the main report.

In[197]:=
```
mixtureDensity[x_, γ_, μ0_, σ0_, μ1_, σ1_] :=
  γ PDF[NormalDistribution[μ0, σ0], x] + (1 - γ) PDF[NormalDistribution[μ1, σ1], x];
```

Now call the above on the 2 sets of 10000 numbers generated in part (a) and display the result

This function makes a histogram which is scaled to be used to overlay density plots, or other functions.
Input: originalData: this is an array of numbers which represents the data to bin

nBins: number of bins

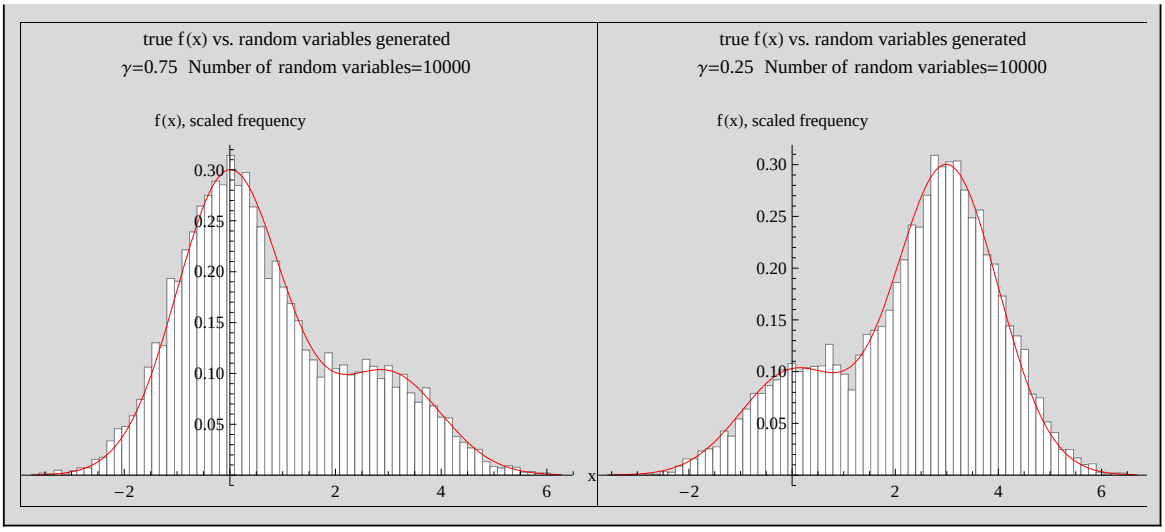
output: the histogram itself but scaled such that area is ONE

In[198]:=
```
Needs["BarCharts`"]
nmaMakeDensityHistogram[originalData_, nBins_] :=
 Module[{freq, binSize, from, to, scaleFactor, j, a, currentArea},
  to = Max[originalData];
  from = Min[originalData];
  binSize = (to - from) / nBins;
  freq = BinCounts[originalData, binSize];
  currentArea = Sum[binSize * freq[[i]], {i, nBins}];
  freq = freq / currentArea;
  a = from;
  Table[{a + (j - 1) * binSize, freq[[j]], binSize}, {j, 1, nBins}]
 ]
```

In[200]:=
```
postprocessPartTwo[d_, γ_, μ0_, σ0_, μ1_, σ1_, nRandomVariables_, imageSize_] :=
 Module[{freq, p, pList, xFrom, xTo, scaleFactor, maxSampled, sampled,
   gz = {}, fz0, fz1, x, maxBin, imSize = imageSize, nBins = 70, from},
  xFrom = Min[d];
  xTo = Max[d];
  gz = nmaMakeDensityHistogram[d, nBins];
  pList = GeneralizedBarChart[gz, BarStyle → White, ImageSize → imSize];

  p = Plot[mixtureDensity[x, γ, μ0, σ0, μ1, σ1], {x, xFrom, xTo},
    AxesOrigin → {0, 0}, PlotRange → All, PlotLabel → "Analytical plot of f(x)",
    ImageSize → imSize, (*PlotStyle→{Dashed,Red}*)PlotStyle → {Red}];

  Show[{pList, p},
   PlotLabel → Style["true f(x) vs. random variables generated\n" <> "γ=" <> ToString[γ] <>
      "  Number of random variables=" <> ToString[nRandomVariables] <> "\n", 10],
   AxesLabel → {"x", "f(x), scaled frequency"}]
 ]
```

In[201]:=
```
γ = .75;  p1 = postprocessPartTwo[setA, γ, μ0, σ0, μ1, σ1, n, 300];
γ = .25;  p2 = postprocessPartTwo[setB, γ, μ0, σ0, μ1, σ1, n, 300];
Grid[{{p1, p2}}, Frame → All, Spacings → 0, ItemSize → Full]
```

Out[203]=



Comment and analysis on result of part 2 (b) plots

In the left plot, γ=.75, hence 75% of the mixture comes from ϕ_0 which has a mean of 0, hence we would expect that at zero the bulk of the concentration, which what the plots shows to be the case (since both concentration have the same variance). Hence there should be more random numbers generated from this mixture around x=0 as well, and we see from the histogram that this is the case.
In the right plot, now γ=.25, hence 75% of the concentration will come the ϕ_1 distribution which has mean of 3. Hence again, we see that more random numbers are generated around 3 than anywhere else. These plots also show that the random numbers generated will have a probability density which will converge to the $f(x)$ given as more and more random variables are generated.

