Midterm Project Report

The Ziggurat Algorithm

Summary

The Ziggurat Algorithm was first developed by George Marsaglia in the year 1960. The Ziggurat algorithm is a special case of rejection sampling. In the rejection sampling method, we assume a large rectangle that covers entire pdf region with a large part of the rectangle falling outside the PDF boundary [4]. The disadvantages of this method are that a large value of samples will be rejected, and g(x) needs to be calculated as the samples will be either accepted or rejected based on the condition y<g(x)[4]. Whereas in Ziggurat Algorithm, these two shortcomings are addressed by covering the PDF with a series of rectangles [4]. These rectangles are all the same size and fit into to the PDF in such a way that there is very small area of the rectangle outside the PDF[4].

In this project, Ziggurat algorithm has been implemented for Burr Distribution. The Burr XII distribution or just Burr was first introduced by Irving W. Burr in 1942 and it comes in several forms [1]. The general form of writing the Burr Distribution is as follows:

$$f(x; c, k) = ck\frac{x^{c-1}}{(1+x^c)^{k+1}} \text{ [2][3]}$$

The Burr Distribution assigned to me is:

$$f(x; 1,2) = 1/(1+x)^3 \text{ [2][3]}$$

When c=1, this distribution is called as Pareto Type II distribution or Lomax Distribution. The Burr distribution is most used to model US household income.

