**RANDOM WALK IN 1D**

**Code written in Matlab**

clear

%% **plot 300 trajectories with 100 hops**

N = 100;

pos = zeros(100,300); % initial condition all aprticles at x=0;

for i = 2:100

for j = 1:300

a=sign(randn);

pos(i,j)= (pos((i-1),j)+a);

end

end

subplot(2,1,1)

plot(pos)

xlabel('hops')

ylabel('position (x)')

%% **plot histogram of last position of the particle**

subplot(2,1,2)

last\_pos = pos(100,:);

histogram(last\_pos);

xlabel('Final Position')

ylabel('Number of Particles')

%% **Compare with Binomial Distribution**

hold on

x = 0:100;

a = binopdf(x,100,0.5);

pos\_bin = (a\*300);

m = -100:2:100;

plot(m,pos\_bin,'r+');

%% **Compare with normal Approximation**

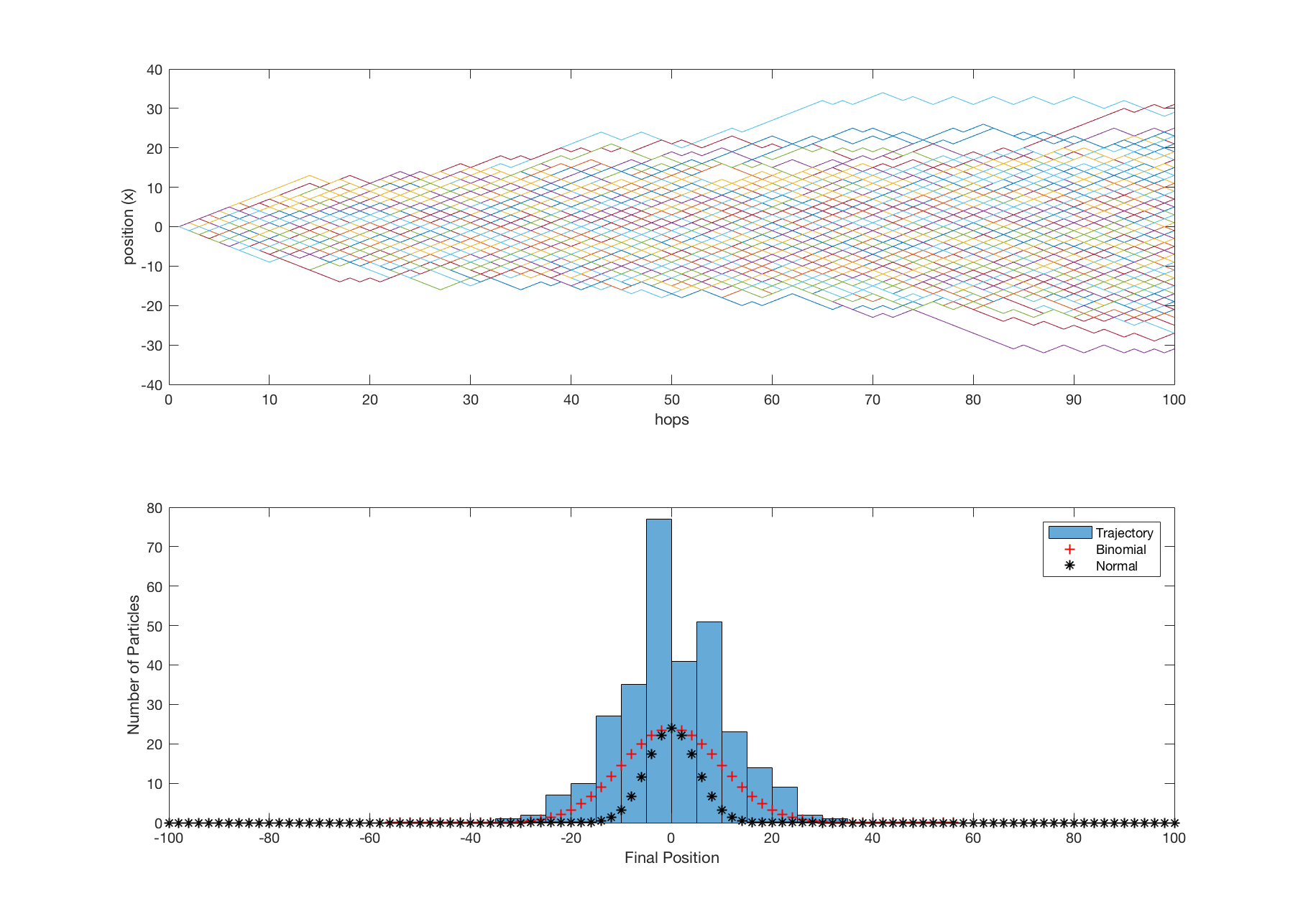
y = -100:2:100;