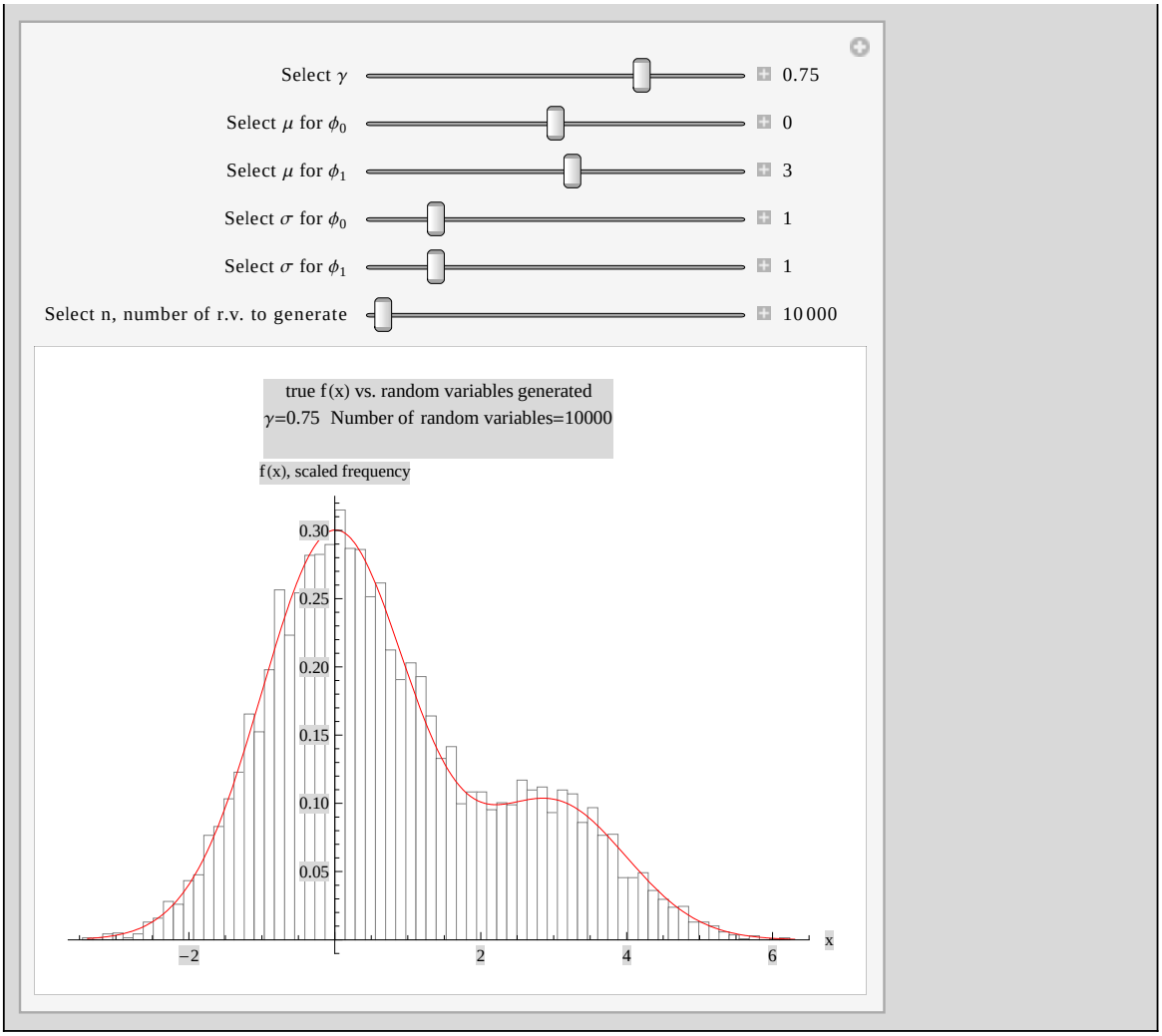
Simulation program for problem 2

GUI to simulate part 2(b) of the project

In[204]:=

```
m = Manipulate[
  postprocessPartTwo[processPartTwo[γ, μ0, σ0, μ1, σ1, n], γ, μ0, σ0, μ1, σ1, n, 400],
  {{γ, .75, "Select γ"}, 0, 1, .1, Appearance → "Labeled"}
  , {{μ0, 0, "Select μ for ϕ0"}, -30, 30, .1, Appearance → "Labeled"}
  , {{μ1, 3, "Select μ for ϕ1"}, -30, 30, .1, Appearance → "Labeled"}
  , {{σ0, 1, "Select σ for ϕ0"}, 0.1, 6, .1, Appearance → "Labeled"}
  , {{σ1, 1, "Select σ for ϕ1"}, 0.1, 6, .1, Appearance → "Labeled"}
  , {{n, 10 000, "Select n, number of r.v. to generate"}, 10 000, 100 000, 10 000,
   Appearance → "Labeled"}, AutorunSequencing → {{1, 15}, {2, 25}, {3, 25}, {4, 25}, {5, 25}}
 ]
```

Out[204]=



In[205]:=

```
(*Export["m.swf",m,"RepeatAnimation"→True,"CompressionMethod"->None]*)
```

Mathematica notebook

0.2.3 problem 3

Project one. Problem Three. Mathematics 502 Probability and Statistics

Nasser Abbasi, September 26,2007. California State University, Fullerton

3. [30 points] Problem 42 on page 111 of your text gives the pdf for the double exponential density with parameter λ. It also suggests a method to generate random numbers from the double exponential family using two random variables W and T, described in the problem.

a. Write a program that generates random numbers from the double exponential family. The input to the program should be the parameter λ, and the number of random numbers to be generated, n. The output should be n pseudo random numbers from the double exponential with parameter λ. You are only allowed to use `runif` in your program for random number generation.

b. Write a program that uses the Accept/Reject algorithm efficiently to generate n observations from the standard normal density $N(0,1)$, using random numbers that are generated from the uniform(0,1) and the double exponential random variate with parameter $\lambda = 1$ [your program in part (a)]. Your program should also count and report the proportion of values that are rejected. Give the density histogram of $n = 10,000$ numbers generated from your program and superimpose it by the standard normal pdf.

Problem 3 part (a)

