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

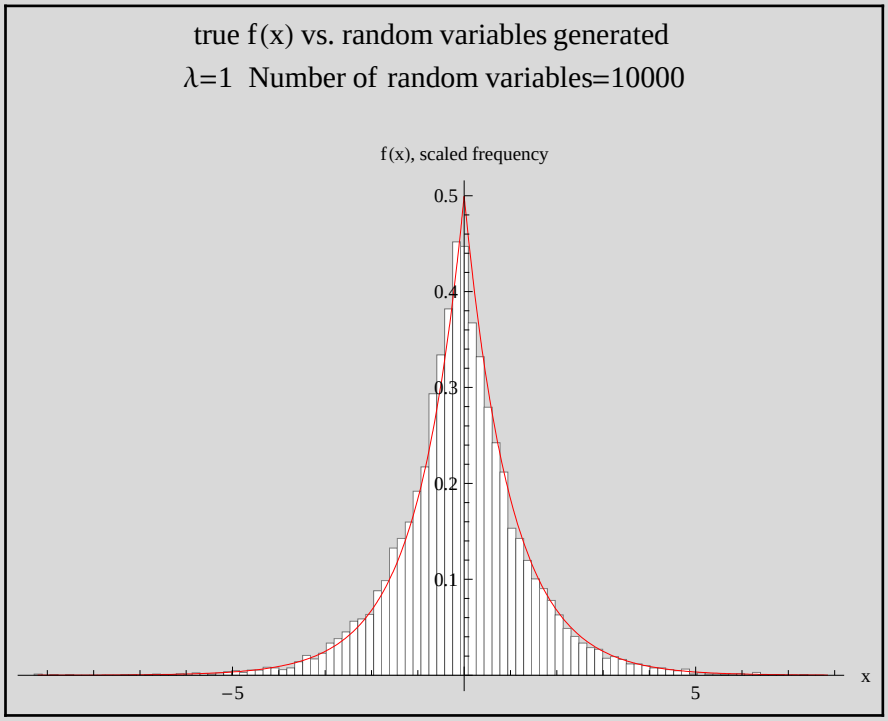
```
Needs["BarCharts`"]
nmaMakeDensityHistogram[originalData_, nBins_] :=
 Module[{freq, binSize, from, to, scaleFactor, j, a, currentArea},
  to = Max[originalData];
  from = Min[originalData];
  binSize = (to - from) / nBins;
  freq = BinCounts[originalData, binSize];
  currentArea = Sum[binSize *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