b = normpdf(y,0,5);

pos\_norm = b\*300;

plot(m,pos\_norm,'k\*')

legend('Trajectory','Binomial','Normal');



**RANDOM WALK IN 3D**

%% Simulate RANDOM WALK IN 3D

r2 = 0; % store the r^2 for mean square displacement

x0 = 0;

y0 = 0;

z0 = 0;

pos\_x = zeros(100,3000);

pos\_y = zeros(100,3000);

pos\_z = zeros(100,3000);

% say each step the total lenght is sqrt(3)

for i = 2:100

for j = 1:3000

a=sign(randn);

b=sign(randn);

c=sign(randn);

pos\_x(i,j)= (pos\_x((i-1),j)+a);

pos\_y(i,j)= (pos\_y((i-1),j)+b);

pos\_z(i,j)= (pos\_z((i-1),j)+c);

end

end

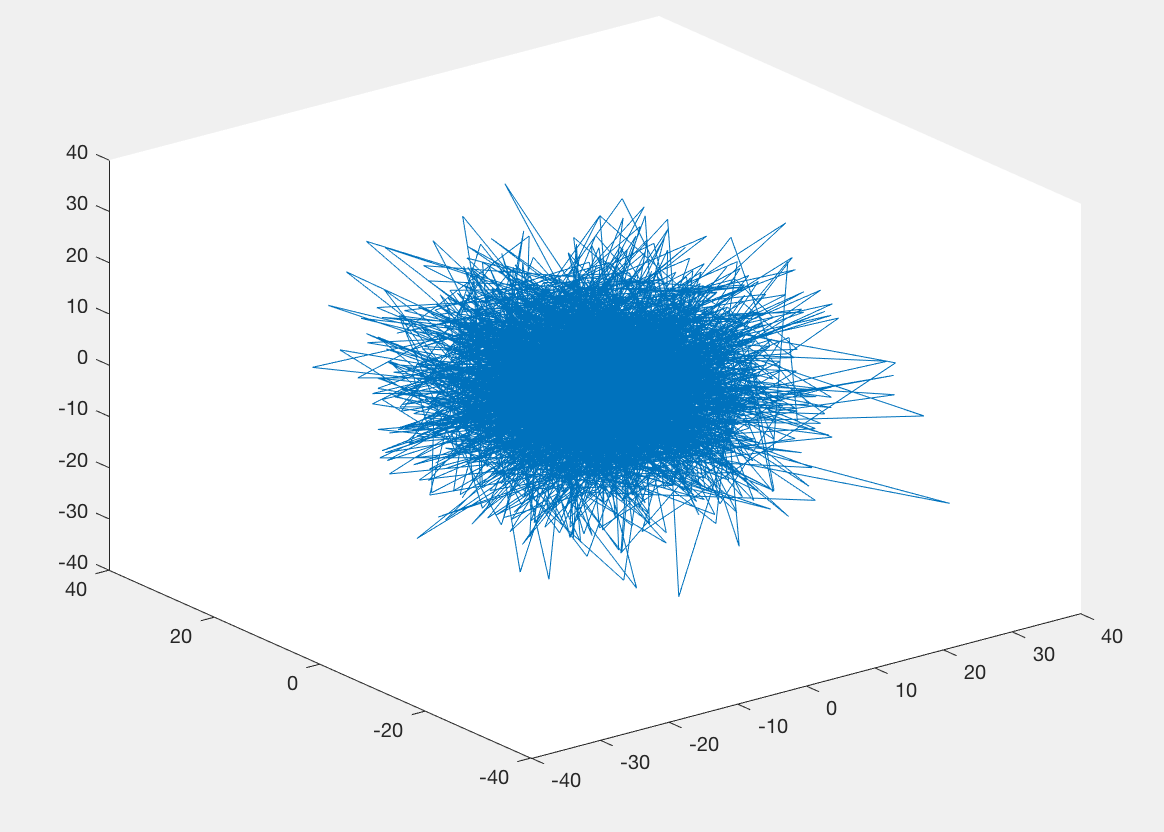
last\_x\_pos = pos\_x(100,:);

