Monte Carlo estimation and Bernoulli random walks

Tomack Gilmore

1 Monte-Carlo estimation

In order to determine a Monte-Carlo estimate for the following integral

$$I = \int_0^\infty x^{3/2} e^{-x} dx$$

by choosing random variables with an exponential p.d.f $f(x) = \mu e^{-\mu x}$ we must first invert the c.d.f $y = F(x) = \int_0^x \mu e^{-\mu t} dt$. Integrating and rearranging we find

$$\begin{array}{rcl} y &=& 1 - e^{-\mu x} \\ & \Downarrow & \\ x &=& -\frac{1}{\mu}\log(1-y), \end{array}$$

so for $y \in Y \sim U(0,1)$ we can derive a value for x using the above formula that is distributed according to $f(x)$. In order to find a Monte-Carlo estimate for the integral we must then calculate the expectation

$$E\left[\frac{x^{3/2}e^{-x}}{\mu e^{-\mu x}}\right],$$

but when $\mu = 1$ this reduces to calculating

$$E[x^{3/2}] \approx \frac{1}{N}\sum_{i=1}^{N} x_i^{3/2},$$

where N is the number of random samples, x_i, we generate that are distributed according to $f(x)$. Our algorithm first generates N uniform random variables, y_i's, from $[0,1]$, then applies the formula above to each one to derive new random variables, x_i's, with a p.d.f $f(x)$. We then sum $x^{3/2}$ over the x_i's and divide by N. We then calculate the difference between this estimate and the actual value of the integral which, after a brief foray into Maple, turns out to be $\frac{3\sqrt{\pi}}{4}$. We call the following procedure (integral_monte_carlo.m) to generate an estimate with an absolute error of 0.00930829987384496:

```
N=10^5; %determines number of random samples we pick

for i=1:N
    y(i)=rand; %generates random variable from U[0,1]
    x(i)=-log(1-y(i)); %generates random variable with pdf f(x)
end

sum=0; %initialises sum as zero

for i=1:length(x)
    sum=sum+x(i)^(3/2); %sums x^(3/2) over each x(i)
end

err=abs(3*sqrt(pi)/4-sum/N);%calculates expected value i.e estimate
```

Suppose we have a random variable X that is distributed according to some p.d.f, say $g(X)$, and we have obtained a Monte-Carlo estimate for some integral I, so that

$$I = E[g(X)] \approx \frac{1}{N}\sum_{i=1}^{N} g(x_i) = \hat{g}(X).$$

It follows that $\hat{g}(X)$ is a superposition of a random variable, and so is also a random variable itself. We can therefore consider the mean and variance of our estimate, as it will fluctuate around the true value of the integral. Firstly we have

$$E[\hat{g}(X)] = E\left[\frac{1}{N}\sum_{i=1}^{N} g(x_i)\right] = \frac{1}{N}\sum_{i=1}^{N} E[g(X)] = E[g(X)] = I,$$

which shows that the Monte-Carlo estimate is an unbiased estimator of the real value of I. In a similar way we can calculate the variance of our estimate:

$$Var[\hat{g}(X)] = Var\left[\frac{1}{N}\sum_{i=1}^{N} g(x_i)\right] = \frac{1}{N^2} Var\left[\sum_{i=1}^{N} g(x_i)\right].$$

Now as each x_i is an independent random variable the right-hand side can be re-written as

$$\frac{1}{N^2}\sum_{i=1}^{N} Var[g(X)] = \frac{1}{N} Var[g(X)],$$

so we see that

$$Var[\hat{g}(X)] = \frac{1}{N} Var[g(X)] = \frac{1}{N} E[(g(X) - E[g(X)])^2] = \frac{1}{N} E[(g(X) - I)^2].$$

Now as $N \to \infty$ the variance of the estimate tends to zero, so again we see that $E[\hat{g}(X)] = E[g(X)] = I$ and it follows that $Var[\hat{g}(X)]$ provides us with an estimate of the error of the Monte-Carlo method. It is clear that in order to calculate this variance we must know the exact value of the integral I and luckily in this problem we have already found that $I = \frac{3\sqrt{\pi}}{4}$. We take the square root of the variance to find the standard deviation, or root mean square error (RMS) of an approximation. There is a slight paradox here that we need to address- although we know the integral exactly it involves π and so we cannot compute it exactly. We therefore have to take whatever Matlab approximates $3\sqrt{\pi}/4$ to be the best possible estimate of I, and we then estimate the variance by

$$\tilde{Var}[g(X)] = \frac{1}{N-1} E[(g(X) - \hat{g}(X))^2],$$

where $\hat{g}(X)$ is an estimate of the integral.

So we find this error by first generating a vector x of N random variables with the correct p.d.f as in part (a). We then sum $\left(g(x) - \frac{3\sqrt{\pi}}{4}\right)^2$, where $g(x) = x^{3/2}$, over each element in x and divide by $N(N-1)$ to find the variance. We then take the square root to generate a root mean square error of 0.00656140227531517¡0.01. We implement this by calling the procedure var_monte_carlo_rms.m:

```
N=10^5; %specifies number of random samples we pick

for i=1:N
    y(i)=rand; %generates random variable in U[0,1]
    x(i)=-log(1-y(i)); %transforms y into a variable with specified pdf
end

var=0; %initialises variance as zero

for i=1:length(x)
    var=var+(x(i)^(3/2)-3*sqrt(pi)/4)^2; %sums distance of each x(i) from
end                                        %true value of I

rms=sqrt(var/(N*(N-1)) %generates root mean square error of estimates
```