We are asked to generate R.V's from $f(x) = \frac{\lambda}{2} e^{-\lambda|x|}$. We note as shown in the problem itself, that R.V. X can be written as product of 2 R.V WT where $W$is ± 1 with probability $\frac{1}{2}$each. Hence to generate R.V. we do the following. We generate n R.V. from uniform distribution [0,1] using Mathematica random number generator. Then we check if each number is $< \frac{1}{2}$ or not, and we generate 1 or -1 as the case may be. We then generate n random variables from the exponential distribution, which we know how to do from part (a). Then we multiply the above 2 vectors, element wise, with each others. The first vector being the vector of 1's and -1's. And the second vector being the RV's from the exponential distribution. This is the algorithm

Algorithm
Input: λ,n (number of random variables to generate)
output: list of random numbers which belong to density $f(x) = \frac{\lambda}{2} e^{-\lambda|x|}$

Seed the random number generator with unique value for us.
A = Generate n random numbers from the exponential distribution with parameter λ (CALL problem 1 part(a) with the input λ,n) This uses F^{-1}method and uniform random number generator as well.
B = Generate n random numbers from uniform random number generator [0,1]

```
FOR i in 1..n LOOP  -- Note: This is algorithm view. In code 'vectored' operation is used.
      IF B(i)<.5 THEN
        B(i) = 1
      ELSE
        B(i) = -1
      END IF
END LOOP

result = B * A
```

Now generate a histogram from the result above.
The following function implements the above algorithm

Code Implementation

Define the function F^{-1}which was derived earlier. This is the inverse of the CDF of the exponential density function $\lambda e^{-\lambda x}$

```
Remove["Global`*"];
gDebug = False;
```

```
inverseCDFofExponentialDistribution[λ_, n_] := Module[{}, -1/λ Log[1 - n]]
```

Function below is called to generate N random numbers using the above F^{-1} function (User needs to seed before calling

```
getRandomNumbersFromExponential[λ_, nRandomVariables_] := Module[{i},
  Table[ inverseCDFofExponentialDistribution[λ, RandomReal[]], {i, nRandomVariables}]]
```

```
getRandomNumbersFromDoubleExponential[λ_, numberOfRandomVariables_] := Module[{W, T},
  W = getRandomNumbersFromExponential[λ, numberOfRandomVariables];
  T = Table[If[RandomReal[] < .5, 1, -1], {i, numberOfRandomVariables}];
  W T]
```

Test the above function by plotting the histogram generated for say $n = 10\,000$ overlaid by the true double exponential density function.
First, define the double exponential function

