# Assignment 10

Chao Cheng

April 25, 2019

## 1 Question 1

In Homeworks 5, 7, 8, and 9, you were asked to use Monte Carlo integration to approximate

$$\int_0^1 e^{x^2} dx$$

Use Importance sampling in a simulation to approximate this integral using at least two different new probability measures. Compare the variance of your estimator with the variance of your previous results.

### 1.1 Importance Sampling Using $g(x) = x^2/3$

#### 1.1.1 Code

```
#include<iostream>
#include<cmath>
#include <tuple>
using namespace std;
/* g(x)  = 3 * x^2 */

tuple<double,double> MCIntegrelImpSamp(){
    double u;           // u(0,1)
    double x, y, x_;    // realization of exp(u^2)
    double est, estA;   // est.
    int n = 1000;       // 1000 points
    est = 0.0;
    estA = 0.0;
    for(int i = 0;i < n;i++){
         u = ((double)rand()/(RAND_MAX));
         x_ = pow(u,1.0/3.0);//inversion
         x = exp(u*u);
         y = exp(x_*x_)/(x_*x_); // h(x_)*f(x_)/g(x_)
         est = est + (x - est)/(i+1);
         estA = estA + (y - estA)/(i+1);
    }
    estA = estA/3.0;
    return make_tuple(est,estA);
}

tuple<double,double,double,double> getVars(int n){
    double x, y, xA;   // realizations
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of antithetic

    auto results = MCIntegrelImpSamp();
    x = double(get<0>(results));
    y = double(get<1>(results));
    xA = y;
    mean = x;
    meanA = xA;
    svs = 0;
    svsA = 0;
    for(int i = 1; i < n; i++)
    {
        auto results = MCIntegrelImpSamp();
        x = double(get<0>(results));
        y = double(get<1>(results));
        xA = y;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);
        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
    }
    return make_tuple(mean, meanA, svs, svsA);
```

```
}

int main(){
    int N=1000;
    srand(1);       //set up the same random seed for comparison
    auto Var = getVars(N);
    cout<<"For N = "<<N<<endl;
    cout<<"Mean Value of crude MC integrel is "<<get<0>(Var)<<endl;
    cout<<"Mean Value of Importance Sampling MC Integrel is "<<get<1>(Var)<<endl;
    cout<<"Variance of crude MC Integrel is "<<get<2>(Var)<<endl;
    cout<<"Variance of Importance Sampling MC Integrel is "<<get<3>(Var)<<endl;
    cout<<endl;
}
```

#### 1.1.2 Results

```
For N = 1000
Mean Value of crude MC integrel is 1.46249
Mean Value of Importance Sampling MC Integrel is 1.46367
Variance of crude MC Integrel is 0.000230932
Variance of Importance Sampling MC Integrel is 0.0594474
```

### 1.2 Importance Sampling Using $g(x) = e^x/(e-1)$

#### 1.2.1 Code

```
#include<iostream>
#include<cmath>
#include <tuple>
#define coef1 (exp(1.0)-1.0)
using namespace std;
/*   g(x) = exp(x)/(e - 1)  */

tuple<double,double> MCIntegrelImpSamp(){
    double u;           // u(0,1)
    double x, y, xx;    // realization of exp(u^2)
    double est, estA;   // est.
    int n = 1000;       // 1000 points
    est = 0.0;
    estA = 0.0;
    for(int i = 0;i < n;i++){
         u = ((double)rand()/(RAND_MAX));
         xx = log(1 + coef1 * u);// inversion
         x = exp(u * u);
         y = exp(xx*xx-xx); // h(x_)*f(x_)/g(x_)
         est = est + (x - est)/(i+1);
         estA = estA + (y - estA)/(i+1);
    }
    estA = estA*coef1;
    return make_tuple(est,estA);
}

tuple<double,double,double,double> getVars(int n){
    double x, y, xA;   // realizations
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of antithetic

    auto results = MCIntegrelImpSamp();
    x = double(get<0>(results));
    y = double(get<1>(results));
    xA = y;
    mean = x;
    meanA = xA;
    svs = 0;
    svsA = 0;
    for(int i = 1; i < n; i++)
    {
        auto results = MCIntegrelImpSamp();
        x = double(get<0>(results));
        y = double(get<1>(results));
        xA = y;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);
        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
    }
```

```
    return make_tuple(mean, meanA, svs, svsA);
}

int main(){
    int N=1000;
    srand(1);       //set up the same random seed for comparison
    auto Var = getVars(N);
    cout<<"For N = "<<N<<endl;
    cout<<"Mean Value of crude MC integrel is "<<get<0>(Var)<<endl;
    cout<<"Mean Value of Importance Sampling MC Integrel is "<<get<1>(Var)<<endl;
    cout<<"Variance of crude MC Integrel is "<<get<2>(Var)<<endl;
    cout<<"Variance of Importance Sampling MC Integrel is "<<get<3>(Var)<<endl;
    cout<<endl;
}
```

