Omer Solakli 2471677864

EE 511: Simulation of Stochastic Processes

Spring 2018

Project#4

[Pi-Estimation]

i]

```
n=20; %number of samples
k=50; %number of estimates
z=20; %number of variance estimates
v=zeros(1,z);
for t=1:1:z
p_bar=zeros(1,k);
for i=1:1:k
u1=zeros(1,n);
u2=zeros(1,n);
x=zeros(1,n);

 for j=1:1:n
    u1(j)=rand;
    u2(j)=rand;
 if ((u1(j))^2+(u2(j))^2)<=1
     x(j)=1;
 end

end
 p_bar(i)=4*(sum(x)/n);
end
histogram(p_bar);

v(t)= var(p_bar);
n=n+20;
end
figure
plot(v,'-o');
ylim([0 0.2])
```

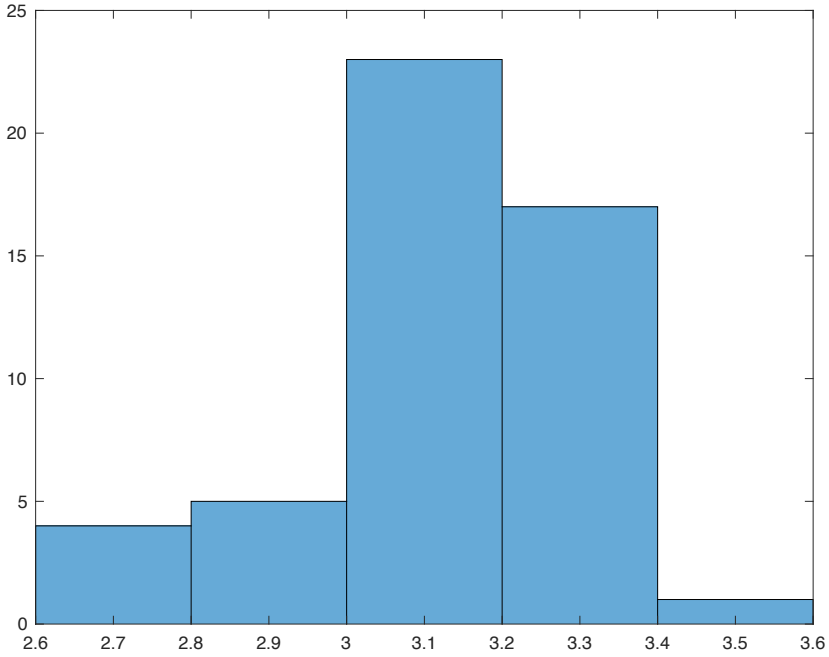

Figure 1: Pi estimation histogram for k=50

ii]

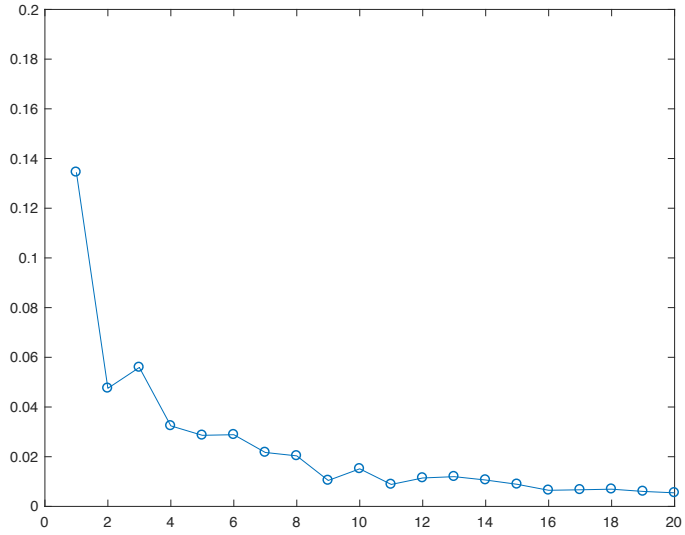

Figure 2: Sample Variance plot

Discussion:

We can see that the estimator does a relatively good job by estimating close to 3.14 within the majority of the trials. Above histogram is generated for n=100, k=50. Then the code is modified for the plot for the variances of p_bar. To estimate pi, we multiply by 4 since we are working with quarter of a circle. Out of 50 trials around 40 of the estimation fell in between [3,3.4].