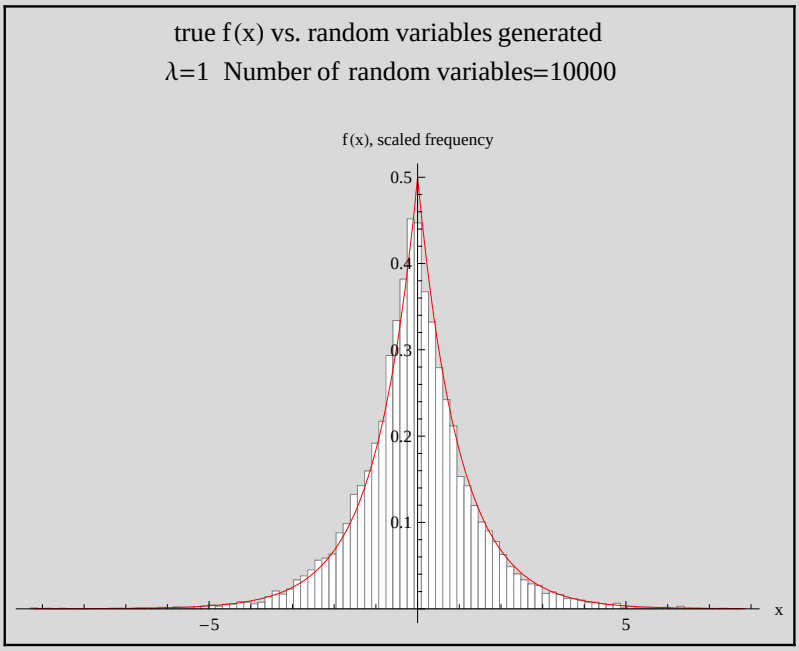
```
doubleExponential[λ_, x_] := λ/2 Exp[-λ Abs[x]]
```

Now do the overlay plot

This function makes a histogram which is scaled to be used to overlay density plots, or other functions.
Input: originalData: this is an array of numbers which represents the data to bin
nBins: number of bins
output: the histogram itself but scaled such that area is ONE

```
Needs["BarCharts`"]
nmaMakeDensityHistogram[originalData_, nBins_] :=
 Module[{freq, binSize, from, to, scaleFactor, j, a, currentArea},
  to = Max[originalData];
  from = Min[originalData];
  binSize = (to - from) / nBins;
  freq = BinCounts[originalData, binSize];
  currentArea = Sum[binSize * freq[[i]], {i, nBins}];
  freq = freq/currentArea;
  a = from;
  Table[{a + (j - 1) * binSize, freq[[j]], binSize}, {j, 1, nBins}]
 ]
```

```
SeedRandom[010 101];
n = 10 000; λ = 1; nBins = 100; imageSize = 400;

postprocessPartThreeA[listOfRandomNumbers_, λ_, nBins_, imageSize_] :=
 Module[{gz, pList, xFrom, xTo},
  xFrom = Min[listOfRandomNumbers];
  xTo = Max[listOfRandomNumbers];
  gz = nmaMakeDensityHistogram[listOfRandomNumbers, nBins];
  pList = GeneralizedBarChart[gz, BarStyle → White, ImageSize → imageSize];
  p = Plot[doubleExponential[λ, x], {x, xFrom, xTo}, AxesOrigin → {0, 0}, PlotRange → All,
    ImageSize → imageSize, (*PlotStyle→{Dashed,Red}*)PlotStyle → {Red}];

  Show[{pList, p},
   PlotLabel → Style["true f(x) vs. random variables generated\n" <> " λ=" <> ToString[λ] <>
      "  Number of random variables=" <> ToString[Length[listOfRandomNumbers]] <> "\n", 14],
   AxesLabel → {"x", "f(x), scaled frequency"}]
 ]

Framed[
 postprocessPartThreeA[getRandomNumbersFromDoubleExponential[λ, n], λ, nBins, imageSize]]
```



Problem 3 part(b)

In this part, we need to generate a list of r.v's that belong to normal distribution N(0,1), using uniform random number generator U[0,1] and using the random numbers generated from the double exponential density function in part (a) above. We are asked to use the accept/reject method.
First the method is explained, then the algorithm outlined, then the implementation shown and a test case given, then a GUI interface written to test the algorithm for different parameters values.

Accept/Reject algorithm
input: n (number of random variables to generate)
λ (the exponential density parameter)
$f(x)$ the density function for random variable X which we wish to generate random variables
f_M(x) the density which we will use to help in generating the random variables from f_X(x). This density is such that it is easy to generate random variables from. Much easier that from f(x) and that is why it was selected.
output: list of random numbers of length n from $f(x)$

Step 1: Find C. Where $c = \sup \forall x \frac{f_X(x)}{f_M(x)}$ To solve this, this is the algorithm