Code

```
%Task 1 Find the constant c so that your PDF is normalized correctly
%f_x is the Burr Distribution. To find c, integrate f_x from 0 to Infinity
%and solve for c
f_x = @(x)1./(1+x).^3; %Burr function
c = 1./integral(f_x, 0 ,Inf) %numerical integration and solved for c


%Task 2 Write a function that computes the CDF of your distribution. f_1 is
%the normalized for of f_x. To find the CDF F_X, f_1 must be integrated
%from 0 to x.
syms x
f_1 =@(x) c./(1+x).^3; %Normalized form of f_x
F_X = int(f_1,0,x) %CDF calculation


%Task 3 Implement a baseline sampling algorithm via the transformation S = F?1S(U) where U ?U(0, 1).
tic %start of timer to calculate the time taken to generate baseline samples
rng(0); %to random numbers from seed 0
```

```
nsamples = 1000; %Initializing number of samples
u = rand(nsamples,1); %u is a vector which contains 1000 random samples all between 0 and 1
y = (1./sqrt(1-u))-1; %(1./sqrt(1-u))-1 is the Inverse function of CDF F_X. y contains 1000 samples from the inverse
function
toc %end of timer to calculate the time taken to generate baseline samples


%Task 4, PDF estimation
del = 0.1; %delta value, a small range for displacement
ymin = min(y); %minimum value from the baseline samples
ymax = max(y); %maximum value from the baseline samples
bincenters = ymin:del:ymax; %a vector containing all the values from ymin to ymax with an interval of del
pdf_est = zeros(1,length(bincenters)); %initializing the vector pdf_est to the same size as that of bincenters
for i=1:length(bincenters) %each value of bincenters must be compared
    pdf_est(i) = nnz(y>bincenters(i)-del/2 & y<=bincenters(i)+del/2)/nsamples/del; %calculating the number of
nonzeros between
end %bincenters(i)-del/2 bincenters(i)+del/2. Probability_est = count/nsamples. PDF est = probability_est/del
x_real = linspace(ymin,ymax,1000); %create a new vectore with 1000 values between ymin,ymax linearly
pdf_real = c./((1+x_real).^3); %calculating the pdf value for each value from vector x_real
plot(bincenters, pdf_est,x_real,pdf_real,'LineWidth',2); %ploting 2 graphs together
title('PDF estimate of baseline sampling VS Real PDF') %Title for the plot
xlabel('x') %x-axis lable
ylabel('PDF') %y-axis lable


%Task 5 generating x1,x2,x3...xn and y1,y2,y3...yn
tail = 0; %Tail area initialization
Amin = 0; %Amin initialization for bisection(guess)
Amax = 10; %Amax initialization for bisection(guess)
tol = 1e-13; %tolerance initialized to 10^-12
prompt = 'Please choose the number of regions. It can be either 4, 32 or 256\n'; %Prompt the user to provide an
input
n = input(prompt); %accept the input for n
xvals = zeros(n,1); %initialize array xvals to store values of x
yvals = zeros(n,1); %initialize array yvals to store values of y
xvals(1) = 0; %assign xvals(1)=x(1) according to the given distribution
Area_nth_Region = 0; %Area of rectangle obtained in the inner loop
while Amax-Amin>tol %run until Amax-Amin<=10^-12
    A = (Amax+Amin)./2; %middle value between Amax and Amin(bisection outer loop)
    for k = 1:(n-1) %must repeat n times to obtain values from 1-n
        xmax = 45; %initialize xmax (guess)
       xmin = 0; %initialize xmin (guess)
        while xmax - xmin > tol %run until xmax-xmin<=10^-12
            xvals(k+1) = (xmin+xmax)./2; %middle value between xmax and xmin(bisection inner loop)
            yvals(k) = 1./(1+xvals(k)).^3; %calculate y(k)
            yvals(k+1) = 1./(1+xvals(k+1)).^3; %calculate y(k+1)
            Area_rect = (xvals(k+1)-xvals(1)).*(yvals(k)-yvals(k+1)); %inner loop area calculation
            if Area_rect > A %check if inner loop area greater or lesser than guessed area
                xmax = xvals(k+1); %guessed x value too small. Increase xmax value to increase x
            else
                xmin = xvals(k+1); %guessed x value too big. Decrease xmin value to decrease x
            end

        end
    end
    tail = integral(f_x,xvals(n),Inf); %calculating area of the tail in nth region by integrating from xn to infinity
```

```
    Area_nth_Region = ((xvals(n)-xvals(1)).*(yvals(n)))+tail; %total area of nth region
    if Area_nth_Region > A %compared area on nth region with guessed area
        Amin = A; %Guessed area too small. Increase Amin to increase A
    else
        Amax = A; %Guessed area too big. Decrease Amax to decrease A
    end
end
disp("Guessed Area = "),disp(A);
disp("Area of the nth region = "), disp(Area_nth_Region);
disp("xvals = "), disp(xvals);
disp("yvals = "), disp(yvals);


%Task 6 Probability of a point lying in the rectangle of nth region
nth_Rect_Area = Area_nth_Region - tail; %calculating the area of the rectangle of nth region
P = nth_Rect_Area/Area_nth_Region; %probability of x being inside the rectangle


%Task 7 Sampling the tail part on nth region
f_t = @(x) 1./(1+x).^3; %initialize f_t to burr function. To indicate it is for tail sampling
c_1 = 1./integral(f_t, xvals(n), Inf); %Find the normalization constant. Differes for each n.
f_t_normalized = @(x) c_1./(1+x).^3; %Normalized form of f_t
F_T = int(f_t_normalized, xvals(n), x); %CDF of the tail
rng('shuffle'); %to change the seed
xsamples = 10000; %Only one value is required. This is created for testing purpose.
A = rand(xsamples, 1); %array of random numbers between 0 and 1
if(n == 4) %if n = 4, create samples using the formula right below. It is the inverse of F_T
    tail_sampling =
sqrt(6822280381494641./(1125899906842624.*((27432874449925277234215308492 8./274328744499252784678716781927)-A)))-1;
elseif(n == 32) %if n = 32, create samples using the formula right below. It is the inverse of F_T
    tail_sampling =
sqrt(2863200573390867./(35184372088832.*((257894180708201982120912633200 64./2578941807082019550781 2898789009)-A)))-1;
else %if n = 256, create samples using the formula right below. It is the inverse of F_T
    tail_sampling =
sqrt(1622492499733937./(2199023255552.*((14614113234426366507245847445504./146141132344263676209 01327224961)-A)))-1;
end


%Task 8 Complete the implementation of the Ziggurat algorithm for n = 4, n= 32, and n = 256 and Task 10 to track and analize the data
Count_of_X_less_than_Xk = 0; %to keep track of (a) X is accepted because X < xk,
Count_of_X_greater_than_Xk_but_Y_less_than_gX = 0; %To keep track of (b) X > xk but X is accepted because Y < g(X),
Count_of_Y_greater_than_gX = 0; %To keep track of (c) Y > g(X) so X is rejected
Count_of_X_from_nth_Rectangle = 0; %To keep track of (d) k = n and a sample is drawn from the rectangular part of the nth region,
Count_of_X_from_Tail = 0; %To keep track of (e) k = n and the tail algorithm is run.
rng('shuffle'); %generate a random seed
Zig_nsamples = 1e6; %Initialize number of samples to 1000000
z = zeros(Zig_nsamples, 1); %Initialize an array to store Ziggurat samples
tic %Begin the timer to calculate the time taken to produce Ziggurat Samples
for i = 1:Zig_nsamples %The algorithm must run 1000000.
    while 1 %Infinite loop, breaks when a sample is accepted
```

```
        K = randi(n); %Generate random integers between 1 and n to choose the kth region
        if(K<n) %check if K is less than n
            X = xvals(1)+(xvals(K+1)-xvals(1)).*rand; %generate X~U(x1,xk+1)
            if(X<xvals(K)) %If true, accept X without any further checks
                Count_of_X_less_than_Xk = Count_of_X_less_than_Xk + 1; %Count the number of times the above
happens
                z(i)=X; %Add X to Ziggurat sample vector
                break; %end the infinite loop
            else %if X>=xvals(k)
                g_X = 1./(1+X).^3; %calculate the value of g(X)
                Y = yvals(K+1)+(yvals(K)-yvals(K+1)).*rand; %Generate Y~U(Yk+1,yk)
                if(Y<g_X) %if X>xk but Y<g_X
                    Count_of_X_greater_than_Xk_but_Y_less_than_gX =
Count_of_X_greater_than_Xk_but_Y_less_than_gX + 1; %count the above occurences
                    z(i)=X; %Add X to Ziggurat sample vector
                    break; %end the loop
                else %if Y>g(X)
                    Count_of_Y_greater_than_gX = Count_of_Y_greater_than_gX +1; %count the number of times this
happens and continue in the loop
                end
            end
        else
            Rand_prob_generator = rand(1,1); %generate random number between 0 and 1
            if(Rand_prob_generator <= P) %If true, Generate value from rectangle of the nth region
                Count_of_X_from_nth_Rectangle = Count_of_X_from_nth_Rectangle + 1; %count the above occurences
                X_nth_Region = xvals(n)*rand; %Generate X~U(x1,xn)
                z(i) = X_nth_Region; %%Add X to Ziggurat sample vector
                break; %End the loop
            else %if not true, generate the value from the tail region
                U = rand; %generate random number between (0,1) and obtain the sample as in Task 7
                if(n == 4)
                    X_nth_Region =
sqrt(6822280381494641./(1125899906842624.*((27432874449925277234215308492 8./274328744499252784678716781927)-U)))-1;
                elseif(n == 32)
                    X_nth_Region =
sqrt(2863200573390867./(35184372088832.*((25789418070820198212091263320064./2578941807082019550781
2898789009)-U)))-1;
                else
                    X_nth_Region =
sqrt(1622492499733937./(2199023255552.*((14614113234426366507245847445504./146141132344263676209013
27224961)-U)))-1;
                end
                Count_of_X_from_Tail = Count_of_X_from_Tail+1; %Count the above occurences
                z(i)= X_nth_Region; %Add X to Ziggurat sample vector
                break;
            end
        end
    end
end
toc %End the timer to calculate time taken for generating Ziggurat Samples


disp('Count_of_X_less_than_Xk = '), disp(Count_of_X_less_than_Xk);
disp('Count_of_X_greater_than_Xk_but_Y_less_than_gX = '),
disp(Count_of_X_greater_than_Xk_but_Y_less_than_gX);
```

```
disp('Count_of_Y_greater_than_gX = '), disp(Count_of_Y_greater_than_gX);
disp('Count_of_X_from_nth_Rectangle = '), disp(Count_of_X_from_nth_Rectangle);
disp('Count_of_X_from_Tail = '), disp(Count_of_X_from_Tail);


% Task 9 PDF Esimation of Ziggurat Samples same as Task 4
del_z = 0.1;
zmin = min(z);
zmax = max(z);
bincenters_z = zmin:del_z:zmax;
pdf_est_z = zeros(1,length(bincenters_z));
for i=1:length(bincenters_z)
    pdf_est_z(i) = nnz(z>bincenters_z(i)-del_z/2 & z<=bincenters_z(i)+del_z/2)/Zig_nsamples/del_z;
end
z_real = linspace(zmin,zmax,1e6);
pdf_real_z = c./((1+z_real).^3);
plot(bincenters_z, pdf_est_z, z_real, pdf_real_z,'LineWidth',2);
title('PDF estimate of Ziggurat Algorithm sampling VS Real PDF')
xlabel('z')
ylabel('PDF')

%Task 10
%Count the total number of samples being generated
Total_Count =
Count_of_X_less_than_Xk+Count_of_X_greater_than_Xk_but_Y_less_than_gX+Count_of_Y_greater_than_gX+C
ount_of_X_from_nth_Rectangle+Count_of_X_from_Tail;
%Probability that the chosen sample X <xk
Probability_of_X_less_than_Xk = Count_of_X_less_than_Xk/Total_Count;
%Probability that the chosen samples X>xk but Y<g(X)
Probability_of_X_greater_than_Xk_but_Y_less_than_gX =
Count_of_X_greater_than_Xk_but_Y_less_than_gX/Total_Count;
%Probability that Y > g(X)
Probability_of_Y_greater_than_gX = Count_of_Y_greater_than_gX/Total_Count;
%Probability of choosing a sample from rectangle of the nth region
Probability_of_X_from_nth_Rectangle = Count_of_X_from_nth_Rectangle/Total_Count;
%Probability of choosing a sample from the tail region
Probability_of_X_from_Tail = Count_of_X_from_Tail/Total_Count;
%Verifying is the calculations are correct
Total_P =
Probability_of_X_less_than_Xk+Probability_of_X_greater_than_Xk_but_Y_less_than_gX+Probability_of_Y_great
er_than_gX+Probability_of_X_from_nth_Rectangle+Probability_of_X_from_Tail;

disp('Total_Count'),disp(Total_Count);
disp('Probability_of_X_less_than_Xk = '), disp(Probability_of_X_less_than_Xk);
disp('Probability_of_X_greater_than_Xk_but_Y_less_than_gX = '),
disp(Probability_of_X_greater_than_Xk_but_Y_less_than_gX);
disp('Probability_of_Y_greater_than_gX = '), disp(Probability_of_Y_greater_than_gX);
disp('Probability_of_X_from_nth_Rectangle = '), disp(Probability_of_X_from_nth_Rectangle);
disp('Probability_of_X_from_Tail = '), disp(Probability_of_X_from_Tail);
disp('Total_Probability'), disp(Total_P);
```