#### 1.2.2 Results

```
For N = 1000
Mean Value of crude MC integrel is 1.46249
Mean Value of Importance Sampling MC Integrel is 1.46255
Variance of crude MC Integrel is 0.000230932
Variance of Importance Sampling MC Integrel is 1.31716e-05
```

### 1.3 Other Methods

#### 1.3.1 Stratified sampling

Code:

```
#include<iostream>
#include<cmath>
#include <tuple>
#define pis 8.0*atan(1)
using namespace std;
/* The control variate is u, therefore x + c(u-0.5)*/

tuple<double,double> MCIntegrel(){
    double u, u_;    // u(0,1)
    double x, y;    //realization of exp(u^2)
    double est,estA;   //est.
    int n = 1000;//1000 points

    for(int i = 0;i < n;i++){
         u = ((double)rand()/(RAND_MAX));
         u_ = ((double) (u + i) / n);
         x = exp(u*u);
         y = exp(u_*u_);
         est = est + (x - est)/(i+1);
         estA = estA + (y - estA)/(i+1);
    }
    return make_tuple(est,estA);
}
tuple<double,double,double,double> getVars(int n){
    double u;          // u(0,1),
    double x, y, xA;   // realizations
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of antithetic

    auto results = MCIntegrel();
    x = double(get<0>(results));
    y = double(get<1>(results));
    xA = y;
    mean = x;
    meanA = xA;
    svs = 0;
    svsA = 0;
    for(int i = 1; i < n; i++)
    {
        auto results = MCIntegrel();
        x = double(get<0>(results));
        y = double(get<1>(results));
        xA = y;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);
        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
    }
```

```
    return make_tuple(mean, meanA, svs,svsA);
}
int main(){
    int N=1000;
    srand(1);       //set up the same random seed for comparison
    auto Var = getVars(N);
    cout<<"For N = "<<N<<endl;
    cout<<"Mean Value of crude MC integrel is "<<get<0>(Var)<<endl;
    cout<<"Mean Value of Stratified Sampling MC Integrel is "<<get<1>(Var)<<endl;
    cout<<"Variance of crude MC Integrel is "<<get<2>(Var)<<endl;
    cout<<"Variance of Stratified Sampling MC Integrel is "<<get<3>(Var)<<endl;
    cout<<endl;
}
```

Results:

```
For N = 1000
Mean Value of crude MC integrel is 1.46261
Mean Value of Stratified Sampling MC Integrel is 1.46265
Variance of crude MC Integrel is 0.00024643
Variance of Stratified Sampling MC Integrel is 4.32473e-10
```