Algorithm for step 1: Let $f_M(x) = \frac{\lambda}{2} e^{-\lambda x}$ (since double exponential is symmetric, I'll use one sided version). Let $f_X(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$. Now find the ratio $r(x) = \frac{f_X(x)}{f_M(x)} = \frac{\frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}}{\lambda e^{-\lambda x}}$ Now find where the maximum of this ratio is using normal calculus method: Take the derivative w.r.t. x and set it to zero. Solve the resulting equation for x. Evaluate the ratio at this root. This gives C. We find that $C = 1.31549$ The following few lines of code finds C:

```
λ = 1;  fm = λ/2 Exp[- x];  fx = PDF[NormalDistribution[0, 1], x];
ratio = fx/fm;
root = First@Solve[D[ratio, x] == 0, x];
c = N[ratio /. root]
```

```
1.31549
```

Step 2: Now that we found C in step 1, then the envelop function becomes $C * f_M(x) = C \frac{\lambda}{2} e^{-\lambda|x|}$

Step 3: seed the number random generator

initialize an array d of size *n* to contain all the accepted random numbers generated

initialize counter number_accepted=0

WHILE number_accepted < n DO

generate r.v. from U[0,1] call it *u*.

Generate r.v. from double exponential density (using part(a)) call this x

IF $u * C * f_M(x) < f_X(x)$ THEN

d[i]=x

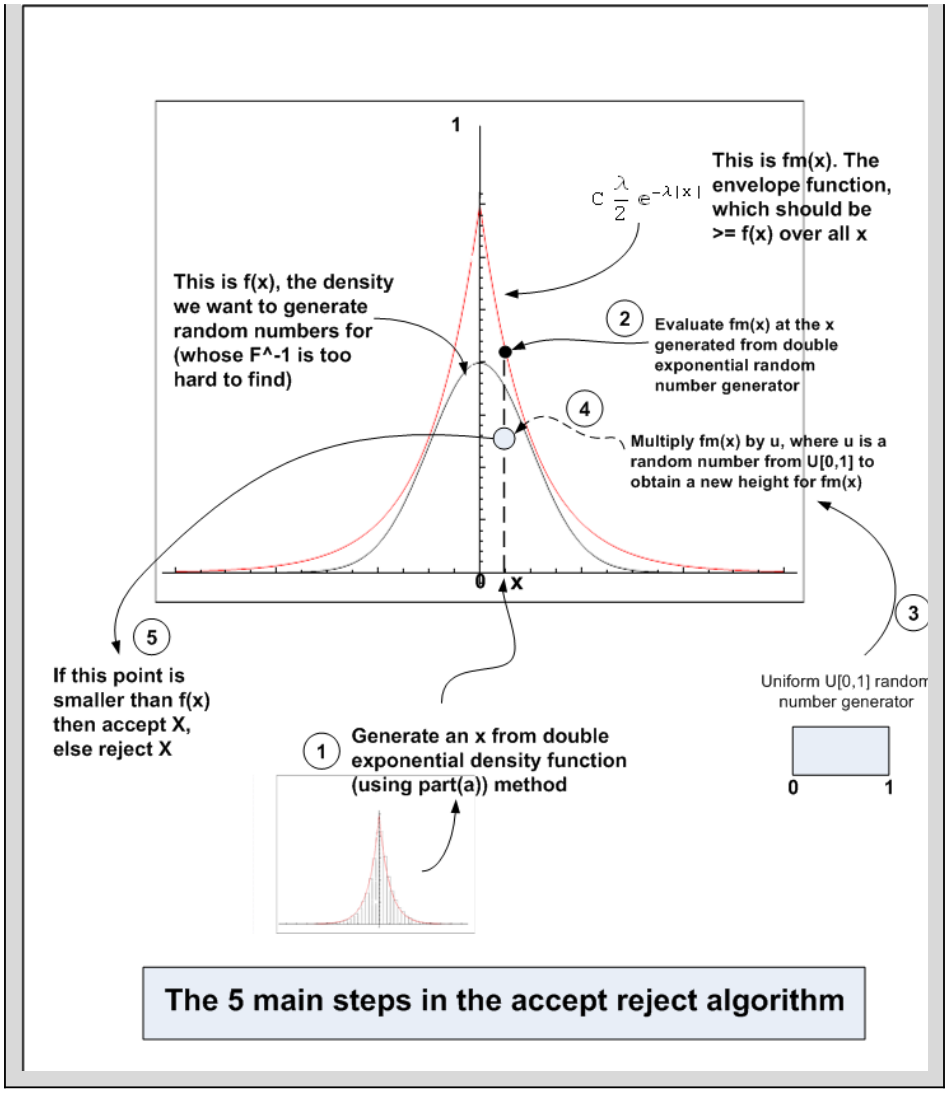
number_accepted++

END IF

END LOOP

Step 4: Now array d contains the n random numbers generated from the normal density N[0,1]. Make histogram and overlay it over N[0,1]