Steps followed for Implementation:

Task 1: Normalization constant for Burr Distribution

A probability distribution function is said to be normalized if the sum of all its possible results is equal to one. We know that the burr distribution is a continuous PDF and it exists for non-negative random variables. Therefore, the normalization constant can be found as follows:

$$f(x) = C.\int_0^\infty 1/(1+x)^3\, dx$$

and $\quad C.\int_0^\infty 1/(1+x)^3\, dx = A = 1$

By substitution:

Let $\qquad (1+x) = u$

$\Rightarrow \qquad dx = du$

$\Rightarrow\ C.\int_0^\infty 1/(u)^3\, du = 1$

$\Rightarrow\ C.\int_0^\infty (u)^{-3}\, du = 1$

$\Rightarrow\ C.-\frac{u^{-2}}{2}\Big|_0^\infty = 1$

$\Rightarrow\ C.\left[-\frac{u^{-2}}{2} - \left[-\frac{u^{-2}}{2}\right]\right]\Big|_0^\infty = 1$

By un-substituting:

$\Rightarrow\ C.\frac{1}{2}\left[-\frac{1}{(1+x)^2} - \left[-\frac{1}{(1+x)^2}\right]\right]\Big|_0^\infty = 1$

$\Rightarrow\ C.\frac{1}{2}\left[-\frac{1}{(1+\infty)^2} - \left[-\frac{1}{(1+0)^2}\right]\right] = 1$

$\Rightarrow\ C.\frac{1}{2}[-0+1] = 1$

Solving for C, we get:

$\boldsymbol{C = 2}$

In MATLAB, the integral function is used to solve numeric functions. Integral(f, 0, inf) means f is being integrated from 0 to infinity.

Code Snippet:

Editor - C:\Users\vvvenka1\Documents\MATLAB\RST_Project_1.m

RST_Project_1.m | testing.m

```
%Task 1 Find the constant c so that your PDF is normalized correctly
f_x = @(x)1./(1+x).^3;
c = 1./integral(f_x, 0 ,Inf)
```

Output:

```
Command Window
>> RST_Project_1

c =

    2.0000
```

Task 2: CDF for Burr Distribution

Cumulative Distribution Function is defined as $F_X(x) = P(X \leq x)$. Therefore, the CDF of a function can be calculated by integrating between the intervals $(-\infty, x)$. And the Burr function exists only for non-negative random variables. Therefore:

$$F_X(x) = \int_{-\infty}^{x} C.1/(1+x)^3 \ dx \qquad and\ C = 2$$

$$F_X(x) = \int_{-\infty}^{0} 2/(1+x)^3 \ dx \ + \int_{0}^{x} 2/(1+x)^3 \ dx$$

$$F_X(x) = 0 \ + \int_{0}^{x} 2/(1+x)^3 \ dx$$

By substitution method:

Let $(1+x) = \ u$

$\Rightarrow \qquad dx = du$

$\Rightarrow \quad F_X(x) = 2\int_0^x 1/(u)^3 \ du$

$\Rightarrow \quad F_X(x) = 2\int_0^x u^{-3} \ du$

$\Rightarrow \quad F_X(x) = 2[-\frac{u^{-2}}{2}]|_0^x$

$\Rightarrow \quad F_X(x) = 2\left[-\frac{u^{-2}}{2} - \left[-\frac{u^{-2}}{2}\right]\right]|_0^x$

By un-substituting:

$\Rightarrow \quad F_X(x) = \left[-\frac{1}{(1+x)^2} - \left[-\frac{1}{(1+x)^2}\right]\right]|_0^x$

$\Rightarrow \quad F_X(x) = \left[-\frac{1}{(1+x)^2} - \left[-\frac{1}{(1+0)^2}\right]\right]$

$\Rightarrow \quad F_X(x) = \left[-\frac{1}{(1+x)^2} + 1\right]$

$\Rightarrow \quad F_X(x) = \left[1 - \frac{1}{(1+x)^2}\right]$

Therefore the CDF of Burr distribution is given as follows:

$$\Rightarrow \quad F_X(x) = \left[1 - \frac{1}{(1+x)^2}\right]$$

In MATLAB, the int function is used to integrate symbolic functions. Its function is like that of the integral function. As the integration of burr distribution function is closed form, we can use the function int.

Code Snippet:

Editor - C:\Users\vvenka1\Documents\MATLAB\RST_Project_1.m

RST_Project_1.m | testing.m

```
%Task 2 Write a function that computes the CDF of your distribution.
syms x
f_1 =@(x) c./(1+x).^3;
F_X = int(f_1,0,x)
```

Output:

Command Window

```
>> RST_Project_1

F_X =

1 - 1/(x + 1)^2
```

Task 3: Baseline Sampling Algorithm

A Baseline sampling algorithm acts as an output predictor for the Ziggurat Algorithm. To implement a Baseline Sampling Algorithm, we must find the inverse of CDF.

The inverse of CDF can be found as follows:

Let $y = \left[1 - \frac{1}{(1+x)^2}\right]$

$\Rightarrow 1 - y = \frac{1}{(1+x)^2}$

$\Rightarrow (1+x)^2 = \frac{1}{(1-y)}$

$\Rightarrow (1+x) = \pm\sqrt{\frac{1}{1-y}}$

$\Rightarrow x = \pm\sqrt{\frac{1}{1-y}} - 1$

$\Rightarrow F_X^{-1}(x) = F_X^{-1}\left(\pm\sqrt{\frac{1}{1-y}} - 1\right)$

In MATLAB, nsamples represents the number of baseline samples that we want. Here, 1000 random values between [0,1] will be created and stored in the vector u[1000x1].
y is the inverse function of the CDF $F_X(x)$. We consider only the positive root value as the function does not exist for -ve values of x. It stores 1000 base sample values.

Code Snippet:

```
Editor - C:\Users\vvenka1\Documents\MATLAB\RST_Project_1.m
RST_Project_1.m    testing.m

%Task 3 Implement a baseline sampling algorithm via the transformation S =
tic
rng(0);
nsamples = 1000;
u = rand(nsamples,1);
y = (1./sqrt(1-u))-1;
toc
```

Output:
u[i=1:1000] ={i ∈ (0,1)}
y[j=1:1000] ={j ∈ $(1./sqrt(1-u))-1, u \in (0,1)$}

The Tic and Toc is a function in MATLAB used to calculate the time elapsed between the tic and the toc. This is to check the time required to generate baseline samples.