### 1.3.2

```
#include<iostream>
#include<cmath>
#include <tuple>
#define pis 8.0*atan(1)
using namespace std;
/* The control variate is u, therefore x + c(u-0.5)*/
tuple<double,double,double,double> MCIntegrelVar(int n){
    double u,us[100];           // u(0,1),
    double x, y;       //realization of exp(u^2)
    double est, estA;   //est.(integral), Control est.
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of control
    double co_xy,yvar,ybar;//covariance, y mean, variance of y
    double xbar,copt;

    u = ((double)rand()/(RAND_MAX));
    us[0] = u;
    x = exp(u*u);
    y = u;
    ybar = y;
    xbar = x;
    yvar = 0;
    co_xy = 0;
    for(int i = 1; i < 100; i++)
    {
         u = ((double)rand()/(RAND_MAX));
         us[i] = u;
         x = exp(u*u);
         y = u;
         yvar = yvar*(i - 1)/i  + (ybar - y)*(ybar - y)/(i + 1);
         ybar = ybar + (y - ybar)/(i + 1);
         xbar = xbar + (x - xbar)/(i + 1);
         co_xy = co_xy*(i - 1)/i + (x-xbar)*(y-ybar)/i; //recursive covariance
    }
    copt = -co_xy/yvar; // c_opt

    est = exp(us[0]*us[0]);
    estA = est + copt*(us[0]-0.5);
    mean = est;
    meanA = estA;
    svs = 0;
    svsA = 0;
    for(int i = 1; i < 100; i++)
    {
         x = exp(us[i]*us[i]);
         y = x + copt*(us[i]-0.5);
         est = est + (x - est)/(i+1);
         estA = estA + (y - estA)/(i+1);
         svs = svs *(i - 1)/i  + (mean - est)*(mean - est)/(i + 1);
         svsA = svsA *(i - 1)/i  + (meanA - estA)*(meanA - estA)/(i + 1);
         mean = mean + (est - mean)/(i+1);
```

```
        meanA = meanA + (estA - meanA)/(i+1);

    }
    for(int i = 100;i < n;i++){
        u = ((double)rand()/(RAND_MAX));
        x = exp(u*u);
        y = x + copt*(u-0.5);
        est = est + (x - est)/(i+1);
        estA = estA + (y-estA)/(i+1);
        svs = svs *(i - 1)/i  + (mean - est)*(mean - est)/(i + 1);
        svsA = svsA *(i - 1)/i  + (meanA - estA)*(meanA - estA)/(i + 1);
        mean = mean + (est - mean)/(i+1);
        meanA = meanA + (estA - meanA)/(i+1);
    }
    return make_tuple(mean, meanA, svs,svsA);
}
int main(){
    int N = 1000;
    srand(1);       //set up the same random seed for comparison
    auto Var = MCIntegrelVar(N);
    cout<<"For N = "<<N<<endl;
    cout<<"Mean Value of crude MC integrel is "<<get<0>(Var)<<endl;
    cout<<"Mean Value of control MC Integrel is "<<get<1>(Var)<<endl;
    cout<<"Variance of crude MC Integrel is "<<get<2>(Var)<<endl;
    cout<<"Variance of control MC Integrel is "<<get<3>(Var)<<endl;
    cout<<endl;
}
```

```
For N = 1000
Mean Value of crude MC integrel is 1.48865
Mean Value of control MC Integrel is 1.45789
Variance of crude MC Integrel is 0.000991057
Variance of control MC Integrel is 7.71649e-05
```

## 2 Question 2

In Homework 9, you were asked to use simulation to estimate $P\{X + Y > 4\}$ where $X$ and $Y$ are independent exponentials with parameters 1 and 1/2 respectively. Use importance sampling in a simulation study of this probability. Compare your efficiency and accuracy of this simulation with your previous results. Also, use importance sampling on your conditional estimator and compare with your previous results.