Diagram showing main steps in the algorithm



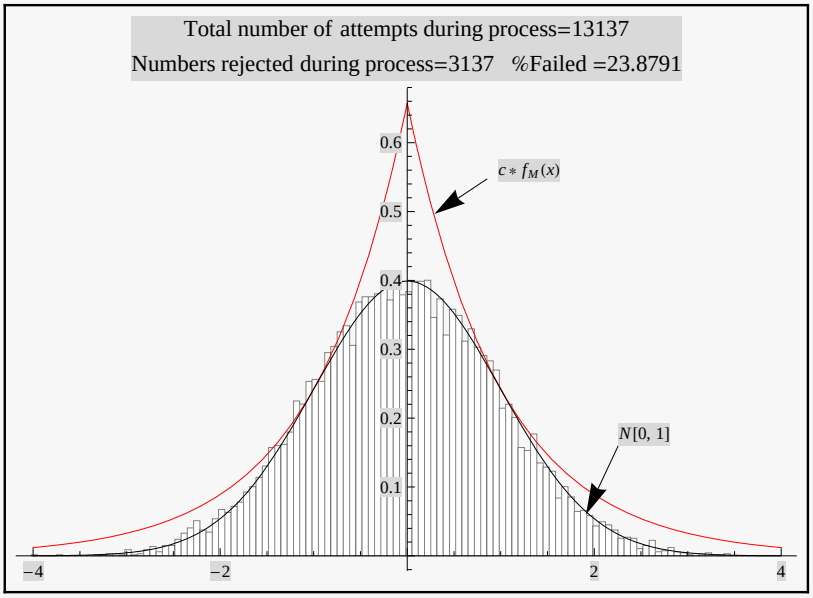
Accept Reject Algorithm Implementation

```
acceptReject[λ_ , numberOfRandomNumbersToGenerate_ , c_
  (*This is for scaling the envelope with so that envelope ≥ f(x) everywhere*)
  , μ_ (*mean of Normal Dist*), σ_ (*std of normal dist*)
 ] := Module[{nFailed = 0, nPassed = 0, y, x, d, i, maxEnvelope, fx, u},
  RandomSeed[010 101]; (*start from clean random number generator*)
  maxEnvelope = c * doubleExponential[λ, 0];
  d = Table[0, {i, numberOfRandomNumbersToGenerate}];

  While[nPassed < numberOfRandomNumbersToGenerate,
   {x = getRandomNumbersFromDoubleExponential[λ, 1][[1]];
    y = c * doubleExponential[λ, x] * RandomReal[{0, 1}];
    fx = PDF[NormalDistribution[μ, σ], x];
    If[y ≤  fx, {nPassed++, d⟦nPassed⟧ = x}, nFailed++];}
  ];

  {d, nFailed}
 ]
```

Test case for n=10,000

Test the above function, and make a plot of histogram overlaid on top of density of N(0,1)

```
λ = 1; μ = 0; σ = 1; xFrom = -4 σ; xTo = 4 σ; n = 10 000;
c = 1.315489246958914; (*see algorithm above on how C was found*)
nBins = 120;
Clear[x];

{listOfNumbers, nFailed} = acceptReject[λ, n, c, μ, σ];

gz = nmaMakeDensityHistogram[listOfNumbers, nBins];
pList = GeneralizedBarChart[gz, BarStyle → White, ImageSize → 400, PlotRange → All];

p = Plot[{c * doubleExponential[1, x], PDF[NormalDistribution[0, 1], x]},
   {x, xFrom, xTo}, PlotRange → All, PlotStyle → {Red, Black}, ImageSize → 400];

Framed[Show[{pList, p},
  PlotLabel → "Total number of attempts during process=" <> ToString[n + nFailed] <>
    "\nNumbers rejected during process=" <> ToString[nFailed] <> "   %Failed =" <>
    ToString[nFailed / (n + nFailed) * 100.]]]
```



The above is a plot showing the histogram for random numbers generated using the accept - reject method for N=10,000. The random numbers are very close the N[0,1] which indicates this method is working well. The larger N is, the more closely the random numbers histogram will approach N[0,1] probability density.

I have implemented a GUI based simulation as well for the above problem, please see the appendix below to run the simulation part.