In order to compare this with the absolute error we consider the errors for an increasing number of samples by implementing the following procedure (rms_abs_plot.m):

```
for n=1:6

   N=10^n; %specifies sample size

    for i=1:N
```

```
          y(i)=rand; %generates random variable from U[0,1]
          x(i)=-log(1-y(i)); %generates random variable with pdf f(x)
     end

     sum=0; %initialises sum as zero
     var=0; %initialises variance as zero

     for i=1:length(x)
          sum=sum+x(i)^(3/2); %sums g(x(i)) over each x(i)
          var=var+(x(i)^(3/2)-3*sqrt(pi)/4)^2; %sums distance of each x(i)
     end                                        %from true value of I

     rms(n)=sqrt(var/(N*(N-1))); %calculates root mean square error for
                                %for corresponding n value
     err(n)=abs(3*sqrt(pi)/4-sum/N); %calculates absolute error for
                                     %corresponding n value
     ns(n)=N; %generates vector containing each value of n

end

plot(log(ns),log(err),'+',log(ns),log(rms),'x') %generates loglog plot
```

We call this function three times to generate the plots found in Figure 1. What is clear from these plots is the rate at which the RMS decreases- the gradient is roughly that of $-1/2$ so the RMS$\sim N^{-1/2}$ which implies that in order to reduce the error by a factor of 10 we need 100 times more sampling points. Another thing to note is that as we increase the sample size the RMS is generally larger than the absolute error. This makes sense as the RMS is the standard deviation of *an* approximation of I. So the RMS error is the maximum error of the majority of our approximations of I i.e. the majority of approximations we make will lie within the RMS error of the true value of I.

In order to sample points with various values for some scalar μ we generate a vector, mu, comprising the different values we want to look at. We then repeat the procedure from part (a) to calculate a Monte-Carlo estimate for each μ, i.e,

$$I \approx \frac{1}{N}\sum_{i=1}^{N} \frac{x^{3/2}e^{-x}}{\mu e^{-\mu x}}.$$

Similarly we use the same type of procedure from part (b) to calculate the RMS of the same sample as that used for the Monte-Carlo estimate for each entry in mu. We implement the following procedure (monte_carlo_mu.m):

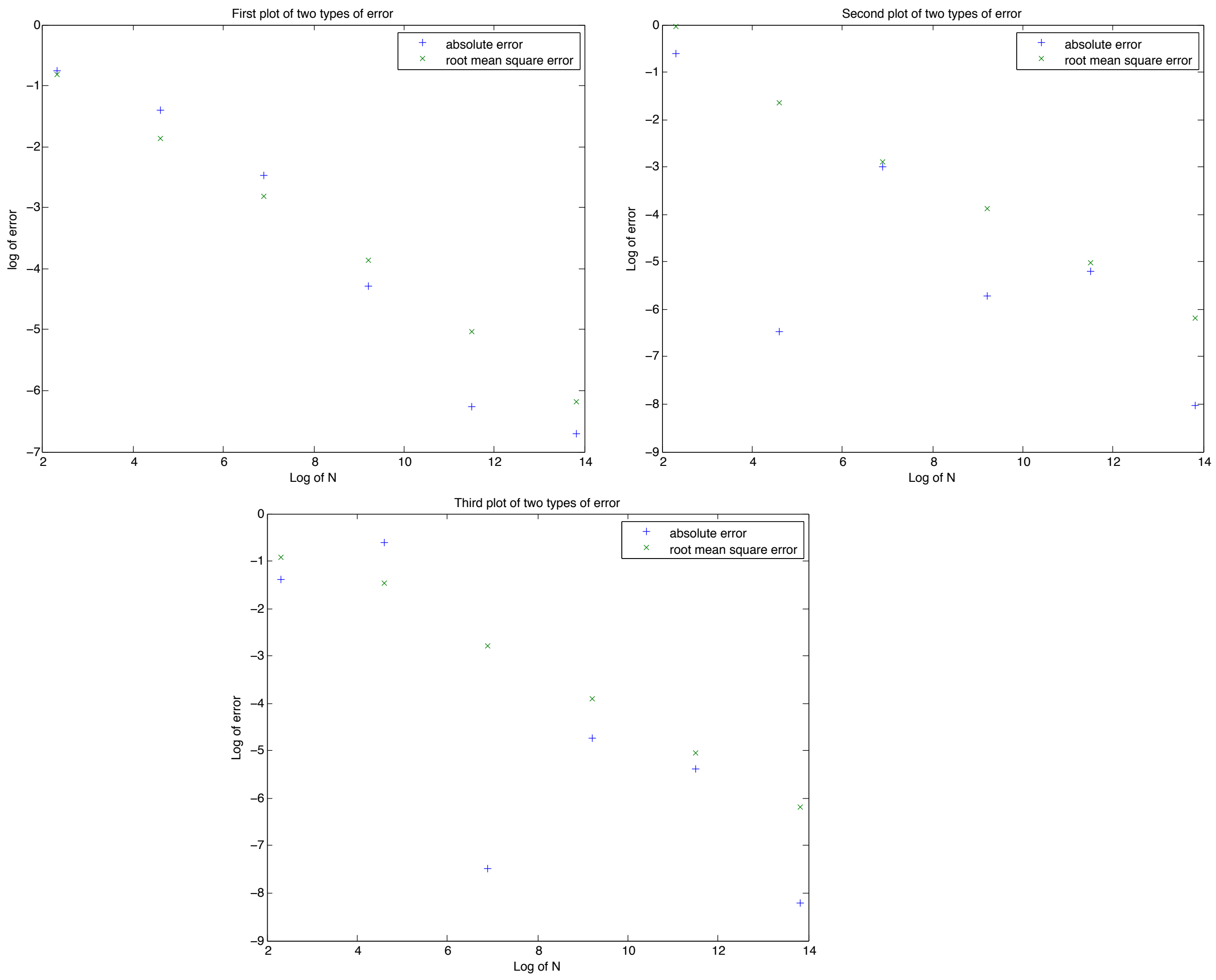

First plot of two types of error
absolute error
root mean square error
log of error
Log of N
Second plot of two types of error
absolute error
root mean square error
Log of error
Log of N
Third plot of two types of error
absolute error
root mean square error
Log of error
Log of N