An example output of tic and toc:

```
Command Window

Elapsed time is 0.000975 seconds.
fx >>
```

Task 4: Estimate PDF of base samples

The general rule to calculate PDF of an inverse function is as follows:

$$f_Y(y) = f_X(g^{-1}(y)) \left|\frac{d}{dx} g^{-1}(y)\right|$$
$$\left|\frac{d}{dx} g^{-1}(y)\right| = \frac{1}{\left(2\sqrt{-\frac{1}{y-1}}\right) \times (y-1)^2}$$

$$f_Y(y) = f_X\left(\sqrt{\frac{1}{1-y}} - 1\right)\frac{1}{\left(2\sqrt{-\frac{1}{y-1}}\right)\times(y-1)^2} + f_X\left(-\sqrt{\frac{1}{1-y}} - 1\right)\frac{1}{\left(2\sqrt{-\frac{1}{y-1}}\right)\times(y-1)^2}$$

$$f_Y(y) = \frac{1}{(1+\left(\sqrt{\frac{1}{1-y}} - 1\right))^3} \times \frac{1}{\left(2\sqrt{-\frac{1}{y-1}}\right)\times(y-1)^2} + \frac{1}{(1-\left(\sqrt{\frac{1}{1-y}} - 1\right))^3} \times \frac{1}{\left(2\sqrt{-\frac{1}{y-1}}\right)\times(y-1)^2}$$

In MATLAB, the PDF estimation method is a little different. Here, del stands for delta, a small value. From the vector y created in the task 3, a minimum value and a maximum value is found. A new array called as bincenters is initialized, which contains all the values from ymin and ymax varying with an interval of del, ie 0.1.
pdf_est vector is created to store all the estimated pdf values. The size of pdf_est is same as the bincenters.
Next, the number of non-zero values in y that lie between bincenters-del/2 and bincenters+del/2 is calculated.
When the count is divided by the number of samples, the probability estimate is obtained. Dividing this probability estimate by del gives us the PDF.
Next, the obtained PDF must be compared with the actual PDF. To do this, a vector is initialed and assigned 1000 values between ymin and ymax. The 1000 values must be linearly spaced between ymin and ymax, therefore, linspace function is used. And PDF is calculated for each of the x_real values. Plot a graph to compare the PDFs.

Code snippet:

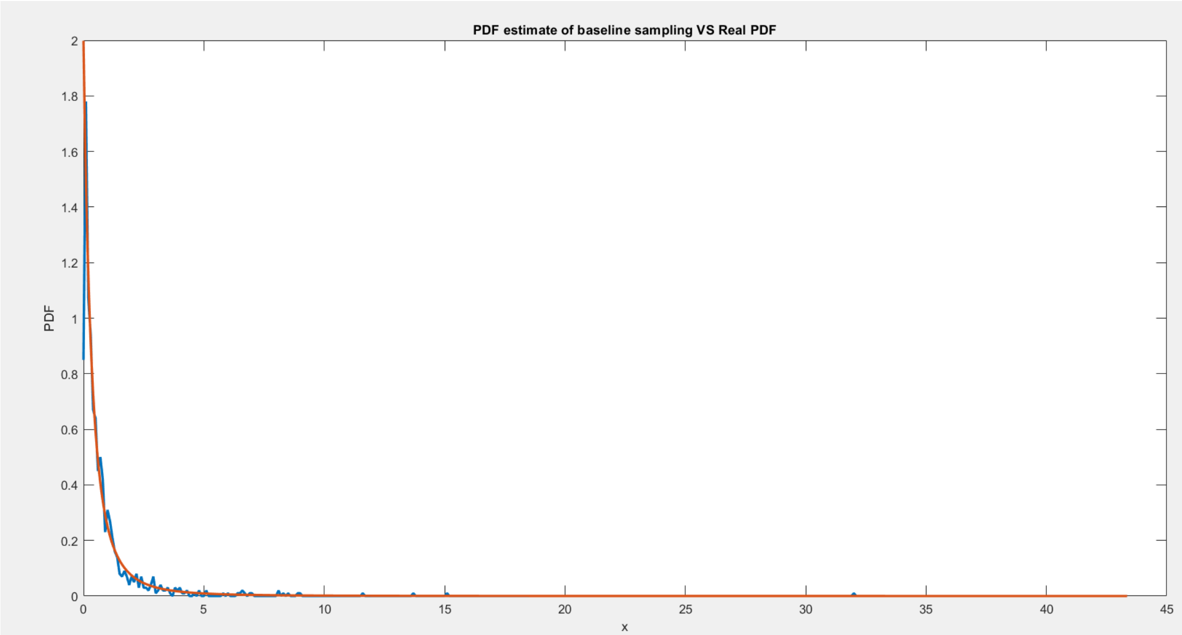
Editor - C:\Users\vvvenka1\Documents\MATLAB\RST_Project_1.m

RST_Project_1.m | testing.m | testforgraphs.m | +

```

%Task 4, PDF estimation
del = 0.1;
ymin = min(y);
ymax = max(y);
bincenters = ymin:del:ymax;
pdf_est = zeros(1,length(bincenters));
for i=1:length(bincenters)
    pdf_est(i) = nnz(y>bincenters(i)-del/2 & y<=bincenters(i)+del/2)/nsamples/del;
end
x_real = linspace(ymin,ymax,1000);
pdf_real = c./((1+x_real).^3);
plot(bincenters, pdf_est,x_real,pdf_real,'LineWidth',2);
title('PDF estimate of baseline sampling VS Real PDF')
xlabel('x')
ylabel('PDF')
```

Output:



Task 5: Set up for Ziggurat Algorithm

To find the values of $x_1, x_2, x_3 \dots x_n$ *and* $y_1, y_2, y_3 \dots y_n$ the following steps have to be followed:

1. Guess a value for Area A and x_{k+1} then calculate the value of Area_rect from the equation given below:
$$(x_{k+1} - x_k)(y_k - y_{k+1}) = Area\_Rect$$
2. If the Area_Rect is greater than the A we guessed, then we must increase the guessed value of x_{k+1}. Else, x_{k+1} value must be reduced.
3. Step 2 must be repeated until Area_Rect == A and n times for all values x_1 *to* x_n.
4. With the $x_n$value obtained; we must calculate the area of nth region Area_nth_Region. If Area_nth_Region is greater than A, then A must be increased. Else A must be decreased, and Step 1, Step 2 and Step 3 must be repeated until Area_nth_Region and A are equal.

In MATLAB, it has been implemented using the bisection algorithm. The bisection algorithm is applied twice, one as inner bisection algorithm for finding the value of x_{k+1}. And the outer bisection algorithm to find the value of A. The values of Amin= 0, Amax=20 are guessed value and xmin=0, xmax=45 are also guessed values from the graph. The variable tol is short for tolerance. It is used to stop the loop when the difference between xmax, xmin and Amax, Amin is less than 10^{-12}. Lesser the tol value, more accurate the algorithm. The nth region is the sum of rectangular area and the tapering region 'tail' from x_n to ∞. The area of the tail is calculated by integrating g(x) from x_n to ∞. The area of nth region is calculated as follows:

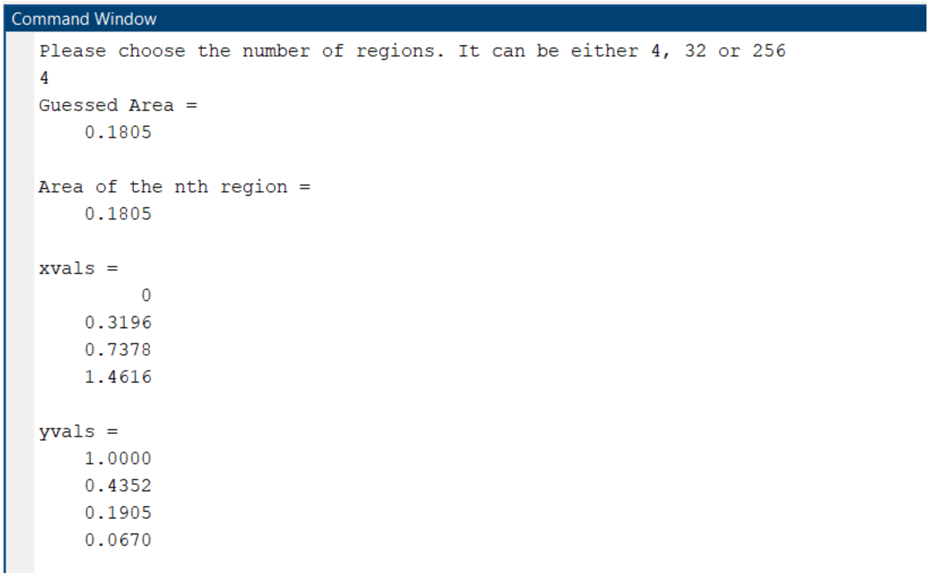
Area_nth_Region = $(x_n - x_1)y_n + \int_{x_n}^{\infty} 1/(1+x)^3 \, dx$

Code Snippet:

```
%task 5
tail = 0;
Amin = 0;
Amax = 10;
tol = 1e-13;
prompt = 'Please choose the number of regions. It can be either 4, 32 or 256\n';
n = input(prompt);
xvals = zeros(n,1);
yvals = zeros(n,1);
xvals(1) = 0;
Area_nth_Region = 0;
while Amax-Amin>tol
    A = (Amax+Amin)./2;
    for k = 1:(n-1)
        xmax = 45;
        xmin = 0;
        while xmax - xmin > tol
            xvals(k+1) = (xmin+xmax)./2;
            yvals(k) = 1./(1+xvals(k)).^3;
            yvals(k+1) = 1./(1+xvals(k+1)).^3;
            Area_rect = (xvals(k+1)-xvals(1)).*(yvals(k)-yvals(k+1));
            if Area_rect > A
                xmax = xvals(k+1);
            else
                xmin = xvals(k+1);
            end

        end
    end
    tail = integral(f_x,xvals(n),Inf);
    Area_nth_Region = ((xvals(n)-xvals(1)).*(yvals(n)))+tail;
    if Area_nth_Region > A
        Amin = A;
    else
        Amax = A;
    end
end
disp("Guessed Area = "),disp(A);
disp("Area of the nth region = "), disp(Area_nth_Region);
disp("xvals = "), disp(xvals);
disp("yvals = "), disp(yvals);
```

Output:



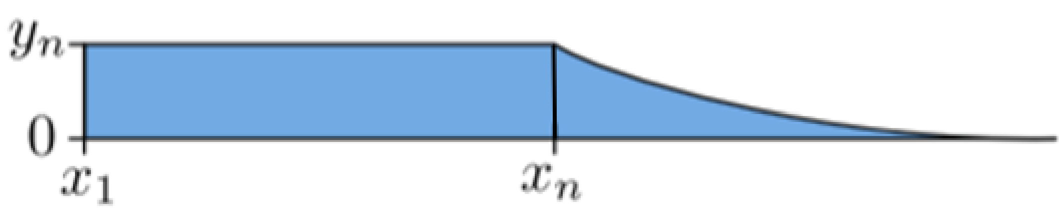
```
Command Window
Please choose the number of regions. It can be either 4, 32 or 256
4
Guessed Area =
    0.1805

Area of the nth region =
    0.1805

xvals =
         0
    0.3196
    0.7378
    1.4616

yvals =
    1.0000
    0.4352
    0.1905
    0.0670
```

Task 6: To find probability that the sample X lies in the rectangle of the nth region.



The probability that X lies in the rectangle = $\frac{Area\ of\ the\ rectangle}{Area\ of\ the\ rectangle + Area\ of\ the\ tail)}$

In MATLAB, the area of the rectangle of the nth region is found by subtracting tail from Area_nth_Region. P is different for each value of n.

Output:

P(n = 4) = 0.5429;
P(n = 32) = 0.6401;
P(n = 256) = 0.6583;

Code Snippet:

Editor - C:\Users\vvenka1\Documents\MATLAB\RST_Project_1.m

RST_Project_1.m | testing.m | testforgraphs.m

```

%Task 6
nth_Rect_Area = Area_nth_Region - tail;
P = nth_Rect_Area/Area_nth_Region;

```

Task 7: To generate samples from the tail distribution.

This is implemented just like the base sampling algorithm is implemented, with a slight change. The lower bound to find the CDF is changed from 0 to x_n. The value of the normalization constant also changes with the change in the limits. Therefore, for each value of n, new c must be calculated. Though only one sample value is required for the ziggurat algorithm, 10000 samples have been generated for testing purpose.
In MATLAB, f_t is just a reinitialization of g(x) to indicate that it is for tail sampling.
c_1 refers to the new normalization constant.

c_1(n=4) = 12.1188
c_1(n=32) = 162.7541
c_1(n=256) = 1475.6

f_t_normalized is reinitialization of g(x) along with the normalized constant c_1.
F_T is the CDF of f_t obtained by integrating f_t from x_n to x(by the definition, CDF = P(X$\leq x$)

To sample the tail part, we must first find the inverse of F_T. Inverse of F_T is assigned to the vector tail_sampling. (I obtained the inverse of F_T manually as finverse and solve were not working for me)

Output(for n = 4):

Command Window

```
>> tail_sampling(1:10)

ans =

    3.4933
    4.7607
    2.0493
    6.9465
   12.9050
    3.5143
    2.2022
    3.6559
    3.8181
    2.8478
```

Code Snippet:

```
%Task 7
f_t = @(x) 1./(1+x).^3;
c_1 = 1./integral(f_t, xvals(n), Inf);

f_t_normalized = @(x) c_1./(1+x).^3;
F_T = int(f_t_normalized, xvals(n), x);

rng('shuffle');
xsamples = 10000;
A = rand(xsamples, 1);
if(n == 4)
    tail_sampling = sqrt(6822280381494641./(1125899906842624.*((274328744499252772342153084928./274328744499252784678716781927)-A)))-1;
elseif(n == 32)
    tail_sampling = sqrt(2863200573390867./(35184372088832.*((25789418070820198212091263320064./25789418070820195507812898789009)-A)))-1;
else
    tail_sampling = sqrt(1622492499733937./(2199023255552.*((14614113234426366507245847445504./14614113234426367620901327224961)-A)))-1;
end
```

Task 8: Implementation of Ziggurat Algorithm.

Now that all the required pre-calculations have been done, we can implement Ziggurat Algorithm. It is like rejection sampling, where if a chosen sample satisfies the condition Y<g(x), then it is accepted. Else the sample is rejected. The ziggurat algorithm has other conditions along with Y<g(x) which makes the algorithm faster.