### 2.1 Codes

```
#include<iostream>
#include<cmath>
#include <tuple>
#define coef1 25.0/12.0
using namespace std;

tuple<double,double,double,double,double> twoNumbers(){
    double u1,u2; //u1,u2~u(0,1)
    double e1,e2,s; //exp(1),exp(0.5)
    double i,j,ii,jj;//i crude, j is the control variate, ii antithetic, jj importance sa
    double jjj;//jjj importance sampling on conditioning

    //importance sampling: g(x) is tilted distribution with t = 0.2
    u1 = ((double)rand()/(RAND_MAX));
    u2 = ((double)rand()/(RAND_MAX));
    e1 = -log(u1);
    e2 = -2*log(u2);
    i = 0;
    j = 0;
    ii = 0;
    jj = 0;
 // crude
    if ((e1+e2>4)) {
        i = 1.0;
    }
//conditioning
    if (e2>=4.0) {
        j = 1.0;
    }else{
        j = exp(e2 - 4.0);
    }
 // antithetic, later averaged
    if (-log(1-u1)-2*log(1-u2)>4) {
        ii = 1.0;
    }
// importance sampling, using tilted distribution with t=0.2
    e1 =  -5.0/4.0 * log(u1);
    e2 = -10.0/3.0 * log(u2);

    s = e1 + e2;
    if(s >=4){
        jj = exp(-0.2*s)*25/12.0;
    }
//importance sampling on conditional estimator
    if (e2 >= 4.0) {
        jjj = exp(-0.2*s)*25/12.0;
    }else{
        jjj = exp(e2 - 4.0) * exp(-0.2*s)*25/12.0;
    }

    return make_tuple(i,j,ii,jj,jjj);
}
tuple<double,double,double,double,double,double,double,double,double,double> getVars(int
    double x, y, xA, xB, xC,xD;   // realizations
    double y1,y2,y3;
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of conditioning
    double meanB,svsB; //mean, var of antithetic
    double meanC,svsC; //mean, var of importance sampling
    double meanD,svsD; //mean, var of importance sampling on conditioning

    auto results = twoNumbers();
    x = get<0>(results);
    y = get<1>(results);
    y1 = get<2>(results);
    y2 = get<3>(results);
```

```
    y3 = get<4>(results);
    xA = y;
    xB = (x + y1)/2.0;
    xC = y2;
    mean = x;
    meanA = xA;
    meanB = xB;
    meanC = xC;
    meanD = xD;
    svs = 0;
    svsA = 0;
    svsB = 0;
    svsC = 0;
    for(int i = 1; i < n; i++)
    {
        auto results = twoNumbers();
        x =  get<0>(results);
        y =  get<1>(results);
        y1 = get<2>(results);
        y2 = get<3>(results);
        y3 = get<3>(results);
        xA = y;
        xB = (x + y1)/2.0;
        xC = y2;
        xD = y3;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);
        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
        svsB = svsB *(i - 1)/i  + (meanB - xB)*(meanB - xB)/(i+1);
        meanB = meanB + (xB-meanB)/(i+1);
        svsC = svsC *(i - 1)/i  + (meanC - xC)*(meanC - xC)/(i+1);
        meanC = meanC + (xC-meanC)/(i+1);
        svsD = svsD *(i - 1)/i  + (meanD - xD)*(meanD - xD)/(i+1);
        meanD = meanD + (xD-meanD)/(i+1);
    }
    return make_tuple(mean, meanA, meanB, meanC, meanD, svs, svsA, svsB,svsC,svsD);
}

int main(){
    int N = 1000;
    srand(5);       //set up the same random seed for comparison
    auto Var = getVars(N);
    cout<<"For␣N␣=␣"<<N<<endl;
    cout<<"Value␣of␣the␣crude␣MC␣is␣"<<get<0>(Var)<<endl;
    cout<<"Value␣of␣Conditioning␣MC␣is␣"<<get<1>(Var)<<endl;
    cout<<"Value␣of␣Anthithetic␣MC␣is␣" <<get<2>(Var)<<endl;
    cout<<"Value␣of␣Importance␣sampling␣MC␣is␣" <<get<3>(Var)<<endl;
    cout<<"Value␣of␣Importance␣sampling␣on␣Conditioning␣MC␣is␣" <<get<4>(Var)<<endl;
    cout<<endl;
    cout<<"Variance␣of␣the␣crude␣MC␣is␣"<<get<5>(Var)<<endl;
    cout<<"Variance␣of␣Conditioning␣MC␣is␣"<<get<6>(Var)<<endl;
    cout<<"Variance␣of␣Anthithetic␣MC␣is␣" <<get<7>(Var)<<endl;
    cout<<"Variance␣of␣Importance␣sampling␣MC␣is␣" <<get<8>(Var)<<endl;
    cout<<"Variance␣of␣Importance␣sampling␣on␣conditioning␣MC␣is␣" <<get<9>(Var)<<endl;
    cout<<endl;
}
```

### 2.2 Results

```
For N = 1000
Value of the crude MC is 0.263
Value of Conditioning MC is 0.248965
Value of Anthithetic MC is 0.256
Value of Importance sampling MC is 0.254361
Value of Importance sampling on Conditioning MC is 0.254236

Variance of the crude MC is 0.194025
Variance of Conditioning MC is 0.112328
Variance of Anthithetic MC is 0.06603
Variance of Importance sampling MC is 0.103503
Variance of Importance sampling on conditioning MC is 0.103551
```

# 3 Question 3

Repeat Exercise 2 for $P\{X + Y > 9\}$.