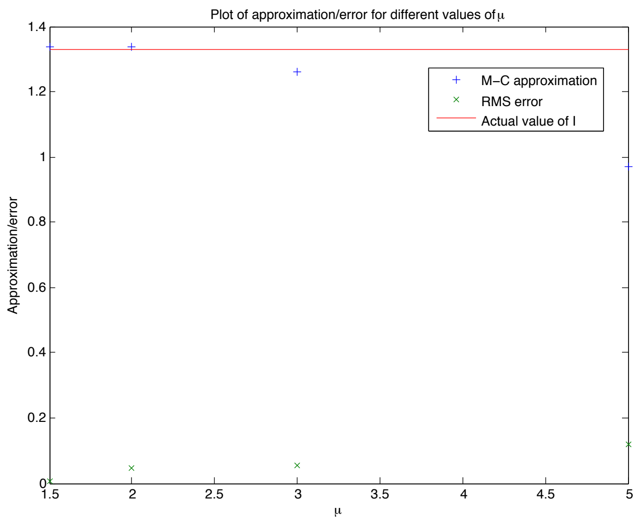
Figure 1

```
mu=[1.5, 2.0, 3.0, 5.0]; %generates vector mu
N=10^6; %sets the sample size

for i=1:length(mu) %runs over entries in mu
     m=mu(i); %sets ith entry of mu as m
       for j=1:N
            y(j)=rand; %generates random variable
            x(j)=-log(1-y(j))/m; %converts y(i) to random variable with pdf
                                  %for corresponding mu
       end

       sum=0; %sets sum as zero for mc estimate
       var=0; %sets variance as zero for RMS error

       for j=1:length(x)
            sum=sum+(x(j)^(3/2)*exp(-x(j)))/(m*exp(-m*x(j))); %sums function
            %of random variables in x for specific m
            var=var+((x(j)^(3/2)*exp(-x(j)))/(m*exp(-m*x(j)))-3*sqrt(pi)/4)^2;
            %calculates variance of rv's in x for specific m
       end

       mc(i)=sum/N; %generates vector of M-C estimate for ith entry of mu
       rms(i)=sqrt(var/(N*(N-1))); %generates vector of RMS for ith entry of
end                                %mu
plot(mu,mc,'+',mu,rms,'x',[1.5,5],[3*sqrt(pi)/4, 3*sqrt(pi)/4])
%plots the estimate, rms error and actual value of I for each mu
```

This produces the plot found in Figure 2 (left). Clearly as μ increases the RMS error and our Monte-Carlo approximation deviate further and further from the actual value of the integral. To see why this is we should consider the pd.f's, $f_\mu(x) = \mu e^{-\mu x}$, for each μ, together with the function $h(x) = x^{3/2}e^{-x}$. To do this we execute the following in Matlab:

```
>> tspan=[0:0.001:6];
>> plot(tspan,tspan.^(3/2).*exp(-tspan),tspan,1.5.*exp(1.5.*(-tspan)),
tspan,2.*exp(2.*(-tspan)),tspan,3.*exp(3.*(-tspan)),tspan,
5.*exp(5.*(-tspan)))
```

(Figure 2, right). Clearly as μ increases the function $f_\mu(x)$ behaves less and less like the function $h(x)$. This means that as we increase μ we increase the proportion of possible points that lie outside the original function we are trying to estimate. So increasing μ decreases how close $f_\mu(x)$ is to the original function $h(x)$. Therefore we may conclude that the best p.d.f to choose for importance sampling is a p.d.f that is as close as possible to the integrand.



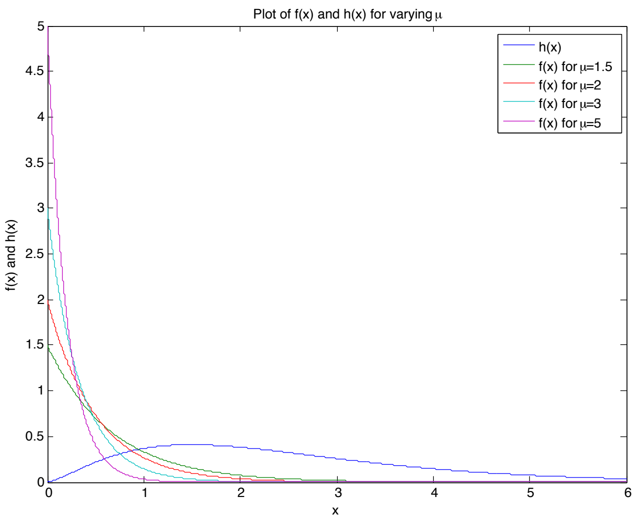


Figure 2: (Left) Plot of Monte-Carlo estimate and RMS for increasing μ. (Right) Plot of $f_\mu(x)$ for each μ together with the original function $h(x)$.

2 Monte-Carlo with different distributions

In order to determine 5000 Monte-Carlo estimates of the integral

$$I = \int_{-\infty}^{\infty} \frac{e^{x^2/2}}{\sqrt{2\pi}} dx = 1$$

using importance sampling from the distribution $U(-10, 10)$ we first run a loop using j from 1 to 5000 which will run over the vector entries of the vector mc that will contain all 5000 estimates. Inside this loop we generate a vector (y) of 1000 random variables from $U(0, 1)$ using the built-in function 'rand'. We then superposition these entries in a new vector x by multiplying each entry of y by 20 and subtracting 10. This maps the unit interval into the interval [-10,10] completely.