last\_y\_pos = pos\_y(100,:);

last\_z\_pos = pos\_z(100,:);

g = [last\_x\_pos;last\_y\_pos;last\_z\_pos];

plot3(g(1,:),g(2,:),g(3,:));



**Now determine the mean square displacement scales with time**

%% mean square displacement

for i = 1:100

for j = 1:3000

r2(i,j) = pos\_x(i,j)^2+pos\_y(i,j)^2+pos\_z(i,j)^2;

end

end

for i = 1:100

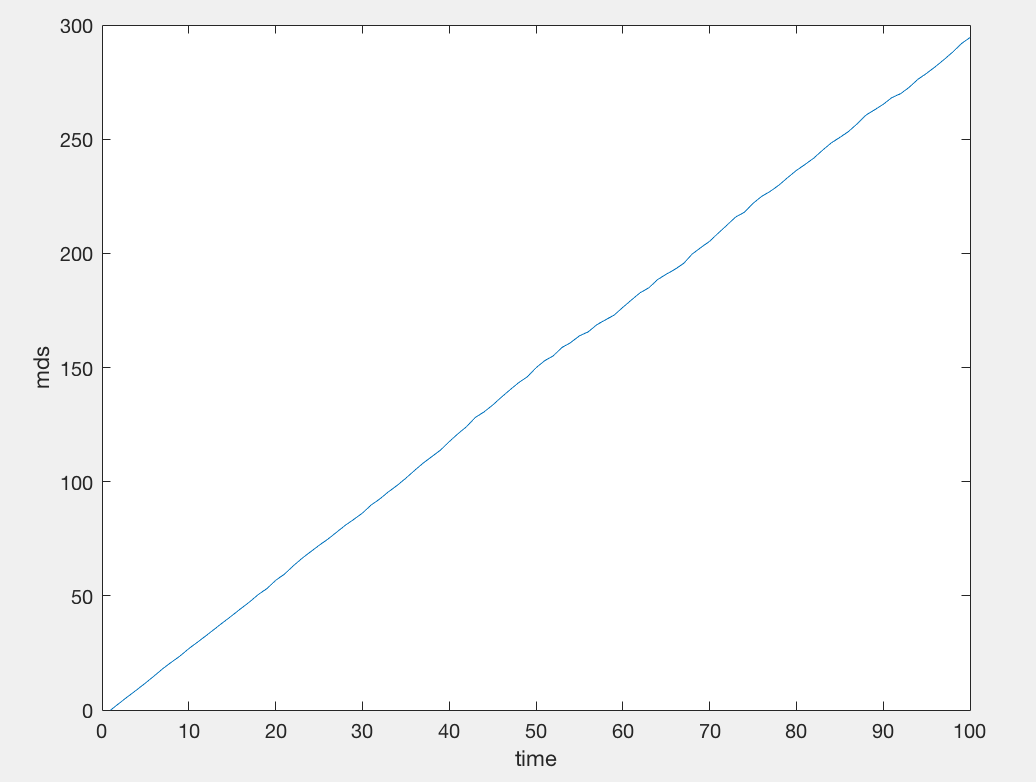
msd(i) = mean(r2(i,:));

end

plot(msd);

xlabel('time');

ylabel('mds');



%% **Calculate Diffusivity**

% D = (l^2)/(2\*tau)

% slope is 300/100;

% D = 300/100/6 = 1/2

**Write MATLAB code that implements a stochastic algorithm for simulating this process.**

% Birth Death Model for gene expression

%% A stochastic algorithm for simulating this process

lamda = 2; %min^-1

k = 0.1; %min^-1

t\_end = 5/k;

t\_ini = 0;

