Monte Carlo Problem Set 1

Güneykan Özgül

February 2017

1 Metropolis et al Algorithm

Following plots are the results of running Metropolis et al. algorithm with parameter $\delta = 1.2$ to generate random numbers obeying the following density function,

$$f(x) = A \cdot e^{-(x^2+y^2)}$$

.

There are three plots generated to get an idea about the distribution of the data. For better comparison I plotted the distribution of x, y components and also radius separately.

Note that histogram values are normalized for better comparison.

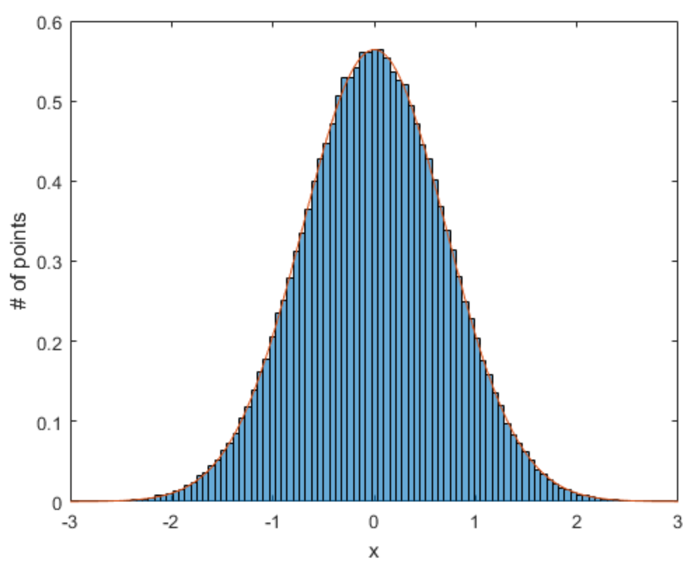

Figure 1: Distribution of X Density

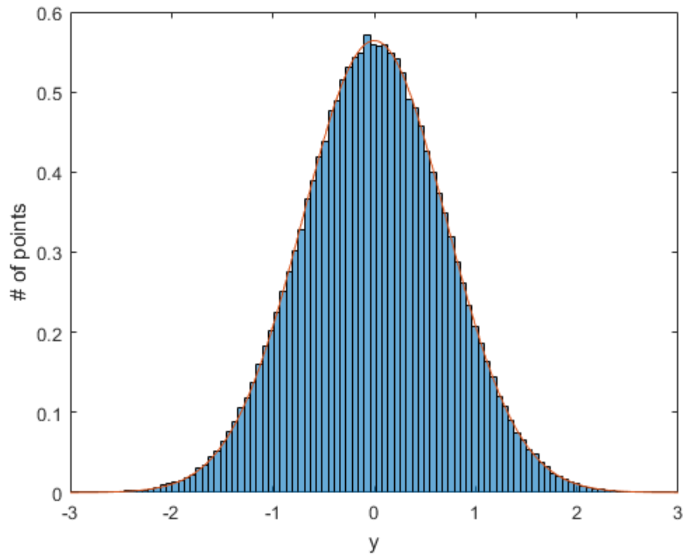

Figure 2: Distribution of Y Density

Finally w(r)=$2\pi r P(r)$ is plotted.

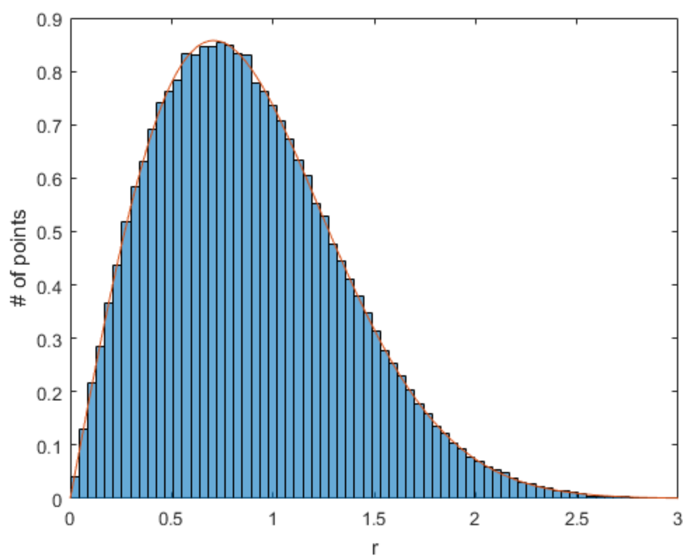

Figure 3: Distribution of Radius Density

2 Monte Carlo Integration

Using simple Monte Carlo method, the following integration is performed.

$$\int_0^1 f(x)dx$$

where $f(x) = 3e^{-10\cdot(x-1/10)^2} + e^{-50\cdot(x-3/5)^2}$

When $w(x) = 1$ is used as weighting function the following table is obtained.