We then calculate $E[h(x_i)/f(x_i)]$ for each x_i in the vector x. The p.d.f of our random variable is given by the function $f(x) = 1/20$, and $h(x)$ is the integrand of I, so we have

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{e^{x^2/2}}{\sqrt{2\pi}} \frac{1}{f(x)} = \frac{1}{N} \sum_{i=1}^{N} \frac{20e^{x^2/2}}{\sqrt{2\pi}}.$$

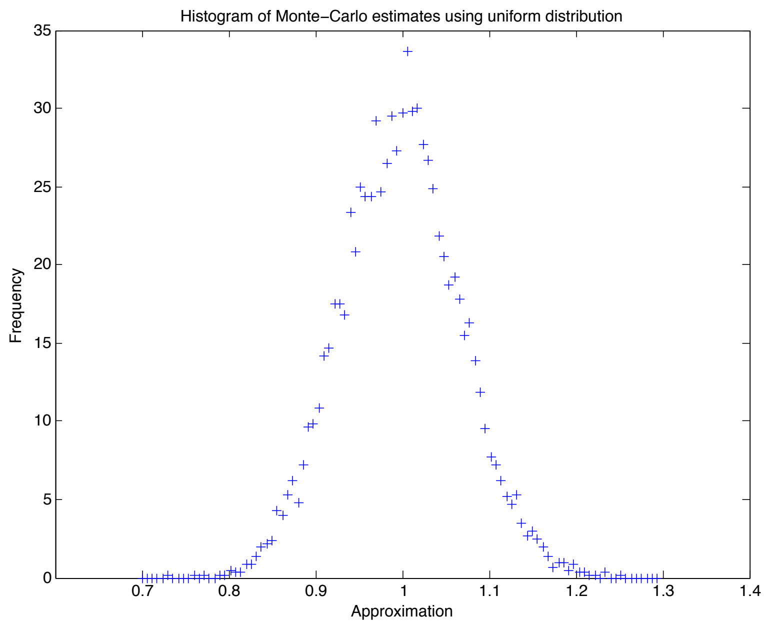
Let us call the function in the summand $g(x)$. So we then sum $g(x_i)$ over each entry of the vector x and divide by the number of estimates, 1000. This provides us with an approximation that we store in the jth position of the vector mc. This process repeats until we have a vector mc with 5000 entries, each of them an approximation of I. We then run through much the same process as that of the previous coursework in order to generate a histogram using the function *myhist*(mc, a, b, n) over a specified interval (a, b) around 1 with n bins. We call the following procedure (monte_carlo_uniform_plot.m) to do this, which results in the plot seen in Figure 3 (top):

```
N=1000; %sets sample size for approximation

for j=1:5000 %loops over number of estimates we want to generate
    sum=0; %sets sum as 0
    for i=1:N %loops over sample
        y(i)=rand; %generates random number from U(0,1)
        x(i)=20*y(i)-10; %superpositions y(i) so that x~U(-10,10)
        sum=sum+20*exp(-x(i)^2/2)/(sqrt(2*pi)); %sums g(x(i)) for each x(i)
    end
    mc(j)=sum/N; %assigns m-c approximation to jth position in vector mc
end

n=100; %sets number of bins
a=0.7; %sets minimum of range for histogram
b=1.3; %sets maximum of range for histogram
dx=(b-a)/n; %sets size of each bin
xrange=[a:dx:b-dx]; %specifies range as vector using left endpoints of bins

h=myhist(mc,a,b,n); %calls function myhist

for i=1:length(h) %runs over entries of h
    ncountsu(i)=h(i)/(N*dx); %appropriately scales entries of h
end
plot(xrange,ncountsu,'+') %generates plot
```

We apply much the same process when sampling points using the Lorentz distribution with $\alpha = 0$ and $\beta = 1$. Again we run a loop, j, from 1 to 5000 which will run over the vector entries of the vector mc that will contain all 5000 estimates. Inside this loop we generate a vector, y, of 1000 random variables from $U(0, 1)$ using the built-in function 'rand'. For each element of y we then use the inversion formula that we found in the previous coursework to generate a corresponding random variable x_i that is distributed according to the Lorentz distribution, i.e.

$$x_i = \beta \tan\left(\pi y_i + \tan^{-1}\left(\frac{\alpha}{\beta}\right)\right) + \alpha.$$

So for $\alpha = 0$ and $\beta = 1$ we simply have $x_i = \tan(\pi y)$. We then sum $g(x_i) = h(x_i)/f(x_i)$ for each x_i in x and divide by $N = 1000$, where $h(x)$ is again the integrand of I and $f(x)$


Histogram of Monte−Carlo estimates using uniform distribution
Frequency
Approximation
0
5
10
15
20
25
30
35
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4

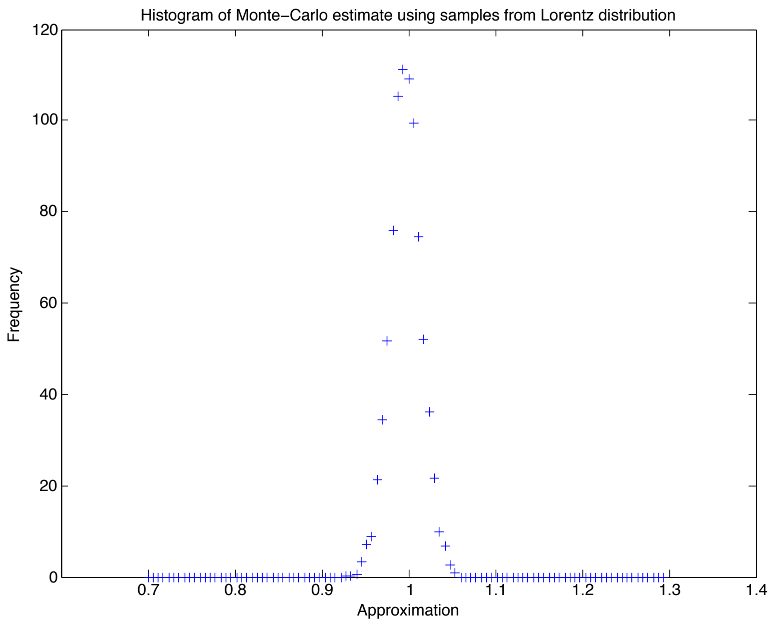

Histogram of Monte−Carlo estimate using samples from Lorentz distribution
Frequency
Approximation
0
20
40
60
80
100
120
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4

Figure 3

is now the function

$$f(x) = \frac{1}{\pi\,(1+x^2)} \quad \Rightarrow \quad I \approx \frac{1}{N}\sum_{i=1}^{N} \frac{\pi(1+x_i^2)e^{-x_i^2/2}}{\sqrt{2\pi}}.$$