First, a point (X, Y) must be chosen uniformly from the rectangle region. The point will then be accepted or rejected based on the following conditions:

1. If the chosen point X<x_K of the kth region, then there is no need to generate Y or even check if Y<g(x). This is because, if X<x_K then Y will be inside g(x) no matter what if K<n.
2. If the above condition is not satisfied, then we must generate Y and check if Y<g(x). If the condition is true, then the chosen sample is accepted given K<n.
3. If K>n then we need to generate sample from the nth region. This is a combination of task 6 and 7.

   Here, we must choose a value for X such that $X{\sim}U(x_1, x_n)$ with a probability P. And we must choose a value for X from the tail region with a probability 1-P. The sampling of the tail region has been shown in Task 7. But for the implementation, we only need one sample.

In MATLAB, the process of sampling must be done only once which is handled by the for loop. But the process of choosing a point and rejecting must be done until a point is accepted. Since we do not know how many times a sample might be rejected, we use the infinite while(1) loop until a point is accepted. Once the point is accepted, we break out of the loop.

A million ziggurat samples are being generated and stored in vector z. Hence the for loop runs a million times. The Kth region is also being selected randomly from 1 to n.
For values of K<n, step 1 and 2 are followed.
For values of K=n, step 3 is followed.

Code Snippet:

RST_Project_1.m | testing.m | testforgraphs.m

```
%Task 8 Complete the implementation of the Ziggurat algorithm for n = 4, n = 32, and n = 256.
rng('shuffle');
Zig_nsamples = 1e6;
z = zeros(Zig_nsamples, 1);
tic
for i = 1:Zig_nsamples
    while 1
        K = randi(n);
        if(K<n)
            X = xvals(1)+(xvals(K+1)-xvals(1)).*rand;
            if(X<xvals(K))
                z(i)=X;
                break;
            else
                g_X = 1./(1+X).^3;
                Y = yvals(K+1)+(yvals(K)-yvals(K+1)).*rand;
                if(Y<g_X)
                   z(i)=X;
                   break;
                end
            end
        else
            Rand_prob_generator = rand(1,1);
            if(Rand_prob_generator < P)
                X_nth_Region = xvals(n)*rand;
                z(i) = X_nth_Region;
                break;
            else
                U = rand;
                if(n == 4)
                    X_nth_Region = sqrt(6822280381494641./(1125899906842624.*((27432874449925277234215308492 8./274328744499252784678716781927)-U)))-1;
                elseif(n == 32)
                    X_nth_Region = sqrt(2863200573390867./(35184372088832.*((257894180708201982120912633200 64./257894180708201955078128987890 09)-U)))-1;
                else
                    X_nth_Region = sqrt(1622492499733937./(2199023255552.*((146141132344263665072458474455 04./146141132344263676209013272249 61)-U)))-1;
                end
                z(i)= X_nth_Region;
                break;
            end
        end
    end
end
toc
```

Output:

Command Window

```
Length of z =
      1000000

>> z(1:10)

ans =

     1.5644
     0.2612
     1.9040
     0.0000
     0.2855
     0.0961
     0.0295
     0.3583
     0.1146
     0.3201
```

Task 9: Estimate PDF of Ziggurat Samples

The PDF estimation process is exactly same as that in Task 4.

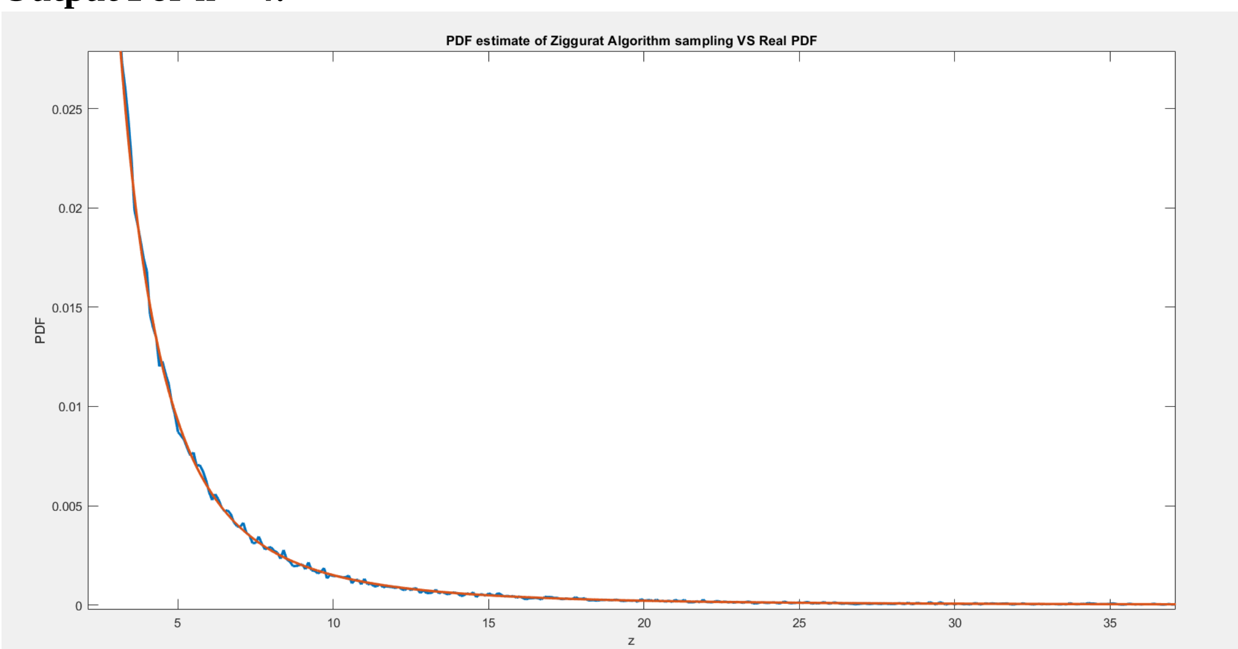
Code Snippet:

Editor - C:\Users\vvenka1\Documents\MATLAB\RST_Project_1.m

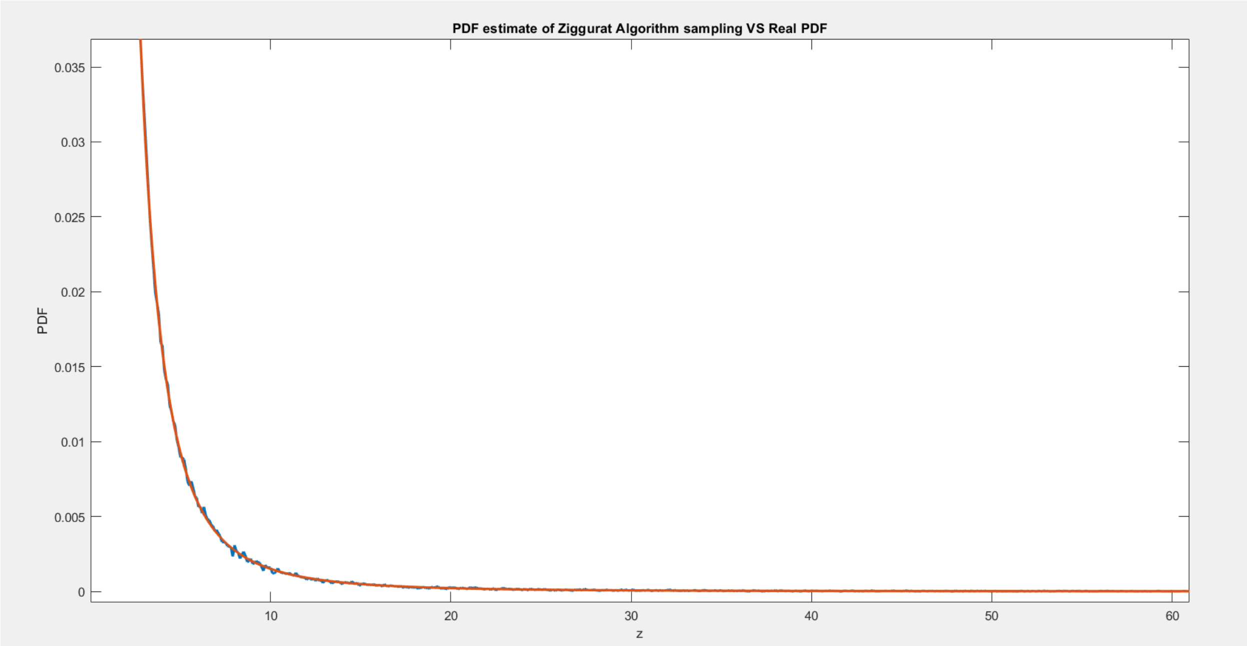
RST_Project_1.m | testing.m | testforgraphs.m

```
% Task 9
del_z = 0.1;
zmin = min(z);
zmax = max(z);
bincenters_z = zmin:del_z:zmax;
pdf_est_z = zeros(1,length(bincenters_z));
for i=1:length(bincenters_z)
    pdf_est_z(i) = nnz(z>bincenters_z(i)-del_z/2 & z<=bincenters_z(i)+del_z/2)/Zig_nsamples/del_z;
end
z_real = linspace(zmin,zmax,1e6);
pdf_real_z = c./((1+z_real).^3);
plot(bincenters_z, pdf_est_z, z_real, pdf_real_z,'LineWidth',2);
```

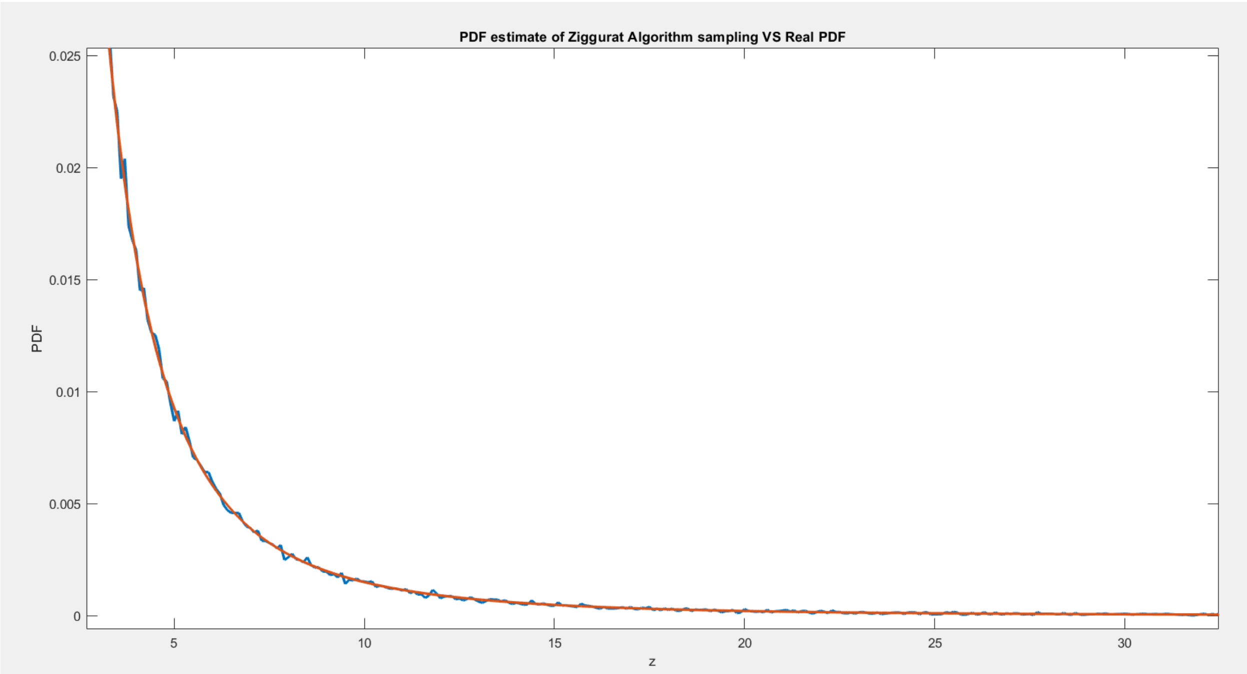
Output For n = 4:



Output for n = 32:



Output for n = 256



Task 10: Keeping track of the data

Code Snippet:

```
%Task 10
%Count the total number of samples being generated
Total_Count = Count_of_X_less_than_Xk+Count_of_X_greater_than_Xk_but_Y_less_than_gX+Count_of_Y_greater_than_gX+Count_of_X_from_nth_Rectangle+Count_of_X_from_Tail;
%Probability that the chosen sample X <xk
Probability_of_X_less_than_Xk = Count_of_X_less_than_Xk/Total_Count;
%Probability that the chosen samples X>xk but Y<g(X)
Probability_of_X_greater_than_Xk_but_Y_less_than_gX = Count_of_X_greater_than_Xk_but_Y_less_than_gX/Total_Count;
%Probability that Y > g(X)
Probability_of_Y_greater_than_gX = Count_of_Y_greater_than_gX/Total_Count;
%Probability of choosing a sample from rectangle of the nth region
Probability_of_X_from_nth_Rectangle = Count_of_X_from_nth_Rectangle/Total_Count;
%Probability of choosing a sample from the tail region
Probability_of_X_from_Tail = Count_of_X_from_Tail/Total_Count;
%Verifying is the calculations are correct
Total_P = Probability_of_X_less_than_Xk+Probability_of_X_greater_than_Xk_but_Y_less_than_gX+Probability_of_Y_greater_than_gX+Probability_of_X_from_nth_Rectangle+Probability_of

disp('Total_Count'),disp(Total_Count);
disp('Probability_of_X_less_than_Xk = '), disp(Probability_of_X_less_than_Xk);
disp('Probability_of_X_greater_than_Xk_but_Y_less_than_gX = '), disp(Probability_of_X_greater_than_Xk_but_Y_less_than_gX);
disp('Probability_of_Y_greater_than_gX = '), disp(Probability_of_Y_greater_than_gX);
disp('Probability_of_X_from_nth_Rectangle = '), disp(Probability_of_X_from_nth_Rectangle);
disp('Probability_of_X_from_Tail = '), disp(Probability_of_X_from_Tail);
disp('Total_Probability'), disp(Total_P);
%
```

Question:

1. How long it takes to run. Compare the time per sample with your baseline algorithm. Make sure you run your code on the same computer, so that the comparison in meaningful.