N	I	σ_I^2	$\frac{N\cdot\sigma_I^2}{\sigma_f^2}$	$\frac{N\cdot\sigma_I^2}{\sigma_f^2} - 1$
10	1.2564	0.12314	1.155	0.15496
20	1.7659	0.050816	0.95323	-0.046768
50	1.5788	0.025483	1.1951	0.19507
100	1.3583	0.010546	0.9891	-0.0109
200	1.4417	0.0053744	1.0082	0.0081598
500	1.372	0.0020388	0.95612	-0.043878
1000	1.433	0.0010433	0.97858	-0.021422
2000	1.3529	0.00052964	0.99353	-0.0064652
5000	1.4024	0.00021366	1.002	0.0019835

Table 1: Smoothing by w(x)=1

When $w(x) = -2x + 2$ is used as weighting function the Table 2 is obtained. This function is a good choice to smooth out the function since it behaves similar to the $f(x)$ in the domain of integration. Besides it is normalized since,

$$\int_0^1 (-2x+2)dx = 1$$

To find x as a function of y we should compute,

$$y = \int_0^x w(x')dx' = \int_0^x (-2x'+2)dx'$$

Then,

$$y = -x^2 + 2x \implies x = 1 \pm \sqrt{1-y}$$

For our interested domain lies between [0-1], we choose the function

$$x = 1 - \sqrt{1-y}$$

as our inverse function.

N	I	σ_I^2	$\frac{N\cdot\sigma_I^2}{\sigma_f^2}$	$\frac{N\cdot\sigma_I^2}{\sigma_f^2}-1$
10	1.38	0.015087	0.1415	-0.8585
20	1.4057	0.010536	0.19765	-0.80235
50	1.316	0.0036232	0.16992	-0.83008
100	1.4026	0.00099574	0.093394	-0.90661
200	1.3951	0.00052656	0.098775	-0.90122
500	1.3657	0.00024361	0.11424	-0.88576
1000	1.386	0.00011966	0.11223	-0.88777
2000	1.3794	6.2538e-05	0.11731	-0.88269
5000	1.3859	2.4483e-05	0.11482	-0.88518

Table 2: Smoothing by w(x)=-2x+2

The following table is obtained upon running the program repetitively for the same number of input points.

N	$\sigma^2_{\sigma_I^2}$	$N\cdot\frac{\sigma^2_{\sigma_I^2}}{\sigma_f^2}$
10	0.00088325	0.0082843
20	0.00012235	0.0022951
50	7.0429e-06	0.00033029
100	7.6029e-07	7.131e-05
200	1.0435e-07	1.9575e-05
500	6.3153e-09	2.9617e-06
1000	6.737e-10	6.3188e-07
2000	1.1426e-10	2.1434e-07
5000	7.1499e-12	3.353e-08

Table 3: Results

3 Variance of Variance

$$\sigma_I = \frac{\sigma_f}{N} = \frac{1}{N}\cdot(< f^2 > - < f >^2)$$

If we wish to find the variance of this value for different executions of the algorithm, we obtain the following relation

$$\sigma_{\sigma_I} = \frac{1}{N}(\cdot < (< f^2 > - < f >^2)^2 > - << f^2 > - < f >^2 >^2)$$

Expanding some of the terms we get,

$$\sigma_{\sigma_I} = \frac{1}{N}(<< f^2 >^2 - 2 < f^2 >< f >^2 + < f >^4 > -(<< f^2 >> - << f >^2 >)^2)$$

and furthermore,
$\sigma_{\sigma_I} = \frac{1}{N}(<< f^2 >^2 - 2 < f^2 >< f >^2 + < f >^4 > -(<< f^2 >>^2 - 2 << f^2 >><< f >^2 > + << f >^2 >^2))$

Eliminating some of the terms, we obtain

$\sigma_{\sigma_I} = \frac{1}{N}(<< f^2 >^2 > + << f >^4 > - << f^2 >>^2 - << f >^2 >)$

where inner mean represents the mean of the function for N points and outer mean represents mean of the values for different executions of the algorithm on same number of input points. Quantitatively,

$$<< g >> = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} g(x_{ij})$$

where g is the relative function for this calculation.