Problem 3 simulation

```
Module[{gnTrialsSoFar = 0, gnRejectSoFar = 0, gnAcceptedSoFar = 0, gmaxEnvelope, gmultiplier,
  gλ, gμ, gσ, gAcceptedXSet, gnBins, gxFrom, gxTo, gAcceptedPointsCoordinates, gMaxAccepted},

 initializeSimulation[] := Module[{},
   RandomSeed[010 101];
   gmultiplier = 1.315489246958914;
   gnBins = 40;
   gλ = 1;
   gμ = 0;
   gσ = 1;
   gxFrom = -6 gσ;
   gxTo = 6 gσ;
   gMaxAccepted = 10 000;
   gnTrialsSoFar = 0; gnRejectSoFar = 0; gnAcceptedSoFar = 0;
   gAcceptedXSet = Table[0, {i, gMaxAccepted}];
   gAcceptedPointsCoordinates = Table[0, {i, gMaxAccepted}];
   gmaxEnvelope = gmultiplier * doubleExponential[gλ, 0]
  ];

 finalizeSimulation[] := Module[{gz, p, pList, x, res},
   If[gnTrialsSoFar > 0,
    {
     gz = nmaMakeDensityHistogram[gAcceptedXSet[[1 ;; gnAcceptedSoFar]], gnBins];

     pList = GeneralizedBarChart[gz,
       BarStyle → White,
       ImageSize → 250,
       PlotRange → {{-4.5 gσ, 4.5 gσ}, {0, 1.2}}];

     p = Plot[PDF[NormalDistribution[gμ, gσ], x],
       {x, gxFrom, gxTo},
       PlotStyle → Red,
       PlotRange → All];

     res = Show[{pList, p},
       PlotLabel → "Total number of attempts during process=" <> ToString[gnTrialsSoFar] <>
         "\nNumbers accepted during process=" <> ToString[gnAcceptedSoFar] <>
         "   %Accepted =" <>
         ToString[gnAcceptedSoFar / (gnTrialsSoFar) * 100.] <>
         "\nNumbers rejected during process=" <>
         ToString[gnRejectSoFar] <> "   %Failed =" <>
         ToString[gnRejectSoFar / (gnTrialsSoFar) * 100.]
      ];
    }
    ,
    res = "Ready...";
   ];
   res
  ];

 processOneAcceptReject[] := Module[{x, y, fx, res, p, accepted, p2, pStats},
   gnTrialsSoFar++;
   If[gnTrialsSoFar < 10, Return["Ready.."]];

   x = getRandomNumbersFromDoubleExponential[gλ, 1][[1]];
```

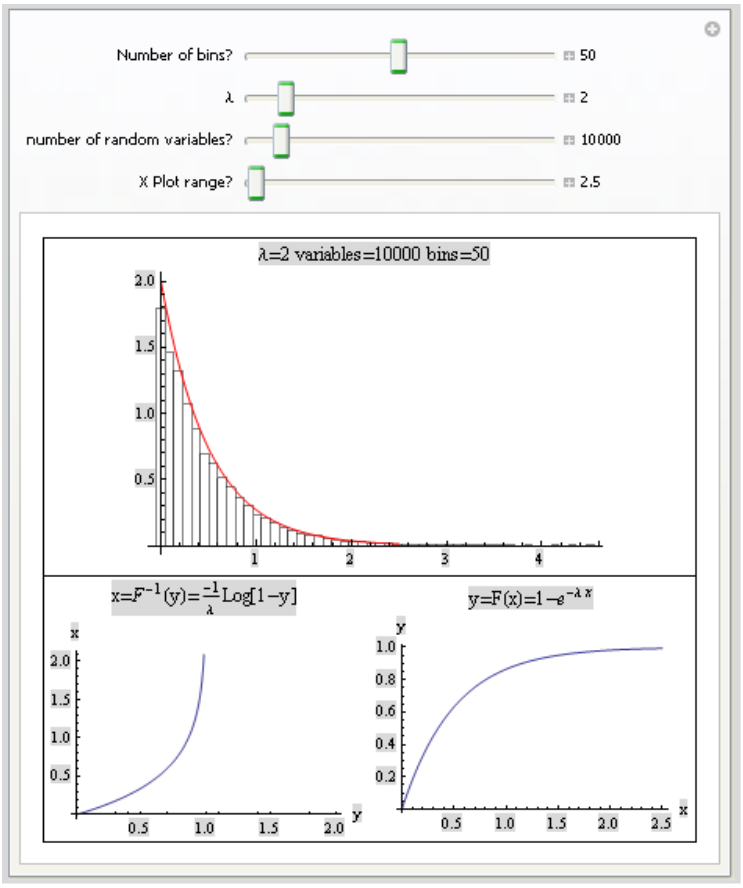
```
   y = gmultiplier * doubleExponential[gλ, x] * RandomReal[{0, 1}];
   fx = PDF[NormalDistribution[gμ, gσ], x];
   If[y ≤ fx
    , {gnAcceptedSoFar++;
     accepted = True;
     gAcceptedXSet⟦gnAcceptedSoFar⟧ = x;
     gAcceptedPointsCoordinates⟦gnAcceptedSoFar⟧ = {x, y}
    },
    {gnRejectSoFar++, accepted = False}
   ];

   p = Plot[{gmultiplier * doubleExponential[1, x], PDF[NormalDistribution[0, 1], x]},
     {x, gxFrom, gxTo}, PlotRange → All, PlotStyle → {Red, Black}, ImageSize → 250,
     PlotLabel → Row[{"Trial [", gnTrialsSoFar, "]\tc=[", gmultiplier, "]\n", If[y ≤ fx,
         Style["Accepted", Black], Style["Rejected", Red]], "\tPoint=(", x, " , ", y, ")"}],
     Epilog → {If[accepted, {PointSize[Large], Green, Point[{x, y}],
         {PointSize[Small], Gray, Point[gAcceptedPointsCoordinates⟦1 ;; gnAcceptedSoFar⟧]}},
        {PointSize[Large], Red, Point[{x, y}], {PointSize[Small], Gray,
          Point[gAcceptedPointsCoordinates⟦1 ;; gnAcceptedSoFar⟧]}}
       ]
      }
    ];
   p2 = finalizeSimulation[];
   pStats = Row[{"Trial [", gnTrialsSoFar, "]\n", If[y ≤ fx,
       Style["Accepted", Black], Style["Rejected", Red]], "\tPoint=(", x, " , ", y, ")"}];
   (*Grid[{ {pStats},{Grid[{ {p,p2}  }]}},Frame→All,Alignment→{Center}];*)
   Grid[{ {p, p2} }, Frame → All, Alignment → {Center}]
  ]
]
```

```
m = Manipulate[res = "Ready to run..."; runIt = False; i = 0;
  Dynamic[
   If[runIt && Not[stopIt] && i < 10 000,
    (i++; res = processOneAcceptReject[]),
    res
   ]
  ],
  {{runIt, True, ""}, Button[Style["Click to start", 10], {i = 0;
      initializeSimulation[]; stopIt = False; runIt = True}] &, ContinuousAction -> False},
  {{stopIt, False, ""}, Button[Style["Click to stop", 10], {stopIt = True; res}] &,
   ContinuousAction -> False}, AutorunSequencing → {{2, 120}}
 ]
```