## 3.1 Codes

```
#include<iostream>
#include<cmath>
#include <tuple>
#define coef1 25.0/12.0
using namespace std;

tuple<double,double,double,double,double> twoNumbers(){
    double u1,u2; //u1,u2~u(0,1)
    double e1,e2,s; //exp(1),exp(0.5)
    double i,j,ii,jj;//i crude, j is the control variate, ii antithetic, jj importance sa
    double jjj;//jjj importance sampling on conditioning

    //importance sampling: g(x) is tilted distribution with t = 0.2
    u1 = ((double)rand()/(RAND_MAX));
    u2 = ((double)rand()/(RAND_MAX));
    e1 = -log(u1);
    e2 = -2*log(u2);
    i = 0;
    j = 0;
    ii = 0;
    jj = 0;
 // crude
    if ((e1+e2>9)) {
        i = 1.0;
    }
//conditioning
    if (e2>=9.0) {
        j = 1.0;
    }else{
        j = exp(e2 - 9.0);
    }
 // antithetic, later averaged
    if (-log(1-u1)-2*log(1-u2)>9) {
        ii = 1.0;
    }
// importance sampling, using tilted distribution with t=0.2
    e1 =  -5.0/4.0 * log(u1);
    e2 = -10.0/3.0 * log(u2);

    s = e1 + e2;
    if(s >=9){
        jj = exp(-0.2*s)*25/12.0;
    }
//importance sampling on conditional estimator
    if (e2 >= 9.0) {
        jjj = exp(-0.2*s)*25/12.0;
    }else{
        jjj = exp(e2 - 9.0) * exp(-0.2*s)*25/12.0;
    }

    return make_tuple(i,j,ii,jj,jjj);
}
tuple<double,double,double,double,double,double,double,double,double,double> getVars(int
    double x, y, xA, xB, xC,xD;   // realizations
    double y1,y2,y3;
    double mean, svs;  //mean, sample variance of estimator
    double meanA,svsA; //mean, var of conditioning
    double meanB,svsB; //mean, var of antithetic
    double meanC,svsC; //mean, var of importance sampling
    double meanD,svsD; //mean, var of importance sampling on conditioning

    auto results = twoNumbers();
    x = get<0>(results);
    y = get<1>(results);
    y1 = get<2>(results);
    y2 = get<3>(results);
    y3 = get<4>(results);
    xA = y;
    xB = (x + y1)/2.0;
```

```
    xC = y2;
    mean = x;
    meanA = xA;
    meanB = xB;
    meanC = xC;
    meanD = xD;
    svs = 0;
    svsA = 0;
    svsB = 0;
    svsC = 0;
    for(int i = 1; i < n; i++)
    {
        auto results = twoNumbers();
        x =  get<0>(results);
        y =  get<1>(results);
        y1 = get<2>(results);
        y2 = get<3>(results);
        y3 = get<3>(results);
        xA = y;
        xB = (x + y1)/2.0;
        xC = y2;
        xD = y3;
        svs = svs *(i - 1)/i  + (mean - x)*(mean - x)/(i+1);
        mean = mean + (x-mean)/(i+1);
        svsA = svsA *(i - 1)/i  + (meanA - xA)*(meanA - xA)/(i+1);
        meanA = meanA + (xA-meanA)/(i+1);
        svsB = svsB *(i - 1)/i  + (meanB - xB)*(meanB - xB)/(i+1);
        meanB = meanB + (xB-meanB)/(i+1);
        svsC = svsC *(i - 1)/i  + (meanC - xC)*(meanC - xC)/(i+1);
        meanC = meanC + (xC-meanC)/(i+1);
        svsD = svsD *(i - 1)/i  + (meanD - xD)*(meanD - xD)/(i+1);
        meanD = meanD + (xD-meanD)/(i+1);
    }
    return make_tuple(mean, meanA, meanB, meanC, meanD, svs, svsA, svsB,svsC,svsD);
}

int main(){
    int N = 1000;
    srand(5);       //set up the same random seed for comparison
    auto Var = getVars(N);
    cout<<"For N = "<<N<<endl;
    cout<<"Value of the crude MC is "<<get<0>(Var)<<endl;
    cout<<"Value of Conditioning MC is "<<get<1>(Var)<<endl;
    cout<<"Value of Anthithetic MC is " <<get<2>(Var)<<endl;
    cout<<"Value of Importance sampling MC is " <<get<3>(Var)<<endl;
    cout<<"Value of Importance sampling on Conditioning MC is " <<get<4>(Var)<<endl;
    cout<<endl;
    cout<<"Variance of the crude MC is "<<get<5>(Var)<<endl;
    cout<<"Variance of Conditioning MC is "<<get<6>(Var)<<endl;
    cout<<"Variance of Anthithetic MC is " <<get<7>(Var)<<endl;
    cout<<"Variance of Importance sampling MC is " <<get<8>(Var)<<endl;
    cout<<"Variance of Importance sampling on conditioning MC is " <<get<9>(Var)<<endl;
    cout<<endl;
}
```

### 3.2 Results

```
For N = 1000
Value of the crude MC is 0.03
Value of Conditioning MC is 0.0228834
Value of Anthithetic MC is 0.028
Value of Importance sampling MC is 0.0239983
Value of Importance sampling on Conditioning MC is 0.0238737

Variance of the crude MC is 0.0291291
Variance of Conditioning MC is 0.0144949
Variance of Anthithetic MC is 0.0132292
Variance of Importance sampling MC is 0.00539017
Variance of Importance sampling on conditioning MC is 0.0053806
```