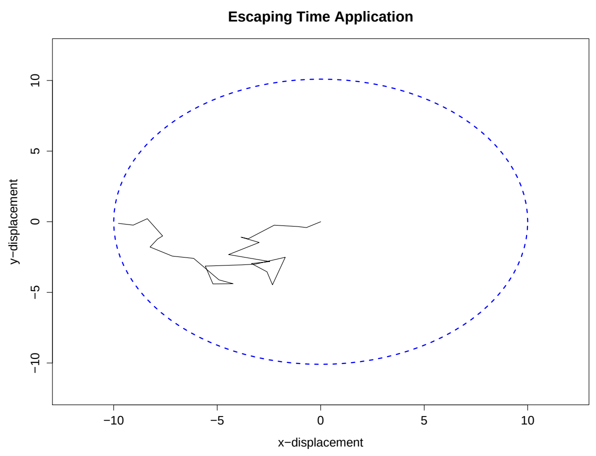
Escaping Time and Partical Collision Modelling and Simulation

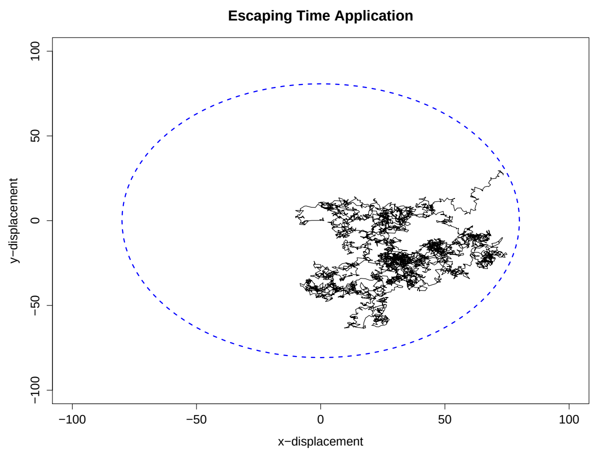
Zhijun Yang
Faculty Adivisor: David Aldous

Average Escaping Time Estimation

Considering the following situation, a microcell within a given domain can not move by itself, it can only moving follow the flows of the domain, suppose the region is two-dimensional and given motion of this movement follows from normal distribution, then we immediately see the movement of the cell follows a Brownian Motion trajectory as we discuss before. But the questions is when will it escape from our given domain, or it is same to say when will it reach the boundary point of our domain?

I will demonstrate first attempt to model this kind of problems by simuate a path follows stochastic process. We define our boundary as a circle plot the move and along with the boundary and let's take our trajectory to be a Brownian Motion for simplicity.





The number of hit (change of trajectory) before the particle reach the boundary is 27. If we assume that the number of hit is given by a certain distribution then this is actually represent the time the particle reach the boundary.

The above two plots are taken when we define our boundary circle have radius of 100 and 6400 respective. The resulting number of counts are 27 and 5050 respectively (The first plot is just the result of running our code in the previous page). Notice here we take our domain to be a circle in order to simplify our code and assumptions.

As you can check if we enlarge the radius of the circle, the number of count is going to become larger. The problem is how large? My intuition suggests that the number should be proportion to the area of the circle, however, the real situation is much more complicated to verify since the margin of error is too large due to the randomness of the movement, I can not give a precise conclusion without spending a lot of time dealng with that, since our primary interest is in simulation and application. The result will be leave for interested readers.

Mutiple Particles Collision

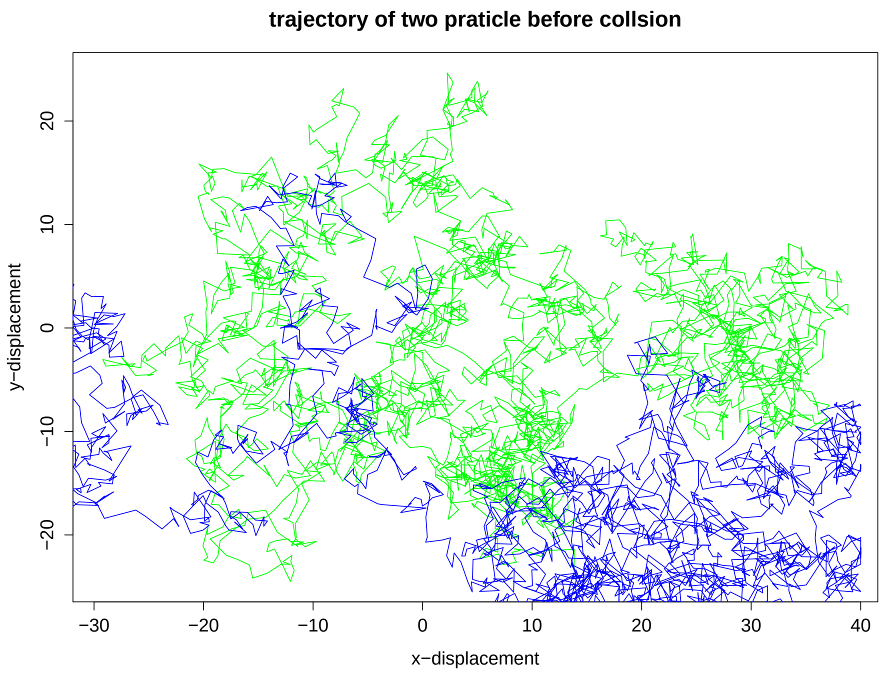
In this situation, let's put several different particles in a given domain, each follows a Brownian motion trajctory and test what is our expected time before any two of those particles hit each other. For simplicity let's assume that each of those particles has certian small volume with circluar shape in two-dimension. And when any two of those particle hit each other we stop the process and record the steps (time) from the beginning.