4 Appendix

4.1 Metropolis Code

```
function z = P(r )
if r>3 z=0;
else
z=(1/pi)*exp(-r^2);
end

end


%Metropolis et al Algorithm.
%Implementation:Guneykan Ozgul

%Set initial Points.
initialX=-3;
initialY=-3;
current(1)=initialX;
current(2)=initialY;

%Set displacement.
delta=1.2;

%Set number of random points.
n=1000000;

%Store each component and radius seperately.
radius=zeros(n,1);
xPoints=zeros(n,1);
yPoints=zeros(n,1);

%Store number of accepted points as a performance measure.
accepted=0;

for i= 1: n

    %Choose test point.
    test(1)=delta*(2*rand-1)+current(1);
    test(2)=delta*(2*rand-1)+current(2);
    %Calculate ratio.
    ratio=P(norm(test))/P(norm(current));

    %Decide if the point should be accepted or not.
    if( ratio > rand)
```

```
            current=test;
            accepted=accepted+1;
        end

        %Set the next point.
        xPoints(i)=current(1);
        yPoints(i)=current(2);
        radius(i)=norm(current);

end


%Plot the figures.
figure
histogram(xPoints,100,'Normalization','pdf');
xlabel('x');
ylabel('# of points')
hold on ;
fplot(@(x) sqrt(pi)*P(x),[-3 3])
figure
histogram(yPoints,100,'Normalization','pdf'); %// plot histogram
xlabel('y');
ylabel('# of points')
hold on;
fplot(@(y) sqrt(pi)*P(y),[-3 3])
figure
histogram(radius,100,'Normalization','pdf'); %// plot histogram
xlabel('r');
ylabel('# of points')
hold on;
fplot(@(r) 2*pi*r*P(r),[0 3])
```

4.2 Monte-Carlo Integration Code

```
function y = f(x)
%   Detailed explanation goes here
y=3*exp(-10*(x-1/10)^2)+exp(-50*(x-3/5)^2);
end

function y = w(x)
y=-2*x+2;
end

function y = inverse1(x)
y=x;
end
```

```
function y = inverse2(x)
y=1-sqrt(1-x);
end

function [y, var] = Integ(f,w,inv,N)


SUM=0;
F2SUM=0;
for i=1:N
  value=rand;
  inverse=inv(value);
  fvalue=f(inverse)/w(inverse);
  SUM=SUM+fvalue;
  F2SUM=F2SUM+fvalue^2;
end

SUM=SUM/N;
F2SUM=F2SUM/N;
y=SUM;
var= F2SUM-SUM^2;
var=var/N;
end

N=[10;20;50;100;200;500;1000;2000;5000];

I=zeros(length(N),1);
var_I=zeros(length(N),1);
var_F=2.97513-(1.38165)^2;
variance_error=zeros(length(N),1);


I2=zeros(length(N),1);
var_I2=zeros(length(N),1);
var_F2=2.97513-(1.38165)^2;
variance_error2=zeros(length(N),1);

for i=1:length(N)

   [I(i), var_I(i)]=Integ(@(x) f(x),@(x) 1,@(x) inverse1(x),N(i));
    variance_error(i)=N(i)*var_I(i)/var_F;

end


for i=1:length(N)
```

```
  [I2(i), var_I2(i)]=Integ(@(x) f(x),@(x) w(x),@(x) inverse2(x),N(i));
   variance_error2(i)=N(i)*var_I2(i)/var_F;

end

%varianceOfvariance=
columnNames = {'N','I1','Var_I1','Var_IF1','VarError1','I2','Var_I2','Var_IF2','
T = table(N,I,var_I,variance_error,variance_error-1,I2,var_I2,variance_error2,va




cvariance=zeros(length(N),1);
I=zeros(50,1);
vars=zeros(50,1);

for i=1:length(N)
    temp=zeros(1000,1);
  for j=1:1000
     [I(j),temp(j)]=Integ(@(x) f(x),@(x) 1,@(x) inverse1(x),N(i));
  end
  cvariance(i)=var(temp);
end


columnNames = {'N','VarOfVars','VarOfVarsError'};
T = table(N,cvariance,N.*cvariance./var_F,'VariableNames',columnNames)

figure
fplot(@(x) f(x),[0 1]);
```