Each estimate is then stored in the jth entry of the vector mc which eventually contains 5000 entries. We then proceed as above to generate a histogram. To do this we call the procedure (monte_carlo_lorentz_plot.m) which produces the plot found in Figure 3 (bottom):

```
alpha=0; %sets value for alpha
beta=1; %sets value for beta
N=1000; %sets sample size
for j=1:5000 %runs from 1 to 5000
    sum=0; %initialises sum as zero
    for i=1:N %runs over sample size
        y(i)=rand; %generates a random variable from [0,1]
        x(i)=beta*tan(pi*y(i)+atan(-alpha/beta))+alpha; %converts y(i) to
        sum=sum+pi*exp(-x(i)^2/2)*(1+x(i)^2)/sqrt(2*pi); %random variable
    end %from Lorentz distribution
    mc(j)=sum/N; %sets estimate as jth entry of vector mc
end

n=100; %sets number of bins
a=0.7; %sets minimum of range for histogram
b=1.3; %sets maximum of range for histogram
dx=(b-a)/n; %sets size of each bin
xrange=[a:dx:b-dx]; %specifies range as vector using left endpoints of bins

h=myhist(mc,a,b,n); %calls function myhist

for i=1:length(h) %runs over entries of h
    ncountsl(i)=h(i)/(N*dx); %appropriately scales entries of h
end
plot(xrange,ncountsl,'+') %generates plot
```

It is clear from Figure 3 that the approximation obtained by sampling from the uniform distribution deviates further from the true value of I than the approximation obtained using the Lorentz distribution. This should be even more obvious by looking at Figure 4. This shows that the majority of our approximations obtained using the Lorentz distribution lie well within 0.05 of the true value of I whereas the majority of approximations obtained using the uniform distribution are spread over a relatively large interval. This makes absolute sense given that the Lorentz distribution with $\alpha = 0$ and $\beta = 1$ is much closer in shape to the integrand of I than the uniform distribution (see Figure 4, top). This means a higher proportion of points sampled using the Lorentz distribution will actually lie under the actual curve defined by the integrand of I. A lower proportion of samples from the uniform distribution lie under the same curve, hence it is a worse approximation (see Figure 4, bottom).

3 Bernoulli random walks

A Bernoulli random walk is defined by

$$x_n = x_{n-1} + y,$$

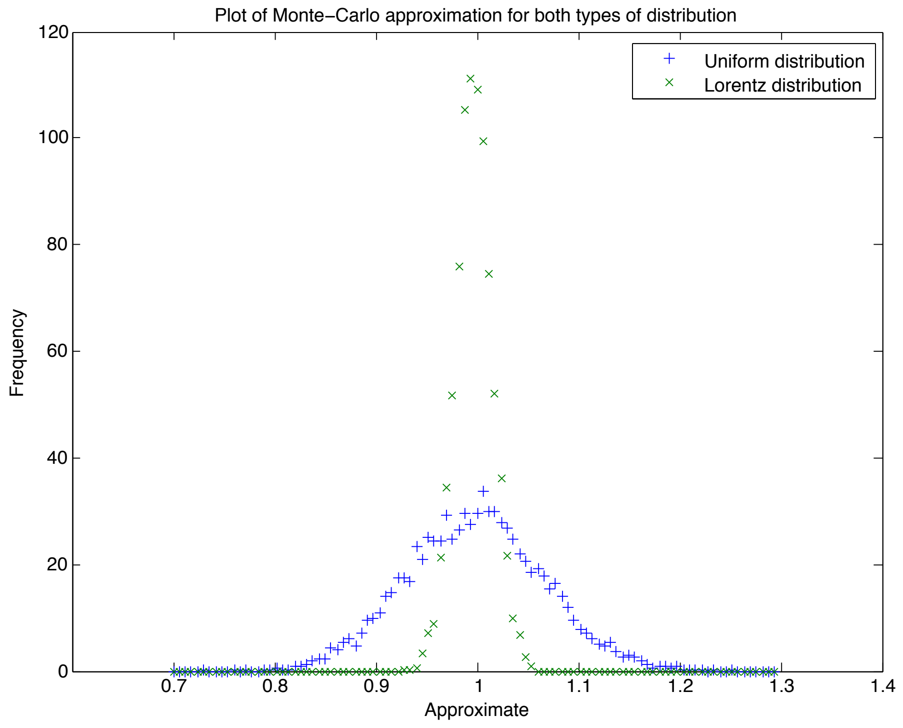
where y is the jump variable such that $P(y = 1) = p$ and $P(y = -1) = 1 - p$ for some probability p. If we are given $x_0 = 0$ and $p = 0.3$ we can generate a simple random walk by generating a column vector x with initial entry 0. For each subsequent ith entry we generate a random number $y \sim U(0, 1)$ and test to see if it is less than or equal to our probability $p = 0.3$. If so then we take the $(i-1)$th entry and and 1 to it and we subtract 1 if $y > p$. This then becomes the ith entry of the column vector x. We repeat the process N many times to generate a random walk of N time steps. We then repeat this entire process 100 times to generate a matrix x with column vectors that are all simple random walks of N many time-steps. Figure 5 (top) shows 10 random paths for $N = 20$ time-steps. We generate this plot by executing the following command:

```
>> plot([0:20],x(:,1:10))
```

which plots the simple paths in the first 10 columns of the matrix x. We then apply the same procedure involving the function $myhist(v, a, b, n)$ where v is the 5th row of the matrix x, a is -5, b is 6 and $n = 11$. So each bin has size 1, and we count how many paths are in each bin after 5 time-steps. In order to compare this with the actual binomial distribution $P(x, n)$ we consider the distribution of the steps to the left (i.e. steps of +1) after n time-steps. If we let l be the number of steps to the left then after n time-steps we have $\binom{n}{l}$ ways of making l left steps, each step occurs with probability p and we have $n-l$ steps to the right that occur with probability $(1 - p)$. So is the probability of observing l left steps after n time-steps is

$$P(l, n) = \binom{n}{l} p^l (1 - p)^{n-l}.$$

However we want $P(x, n)$, the probability that $x_n = x$ after n time-steps. We have that the displacement after n time-steps, x, is the number of left steps minus the number of right steps (i.e steps of -1, say r). So $x = l - r$ We also have that the total number of time-steps