Mathematica notebook

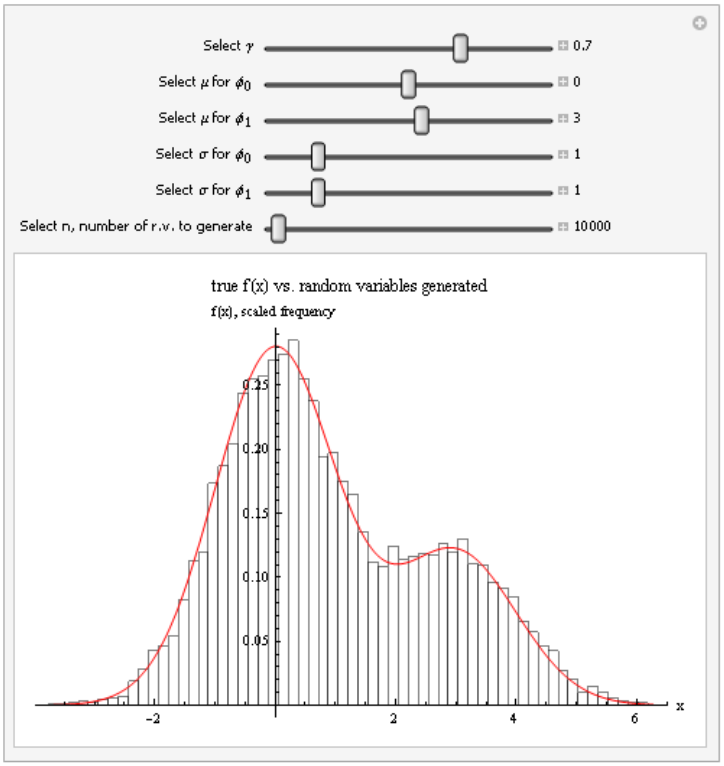
0.3 Simulation movies

0.3.1 Simulation problem 1. Random variables from exponential distribution using F^{-1} method



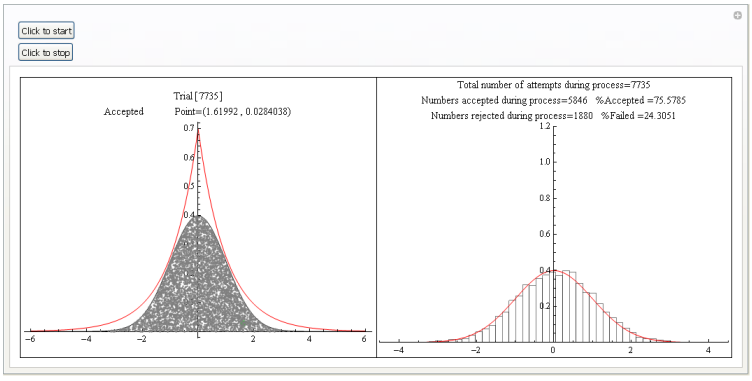
Movie swf

0.3.2 Simulation problem 2 (Random numbers from mixture)



Movie swf

0.3.3 Simulation Problem 3 (Accept/Reject)



Movie swf