Observations:

For n=4:

Time taken to generate 1000 Baseline samples = 0.000891 second
Time taken to generate 1000000 Ziggurat samples = 0.459159 second
Time taken to generate 1 Baseline sample = 0.000891/1000
= 0.000000891
= 891 nanosecond
Time taken to generate 1 Ziggurat sample = 0.459159/1000000
= 0.000000459159
= 459.159 nanosecond

For n=32:

Time taken to generate 1000 Baseline samples = 0.000370 second
Time taken to generate 1000000 Ziggurat samples = 0.167678 second
Time taken to generate 1 Baseline sample = 0.000370 /1000
= 0.000000370
= 370 nanosecond
Time taken to generate 1 Ziggurat sample = 0.459159/1000000
= 0.000000167678
= 167.678 nanosecond

For n=256:

Time taken to generate 1000 Baseline samples = 0.000583 second
Time taken to generate 1000000 Ziggurat samples = 0.111883 second
Time taken to generate 1 Baseline sample = 0.000891/1000
= 0.000000583
= 583 nanosecond
Time taken to generate 1 Ziggurat sample = 0.459159/1000000
= 0.000000111883
= 111.883 nanosecond

These observations prove that Ziggurat Algorithm is much faster than the Sampling method.

Question:

2. How often each of the following possible outcomes occurs in the rejection loop:
(a) X is accepted because X < xk,
(b) X > xk but X is accepted because Y < g(X),
(c) Y > g(X) so X is rejected,
(d) k = n and a sample is drawn from the rectangular part of the nth region,
(e) k = n and the tail algorithm is run.

Observation:

For n = 4:

(a) Count of X is accepted because X < xk = 338233
(b) Count of X > xk but X is accepted because Y < g(X) = 300955
(c) Count of Y > g(X) so X is rejected = 443569
(d) Count of k = n and a sample is drawn from the rectangular part of the nth region = 195682
(e) Count of k = n and the tail algorithm is run = 165130

The obtained value of P(n=4) = 0.5429
From the counts p can be calculated as follows:
p = (Count of X from nth rectangle) / (count of X from nth rectangle + count of X from tail)
p = 195682/360812 = 0.5423378 ~ 0.5429 = P

For lesser values of n, the X being rejected because Y>g(X) is higher. The probability that X lies on either side of xk is nearly same. Can be verified from the probability calculations below.
The number of rejected samples = 445247

```
Command Window
Count_of_X_less_than_Xk =
      338233

Count_of_X_greater_than_Xk_but_Y_less_than_gX =
      300955

Count_of_Y_greater_than_gX =
      443569

Count_of_X_from_nth_Rectangle =
      195682

Count_of_X_from_Tail =
      165130
```

```
Command Window

Total_Count
     1445247

Probability_of_X_less_than_Xk =
    0.2340

Probability_of_X_greater_than_Xk_but_Y_less_than_gX =
    0.2077

Probability_of_Y_greater_than_gX =
    0.3081

Probability_of_X_from_nth_Rectangle =
    0.1356

Probability_of_X_from_Tail =
    0.1147

Total_Probability
    1.0000
```

For n = 32:

(a) Count of X is accepted because X < xk = 882956
(b) Count of X > xk but X is accepted because Y < g(X) = 82873
(c) Count of Y > g(X) so X is rejected = 92788
(d) Count of k = n and a sample is drawn from the rectangular part of the nth region = 21960
(e) Count of k = n and the tail algorithm is run = 12211

The obtained value of P(n=32) = 0.6401
From the counts p can be calculated as follows:
p = (Count of X from nth rectangle) / (count of X from nth rectangle + count of X from tail)
p = 21960/34171 = 0.642650 ~ 0.5429 = P

For higher values of n, the X being rejected because Y>g(X) is less. The probability that X lies on left side of xk is much higher than X on the right side of xk. Can be verified from the calculations below.

The number of rejected samples = 92424

```
Command Window

Elapsed time is 0.169773 seconds.
Count_of_X_less_than_Xk =
      882956

Count_of_X_greater_than_Xk_but_Y_less_than_gX =
       82873

Count_of_Y_greater_than_gX =
       92788

Count_of_X_from_nth_Rectangle =
       21960

Count_of_X_from_Tail =
       12211
```

```
Command Window

Total_Count
     1092424

Probability_of_X_less_than_Xk =
    0.8083

Probability_of_X_greater_than_Xk_but_Y_less_than_gX =
    0.0759

Probability_of_Y_greater_than_gX =
    0.0846

Probability_of_X_from_nth_Rectangle =
    0.0200

Probability_of_X_from_Tail =
    0.0112

Total_Probability
     1
```

For n = 256:
(a) Count of X is accepted because X < xk = 981431
(b) Count of X > xk but X is accepted because Y < g(X) = 14662
(c) Count of Y > g(X) so X is rejected = 15330

(d) Count of k = n and a sample is drawn from the rectangular part of the nth region = 2549
(e) Count of k = n and the tail algorithm is run = 1358

The obtained value of P(n=32) = 0.6583
From the counts p can be calculated as follows:
p = (Count of X from nth rectangle) / (count of X from nth rectangle + count of X from tail)
p = 2549/3907 = 0.6524 ~ 0.6583 = P

For much higher values of n, the X being rejected because Y>g(X) is much less. The probability that X lies on left side of xk is very much higher than X on the right side of xk. Can be verified from the probability calculation given below.

The number of rejected samples = 15431

Command Window

```
Elapsed time is 0.126626 seconds.
Count_of_X_less_than_Xk =
      981431

Count_of_X_greater_than_Xk_but_Y_less_than_gX =
       14662

Count_of_Y_greater_than_gX =
       15330

Count_of_X_from_nth_Rectangle =
        2549

Count_of_X_from_Tail =
        1358
```

Command Window

```
Total_Count
     1015431

Probability_of_X_less_than_Xk =
    0.9665

Probability_of_X_greater_than_Xk_but_Y_less_than_gX =
    0.0144

Probability_of_Y_greater_than_gX =
    0.0152

Probability_of_X_from_nth_Rectangle =
    0.0026

Probability_of_X_from_Tail =
    0.0013

Total_Probability
     1
```

Task 11: Improving the Ziggurat Algorithm

1) We know that a sample from the Nth region is picked with a probability of 1/n. Since the probability of this occurring is so rare, instead of using if else condition statement inside the algorithm, a general inverse function can be found which uses the average of all three normalization constants. Thus, avoiding repetitive checks.

2) Avoid extra calculations inside the algorithm such as keeping track of data. A few nanoseconds will make a difference.

3) Generating a random variable is also time taking. If it can be avoided inside the algorithm, generation of Ziggurat samples might be faster. But I cannot think of any alternate way to do this.

4)The higher the number of rectangular regions, greater the acceptance rate of the samples, hence more efficient. Hence, it is better to work with higher values on n.

References:

[1]Burr, I. W. (1942). "Cumulative frequency functions". Annals of Mathematical Statistics. 13 (2): 215–232. doi:10.1214/aoms/1177731607. JSTOR 2235756.

[2] Maddala, G. S. (1996) [1983]. Limited-Dependent and Qualitative Variables in Econometrics. Cambridge University Press. ISBN 0-521-33825-5.

[3] Tadikamalla, Pandu R. (1980), "A Look at the Burr and Related Distributions", International Statistical Review, 48 (3): 337–344, doi:10.2307/1402945, JSTOR 1402945

[4]The Ziggurat Random Normal Generator Blogs of MathWorks, posted by Cleve Moler, May 18, 2015.