%% Get the time point the next event happens, assume only 1 event happens in that interval

for i=1:2

t =[];

t(1) = t\_ini;

N =[];

N(1) =0;

t\_new = 0;

while t\_new < t\_end

a = rand(1); % first generate a r.v

ti = -log(1-a)/(lamda+k\*N(end)); % intervt it and get the nth time point

t\_new = t\_new + ti;

t = [t t\_new];

p\_trans = lamda/(lamda+k\*N(end)); % Determine whether transcript or degradation happens

b = rand(1);

num\_generated = 2\*(b <= p\_trans)-1;

n = N(end) + num\_generated;

N = [N, n];

end

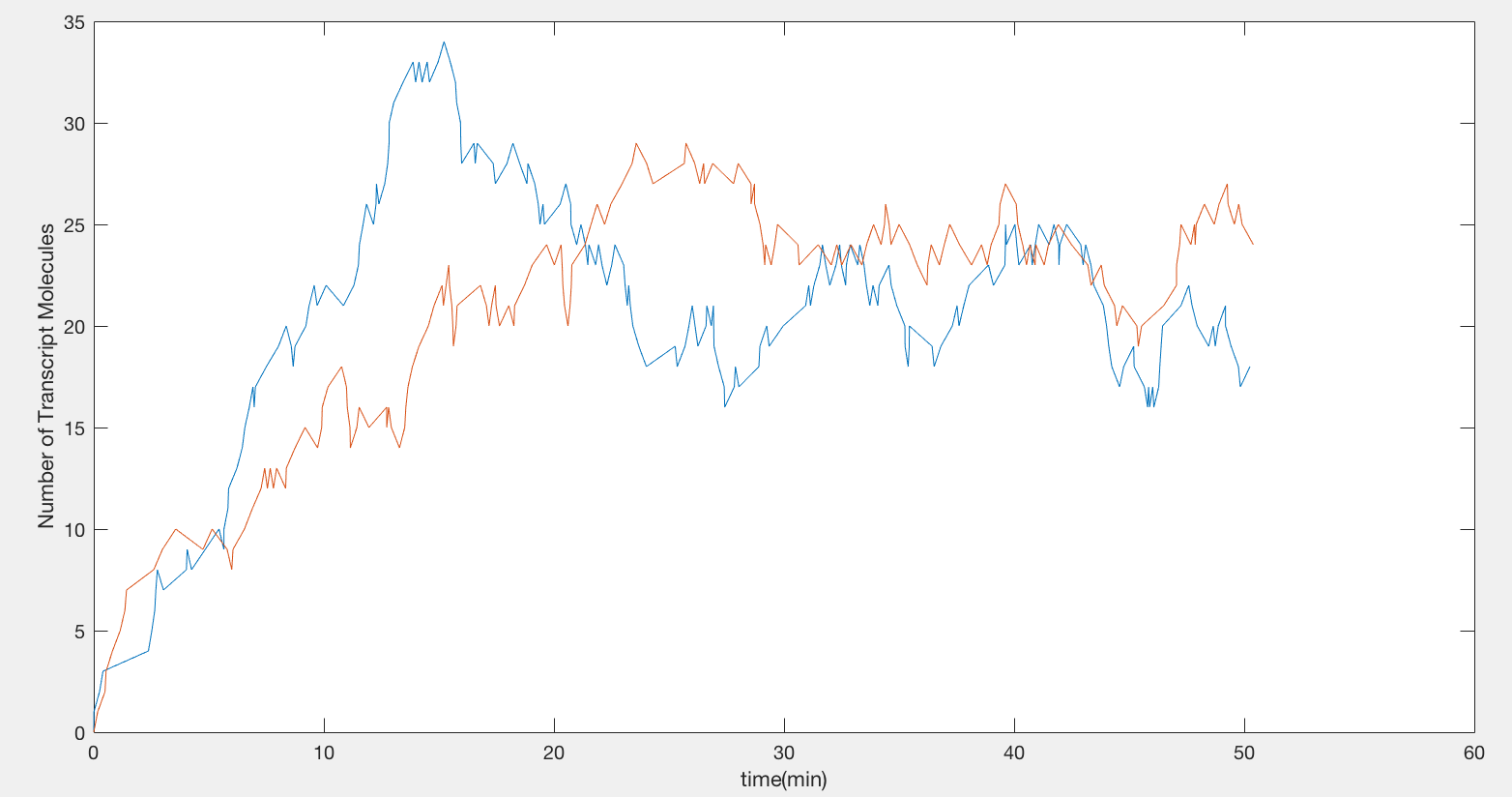
plot(t,N);

hold on

end

xlabel('time(min)');

ylabel('Number of Transcript Molecules');