Plot of Monte−Carlo approximation for both types of distribution
Uniform distribution
Lorentz distribution
Frequency
Approximate
0
20
40
60
80
100
120
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4

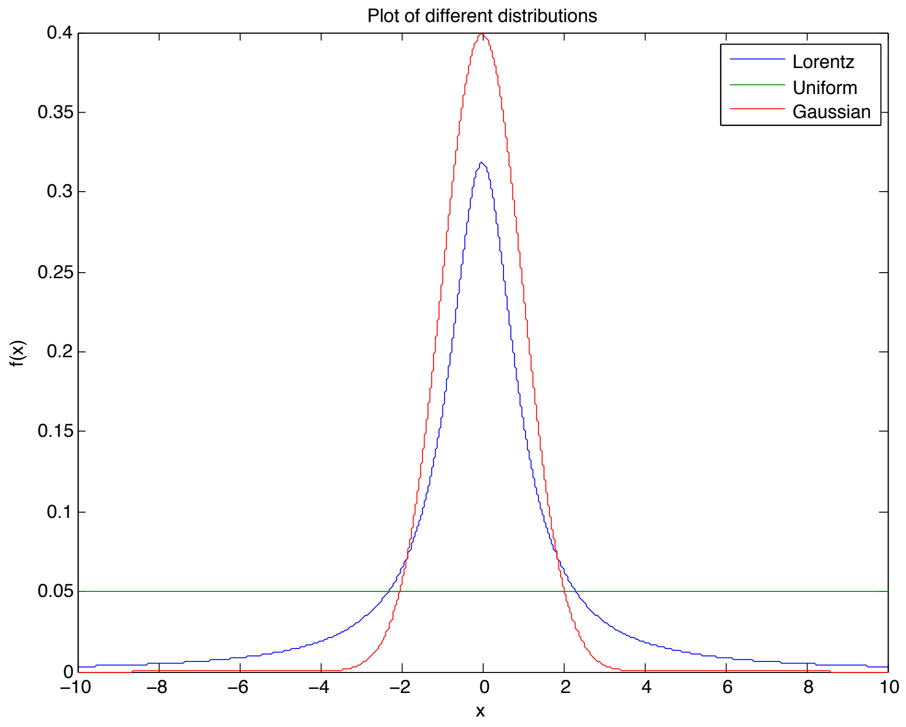

Plot of different distributions
Lorentz
Uniform
Gaussian
f(x)
x
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
−10
−8
−6
−4
−2
0
2
4
6
8
10

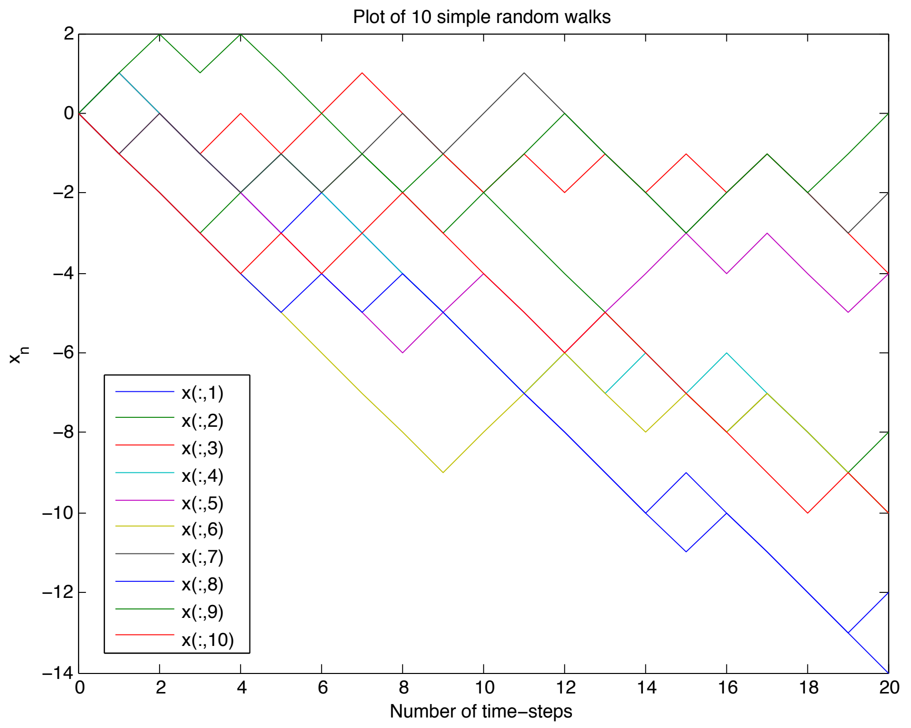
Figure 4

is $n = l + r$, i.e $r = n - l$, so $x = 2l - n$, so $l = (x + n)/2$. Replacing our expression for l in the above formula yields

$$P(x, n) = \binom{n}{\frac{1}{2}(x+n)} p^{\frac{1}{2}(x+n)} (1-p)^{\frac{1}{2}(n-x)}.$$