Above code is generated for the plot of the sample variances with changing n. I am starting with n=20 and increasing n by 20 at each iteration. I am passing the variances at each time interval to v(t) array and in the end plotting them against t.

It can be clearly seen that there is a big drop for variance when we increase number of trials. This is due to Law of Large Numbers and Central Limit Theorem. We can see that when we increase n from n=20, to n=40 variance drops drastically. Same thing can be said when it is increased from n=60, to n=80. But after n=200, the variance won't change much. Thus, our performance does not improve as much as our computation increase. Thus, I would suggest to use n=[150,200] for this experiment.

[Monte Carlo Integration and Variance Reduction Strategies]

Simple Monte Carlo Integration :

```
%--------------a---------------%
n=1000; %random samples
x=zeros(1,n);
f=zeros(1,n);

var=0;
for j=1:1:n
    x(j)=unifrnd(0.8,3.0);
    f(j)=(1+sinh(2*x(j))*log(x(j)))^-1;

end
integral=(3.0-0.8)*sum(f)/n;
for j=1:1:n
    var=var+(f(j)-integral)^2;

end
var=var/(n-1);

%--------------a---------------%
```

integral =0.6308
var =0.2729

```
%--------------b---------------%

n=1000; %random samples
x=zeros(1,n);
y=zeros(1,n);
f=zeros(1,n);

var=0;
for j=1:1:n
    x(j)=unifrnd(-pi,pi);
    y(j)=unifrnd(-pi,pi);
    f(j)=exp(-x(j)^4-y(j)^4);

end
integral=(2*pi)*(2*pi)*sum(f)/n;
for j=1:1:n
    var=var+(f(j)-integral)^2;

end
var=var/(n-1);
```

```
%--------------b---------------%
```

integral =3.4613
var =11.4501

Discussion 1:

We can see that for simple monte carlo estimation, integrals for both parts are close to the real values. According to wolfram integral for part a is around 0.6 and for part b it is 3.3. Our estimator does fairly good with integral estimation. The is problem with variance. When we run the above code multiple times. Variance for part a is relatively small but for part b it is very high and this suggest us to use variance reduction techniques.

Monte Carlo Estimation with Stratification:

```
%--------------a---------------%


n=1000; %random samples
lambda=0.25;
Na=lambda*n;
Nb=n-Na;
x1=zeros(1,Na);
f1=zeros(1,Na);
x2=zeros(1,Nb);
f2=zeros(1,Nb);
var1=0;
var2=0;
for j=1:1:Na
    x1(j)=unifrnd(0.8,0.8+(3.0-0.8)*lambda);
    f1(j)=(1+sinh(2*x1(j))*log(x1(j)))^-1;

end
integral1=(((3.0-0.8)*lambda))*sum(f1)/Na;
for j=1:1:Na
    var1=var1+(f1(j)-integral1)^2;

end
var1=var1/(Na-1);
for j=1:1:Nb
    x2(j)=unifrnd((0.8+(3.0-0.8)*lambda),3);
    f2(j)=(1+sinh(2*x2(j))*log(x2(j)))^-1;

end
integral2=((3.0-0.8)*(1-lambda))*sum(f2)/Nb;

integral=integral1+integral2;

for j=1:1:Nb
    var2=var2+(f2(j)-integral2)^2;

end
var2=var2/(Nb-1);
var=(lambda/n)*var1+((1-lambda)/n)*var2;

%--------------a---------------%
```