**Run 200 realizations of the process. Compare these distributions with the analytical distribution function.**

**Case 1: t = 1/k**

clear

lamda = 2; %min^-1

k = 0.1; %min^-1

t\_end = 1/k;

t\_ini = 0;

%% **200 realizations and store the last position**

for i=1:200

n =0;

t\_new = 0;

while t\_new < t\_end

a = rand(1); % first generate a r.v

ti = -log(1-a)/(lamda+k\*n); % intervt it and get the nth time point

t\_new = t\_new + ti;

p\_trans = lamda/(lamda+k\*n); % Determine whether transcript or degradation happens

b = rand(1);

num\_generated = 2\*(b <= p\_trans)-1;

n = n + num\_generated;

end

last\_pos (i) = n;

end

histogram(last\_pos); % get the distribution function for the num of molecules

hold on % y is frequency, x is the number of particles

%% **Compare these Distributions with the Analytical distribution function.**

P(1) = 0;

Nt = lamda/k\*(1-exp(-k\*t\_end)); %Nt: expected number of molecules at time t

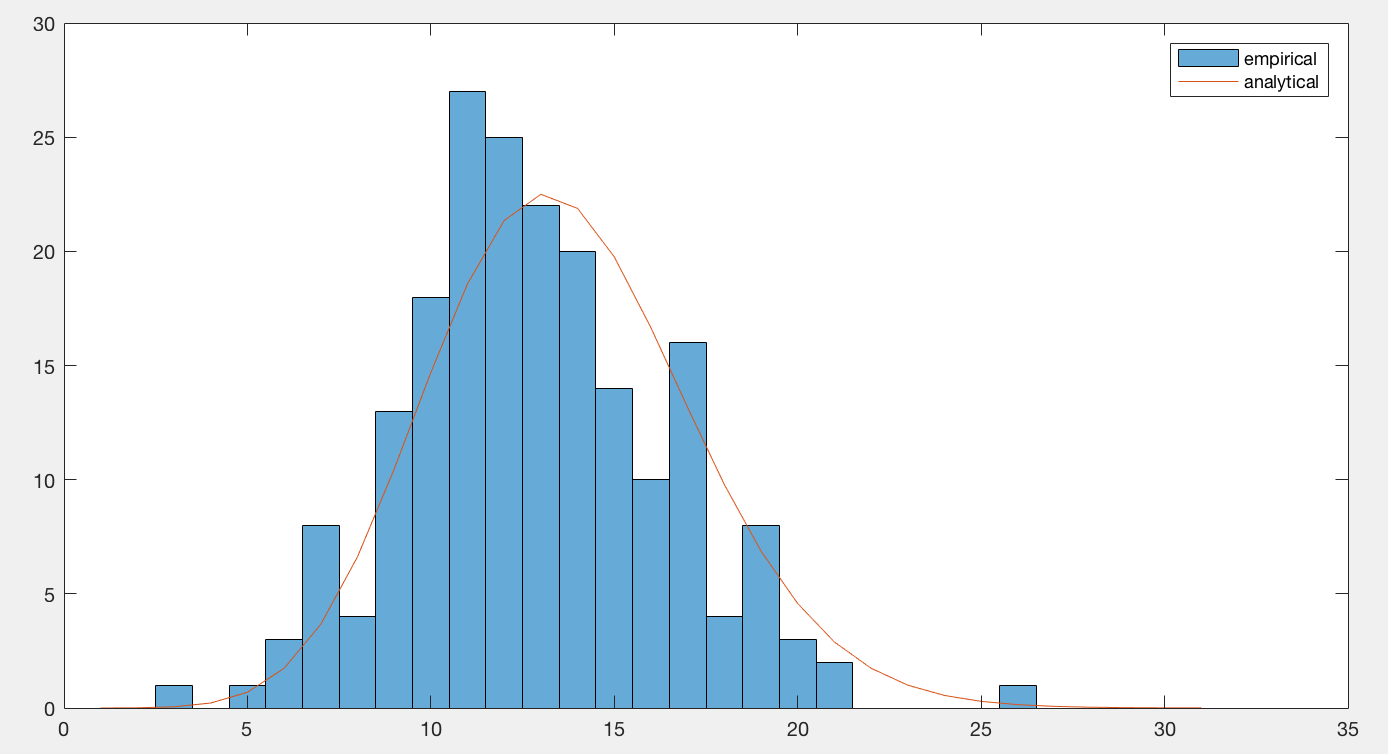
for n = 1:30

P(n+1) = Nt.^n/(factorial(n)).\*exp(-Nt);

end

plot(P\*200)

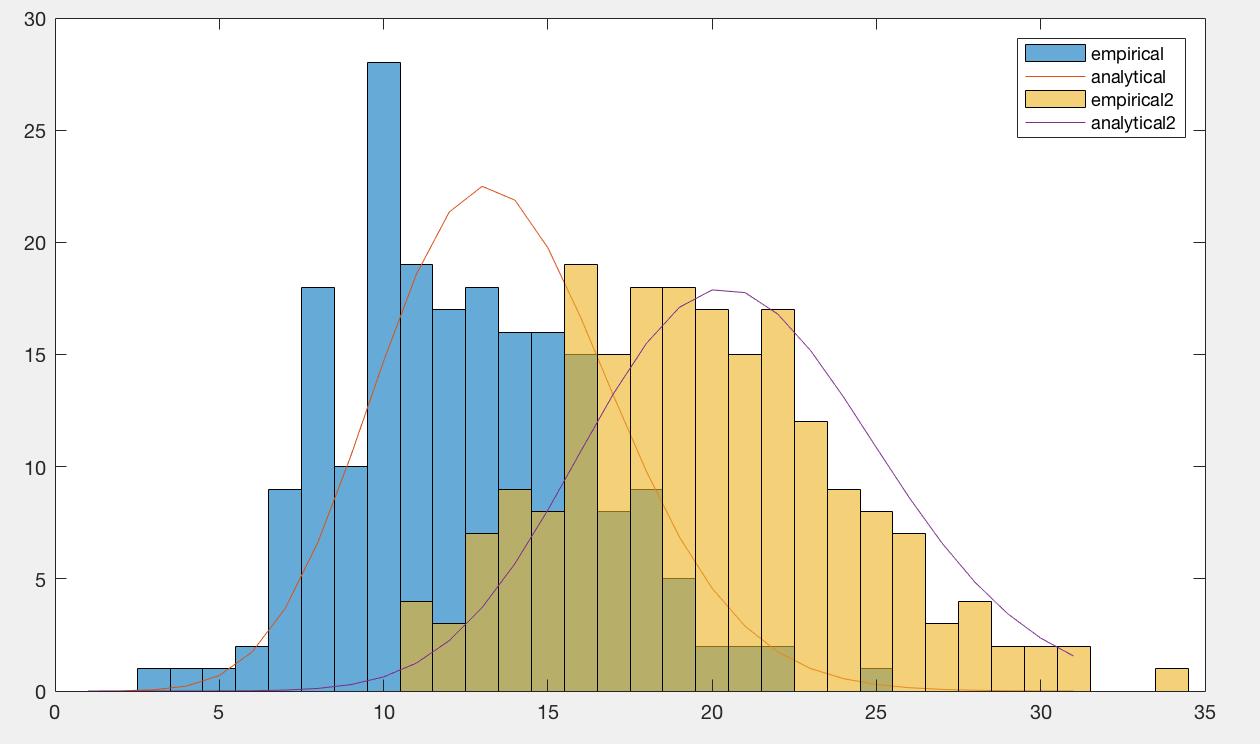
legend('empirical','analytical')

****

**Case 2: t – 5\*1/k**

Simply change the code to: t\_end = 5\*1/k;

All the other part remains the same

****

**Yellow and purple lines represent t = 5\*1/k.**