We calculate this explicitly and plot it together with the histogram (Figure 5, bottom). Clearly after 5 steps the displacement will be a maximum of 5 and a minimum of -5. The plot of 10 paths shows that as the number of time-steps increases every path tends towards negative numbers which makes sense given that roughly 70% of the time we are contributing -1 to the displacement. This is also reflected in the histogram, which follows the binomial distribution fairly closely. Obviously there is a zero chance of the displacement being an even number after an odd number of steps. Also the histogram is skewed to the left, indicating that the bias of p, i.e we are more likely to add -1 to the displacement at each time step. If we were to change p to a half we would expect to see an even distribution around zero, and if we let $p > 1/2$ then we would expect to see a skew to the right. As we take more and more sample paths and increase the number of time-steps these histograms will look increasingly like the actual binomial distribution. We do all of this by calling the procedure bernoulli_binom_plot.m:

```
n=100; %sets number of sample paths
N=20; %sets number of time-steps
p=0.3; %sets proabability p

for j=1:n %loops over the number of paths
    for i=1:N; %loops over time-steps
        x(1,j)=0; %sets initial value as 0
        y=rand; %generates random number ~U(0,1)
        if y<=p %checks if y<=p
            xn=1; %sets xn to 1 if so
        else
            xn=-1; %sets xn to -1 if not
        end
        x(i+1,j)=x(i,j)+xn; %sets ith value of column vector as x(i-1)+xn
    end
end

n1=11; %sets number of bins
a=-5; %sets minimum value of range of histogram
b=6; %sets maximum value of range for histogram
dx=(b-a)/n1; %sets size of each bin
v=x(5,:); %sets v as 5th row of the vector x
```


Plot of 10 simple random walks
x_n
Number of time−steps
x(:,1)
x(:,2)
x(:,3)
x(:,4)
x(:,5)
x(:,6)
x(:,7)
x(:,8)
x(:,9)
x(:,10)

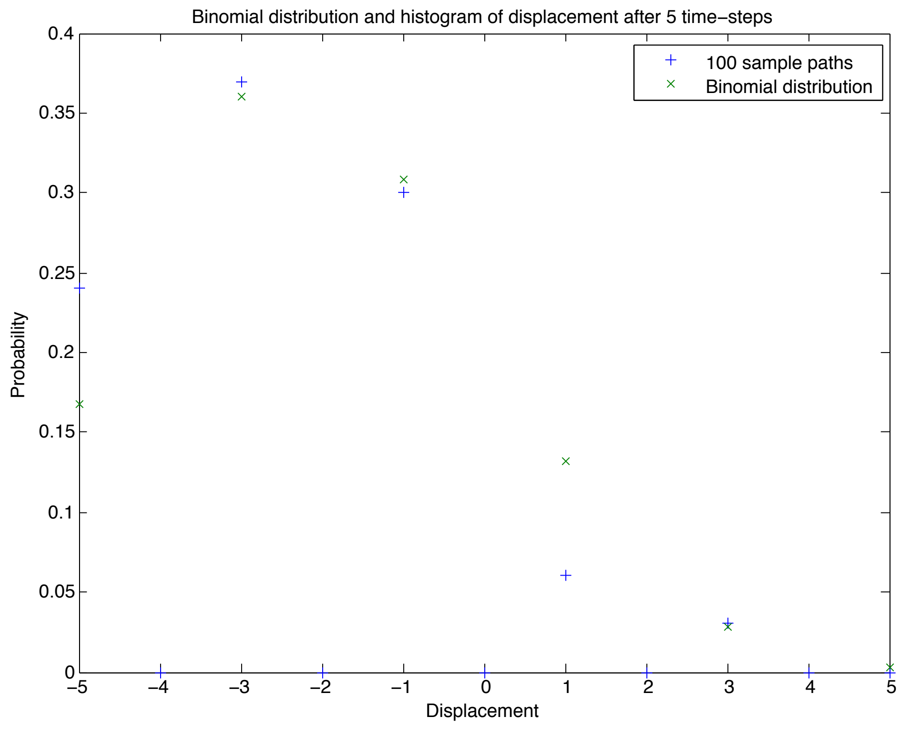

Binomial distribution and histogram of displacement after 5 time−steps
Probability
Displacement
100 sample paths
Binomial distribution

Figure 5

```
xrange=[a:dx:b-dx]; %specifies range for histogram

h=myhist(v,a,b,n1); %calls function myhist that generates a vector h

for i=1:length(h)
    ncounts(i)=h(i)/(n*dx); %scales h appropriately
end

n=5; %sets value of n for actual binomial distribution
x1=[-5:2:5]; %generates range of values for the displacement x1
for i=1:length(x1) %runs over entries of x1
    y(i)=mybinom(n, (x1(i)+n)/2)*p^((x1(i)+n)/2)*(1-p)^((n-x1(i))/2);
end%calculates the actual probability of observing x1(i) after n time-steps
plot(xrange,ncounts,'+',x1,y,'x') %plots histogram and actual pdf
```

4 The Weiner process

Given the linear stochastic differential equation

$$dx(t) = -xdt + dW(t)$$

where $dW(t)$ represents the increment of the Wiener process. In order to obtain the stationary probability density of the solutions of this equation we generate n random walks over a time interval $[0, T]$ in steps of size dt with the initial condition x_0. The walks form the columns of the matrix x where each ith entry of the columns is given by the equation

$$x_i = x_{i-1} + x_{i-1}dt + \sqrt{dt}(y),$$

where $y \sim N(0, 1)$ is generated using the function "randn". We then make a histogram of the last row of x which comprises the endpoints of each walk in a similar way as before to generate the plot found in Figure 6 (top). We do implement this by calling the procedure weiner_plot.m:

```
n=500; %sets number of random walks generated
t0=0; %sets initial time
T=500; %sets final time
dt=0.001; %sets size of timesteps
tspan=[t0:dt:T]; %sets vector of time steps

x0=0; %sets initial condition as zero
for j=1:n %loops over number of random walks
    x(1,j)=x0; %sets inital condition as first entry in jth column of x
    for i=1:length(tspan)-1 %runs over time increments
        x(i+1,j)=x(i,j)-(x(i,j)*dt)+sqrt(dt)*randn; %generates random walk
    end
end

n1=100; %sets number of bins
a=-2.5; %sets minimum value of range of histogram
b=2.5; %sets maximum value of range for histogram
dx=(b-a)/n1; %sets size of each bin
v=x(i+1,:); %sets v as final row of matrix x
xrange=[a:dx:b-dx]; %specifies range for histogram

h=myhist(v,a,b,n1); %calls function myhist that generates a vector h

for i=1:length(h)
    ncounts(i)=h(i)/(n*dx); %scales h appropriately
end

plot(xrange,ncounts,'+',xrange,exp(-xrange.^2)/sqrt(pi)) %generates plot
```