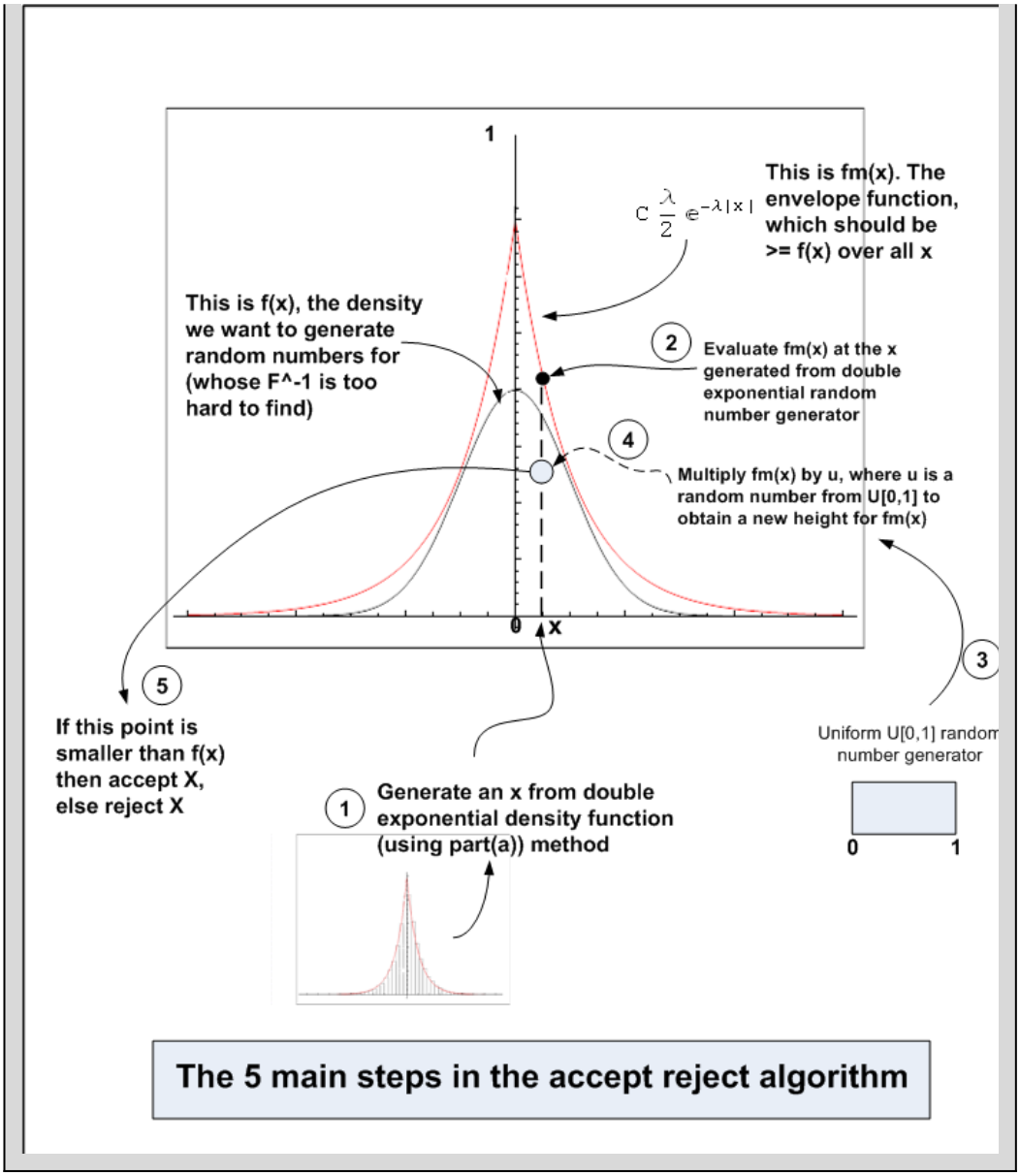
number_accepted++

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)

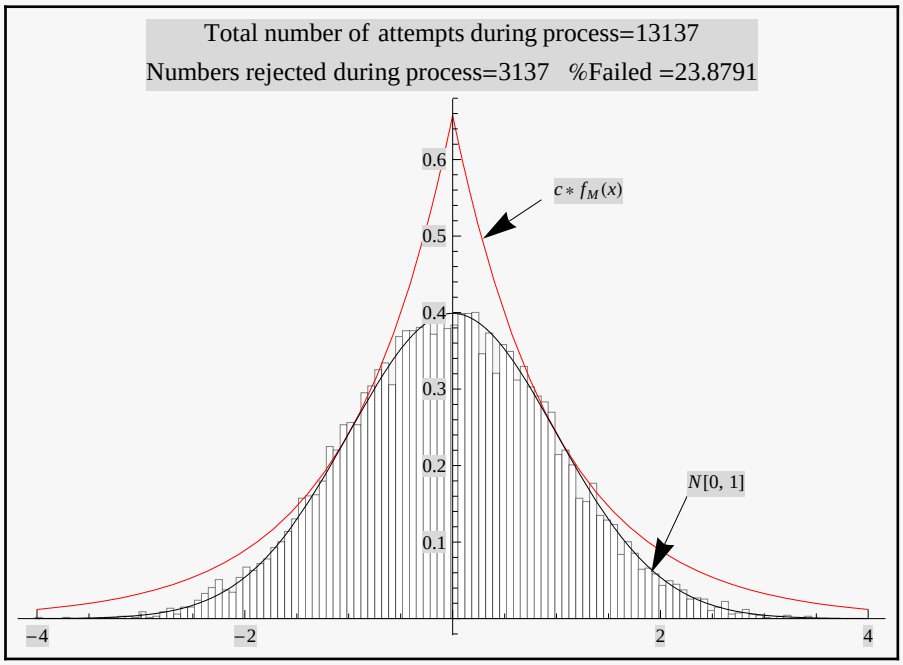
```
λ = 1; μ = 0; σ = 1; xFrom = -4 σ; xTo = 4 σ; n = 10 000;
c = 1.315489246958914; (*see algorithm above on how C was found*)
nBins = 120;
Clear[x];

{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}}
 ]
```