integral = 0.6268

var =1.0141e-04

```
%--------------b---------------%

n=1000; %random samples
lambda=0.4;
Na=lambda*n;
Nb=n-Na;
x1=zeros(1,Na);
y1=zeros(1,Na);
f1=zeros(1,Na);
x2=zeros(1,Nb);
y2=zeros(1,Nb);
f2=zeros(1,Nb);
var1=0;
var2=0;
for j=1:1:Na
    x1(j)=unifrnd(-pi,-pi+(pi-(-pi))*lambda);
    y1(j)=unifrnd(-pi,pi);
    f1(j)=exp(-x1(j)^4-y1(j)^4);
end
integral1=(pi-(-pi))*(pi-(-pi))*lambda*sum(f1)/Na;
for j=1:1:Na
    var1=var1+(f1(j)-integral1)^2;

end
var1=var1/(Na-1);
for j=1:1:Nb
    x2(j)=unifrnd((-pi+(pi-(-pi))*lambda),pi);
    y1(j)=unifrnd(-pi,pi);
    f2(j)=exp(-x2(j)^4-y2(j)^4);

end
integral2=((pi-(-pi))^2)*(1-lambda)*sum(f2)/Nb;

integral=integral1+integral2;

for j=1:1:Nb
    var2=var2+(f2(j)-integral2)^2;

end
var2=var2/(Nb-1);
var=(lambda/n)*var1+((1-lambda)/n)*var2;

%--------------b---------------%
```

integral =10.3828

var =0.0555

Discussion 2:

Above code divides the integral intervals to two subintervals and generates Na samples from interval1 and generates Nb samples from interval2 which are proportional to parameter lambda.

We can see that the variances are reduces drastically. Stratified sampling partitions the integral boundaries into the subsets that union of the subsets generates the original

boundaries. Then it calculates the integrals and variances within those boundaries. Then it sums the integrals. The idea behind stratified sampling is variance of a subinterval should be lower than variance of the entire interval. Hence, the overall variance is reduced.

For part a, after some tuning I decided to choose lambda=0.25 as it gives very low variance with **var =1.0141e-04.** For part b, as much as I tuned lambda even though variances are reduced the integral calculations seemed to be a little bit off. I used stratification method only for variable x, rather than using for both x and y that might be the reason why I am getting different integral values than the real answer. But the variance is reduced to **var =0.0555.** Which is a great improvement.

Monte Carlo Integration with Importance Sampling:

Idea behind importance sampling is to sample from the function close to our function to be integrated. Using this technique, instead of sampling from uniform and evaluating **f** at those points, we sample from the points which **f** is more.

g should have a shape close to that of f while being simple to sample. Thus, I select g as g=1/(1+ln(x)) since it behaves similar to our function. Maximum of this function occurs at x=0.8 with 1.27 and minimum occurs at x=3.0 with 0.47. My idea is to sample between different y values and taking the inverse of 1/(1+ln(x)) to get the respective x values which are more concentrated on 0.8 side of the function. Since its value is greater there it is reasonable to have more sample around there.

```
%--------------a---------------%



n=1000; %random samples
y=zeros(1,n);
x=zeros(1,n);
f=zeros(1,n);
var=0;

for j=1:1:n
    y(j)=unifrnd(0.47650535804,1.28723910536); %generating samples of y,
max of y is 1.287 on our interval
    x(j)=exp((1-y(j))/y(j)); %inverse of y method to generate the observed
    f(j)=(1+sinh(2*x(j))*log(x(j)))^-1;

end
integral=sum(f)/n;
for j=1:1:n
    var=var+(f(j)-integral)^2;

end
var=var/(n-1);
%--------------a---------------%
```

integral =0.7668
var =0.431

Testing against f(x,y):

```
n=10000; %random samples
x=zeros(1,n);
y=zeros(1,n);
f=zeros(1,n);

var=0;
for j=1:1:n
    x(j)=unifrnd(-5,5);
    y(j)=unifrnd(-5,5);
    f(j)=20+x(j)^2+y(j)^2-10*(cos(2*pi*x(j))+cos(2*pi*y(j)));
end
integral=10*5*sum(f)/n;
for j=1:1:n
    var=var+(f(j)-integral)^2;

end
var=var/(n-1);
```

integral =

1.8375e+03