In order to plot the correlation function

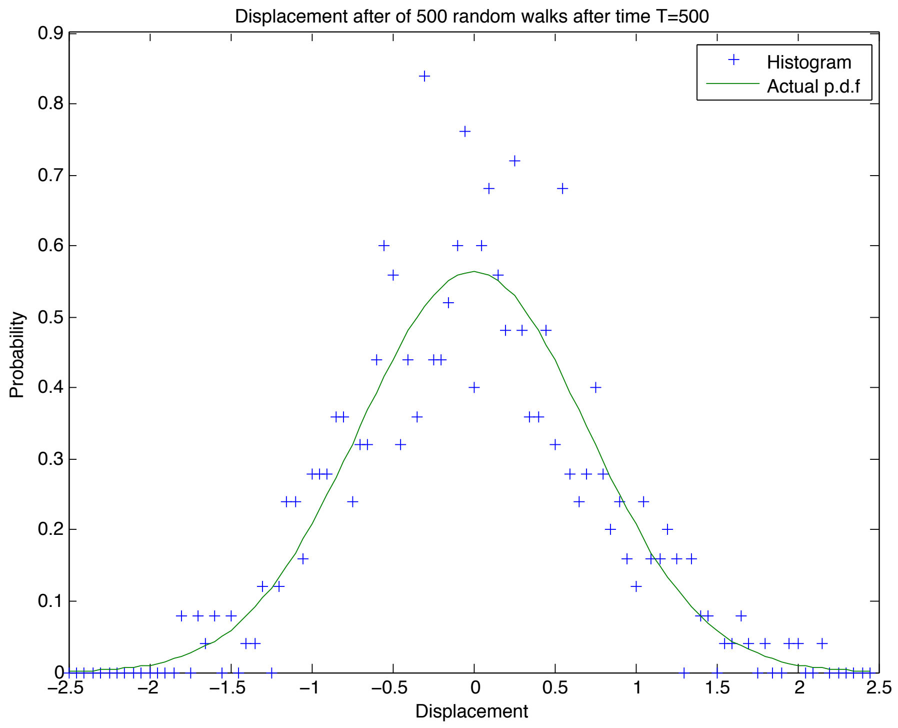
$$\phi(\tau) = E[x(t_1)x(t_1+\tau)],$$

we choose some value for $t_1 > t_0$ and then generate a vector "tau" that runs from 0 to $(T - t_1)$. We then consider the t_1th point in each path and calculate the expectation

$$E[x_{i,j}x_{(i+\tau,j)}] = \frac{1}{n}\sum_{i,j} x_{i,j}x_{(i+\tau,j)},$$

where $i = t_1, ..., T$ and $j = 1, ..., n$ for each τ in the vector tau. we do this by implementing the following procedure (corr.m):

```
t1=250; %sets value for t1
nt=T-t1; %calculates difference between t1 and T
tau=[0:nt]; %generates vector tau

for i=1:length(tau) %runs over entries of tau
    sum=0; %sets sum to 0
    for j=1:n %runs over columns of x
        sum=sum+x(t1,j)*x(t1+tau(i),j); %calculates sum
    end
    E(i)=sum/n; %calculates expected value for each tau
```


Displacement after of 500 random walks after time T=500
Histogram
Actual p.d.f
Probability
Displacement
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−2.5
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
2.5

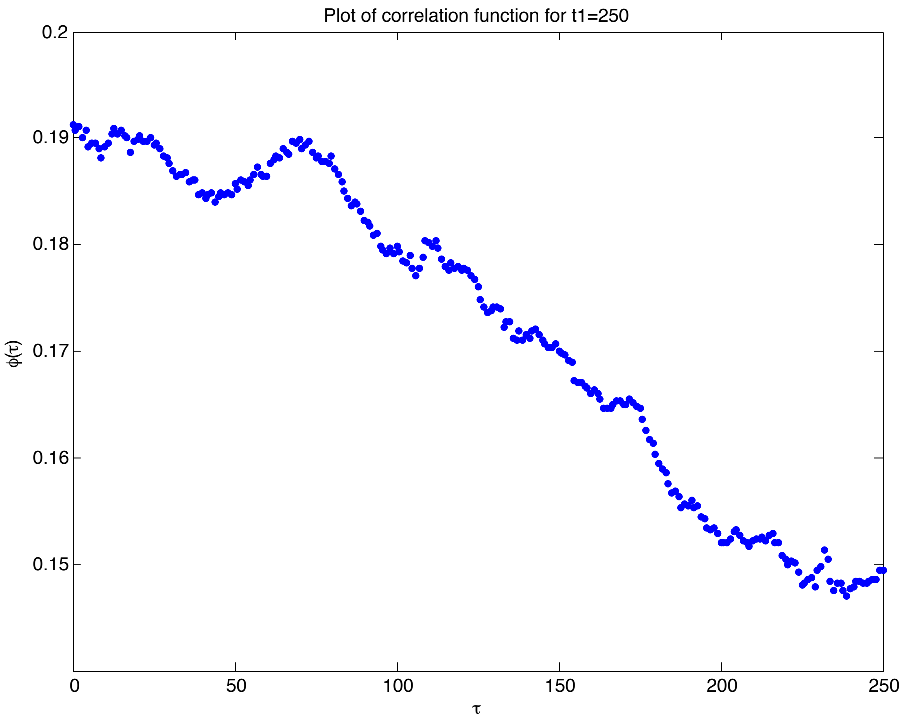

Plot of correlation function for t1=250
φ(τ)
τ
0.2
0.19
0.18
0.17
0.16
0.15
0
50
100
150
200
250

Figure 6

```
end
plot(tau,E,'.'); %generates plot
```

This produces the plot at the bottom of Figure 6 and we can clearly see that as time increases the histogram of the random walks at time T converges to the actual pdf for large T. The plot of the correlation function is decaying which confirms that the distribution becomes independent of time as T increases.