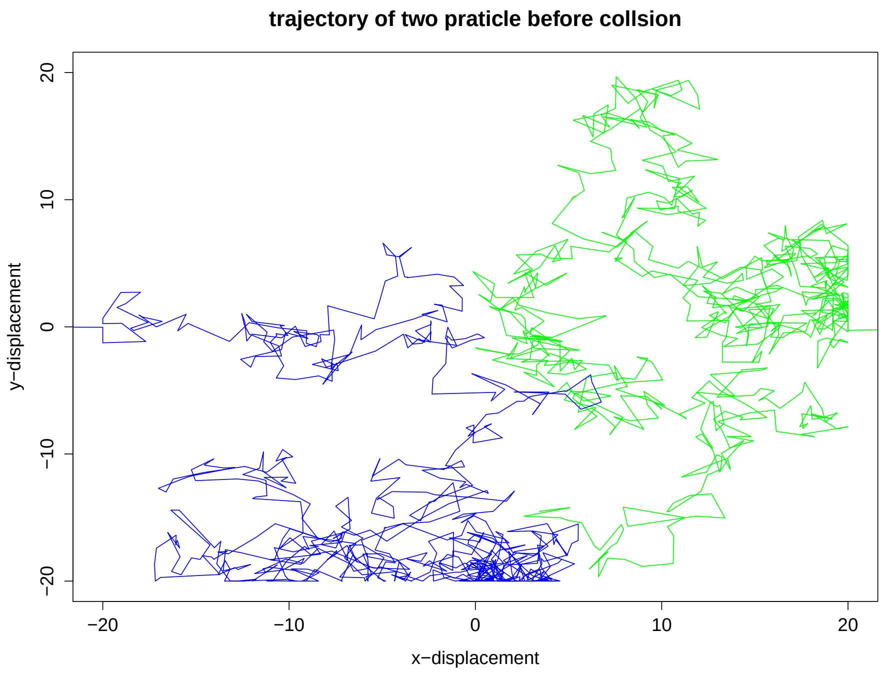
The trick here is that since we simplify our balls to have same volumes as a circular shape, we only need to record the distance between any two of our object to see whether any of those distance are less than two times the radius of our ball for each time.

It is significant easy to write the code in an object-oriented programming language, especailly considering the multiple particle collsion case. However, consider the context of our research is under statistical category, I will write the code in R, which is somehow complicate than it should be. For simplicity I will restrict my case to two particles within a square bounded domain. The code I write is given below (where we take the radius of each ball to be 2, and the boundary is between -40 to 40 for both x and y coordinates) :

As you can try for your own, the number of steps for each times varies a lot, sometimes it reaches max time of steps (N= 5000 in our case). Even we expected that two particle eventually collsion with each other, however there is not a theortically upper bound for time, that is this process would last forever. And due to the randomness of the trajectories, it is complicated to give an estimated expectation time (steps), and I will leave that for interested readers.

The resulting graph of two particle trajectories before they finally collsion is given by following, where the light color and dark color lines are respresenting two particle's trjectories respectively, for simplicity of the code and graph, we only produce this two particle collsion simulation using Brownian motion trajctory. If you are interested, you can modify the code above to produce a multiple particle collsion simulation in any given region.





The above two graphes represent when we taken boundary to be (-40,40) and (-20,20) for both x and y axis respectively, the first graph has step counts as 2707, the second graph has step counts as 684.

Appendix: Code 1.Escaping Time Coding*library(plotrix)*
N = 1000;
xdis = rnorm(N, 0 ,1); ydis = rnorm(N, 0, 1);
xdis1 = rep(0,N); ydis1 = rep (0,N);

```
count = 0;
for(i in 1:N){
if((xdis1[i]+xdis[i+1])^2 + (ydis1[i]+ydis[i+1])^2 ¡ 100){
count = count + 1;
xdis1[i+1] = xdis1[i]+xdis[i+1];
ydis1[i+1] = ydis1[i]+ydis[i+1];
}else{
count = count + 1;
xdis1[i+1] = xdis1[i]+xdis[i+1];
ydis1[i+1] = ydis1[i]+ydis[i+1];
break;
}}
xdis = rep(0,count); ydis = rep(0,count);
for(i in 1:count){
xdis[i] = xdis1[i];
ydis[i] = ydis1[i];
}

plot(xdis, ydis, type="l",main="Escaping Time Application", xlab="x-displacement",ylab="y-displacement",xlim=c(-
12,12),ylim=c(-12,12))
draw.circle(0,0,radius=10,border="blue",lty=2,lwd=2)

count;
```

this return 27, this means it requires 27 moves before our particle leaving our given domain.

2. Particle Collision Code

```
N = 5000;
radius = 2;
ub = 40; lb = -40;
axdis = rnorm(N, 0 ,1); aydis = rnorm(N, 0, 1);
bxdis = rnorm(N, 0, 1); bydis = rnorm(N, 0 ,1);
axdis1 = rep(0,N); aydis1 = rep(0,N);
bxdis1 = rep(0,N); bydis1 = rep(0,N);
axdis1[1] = 30; aydis[1] = 0;
bxdis1[1] = -30; bydis[1] = 0;
xdis1 = array(0,dim=c(2,N)); xdis = array(0,dim=c(2,N));
ydis1 = array(0,dim=c(2,N)); ydis = array(0,dim=c(2,N));
for(i in 1:N){
xdis1[1,i] = axdis1[i] ; xdis1[2,i] = bxdis1[i];
xdis[1,i] = axdis[i] ; xdis[2,i] = bxdis[i];
ydis1[1,i] = aydis1[i] ; ydis1[2,i] = bydis1[i];
ydis[1,i] = aydis[i] ; ydis[2,i] = bydis[i];}

    for(i in 1:N){
for (j in 1:2){
if(xdis1[j,i] + xdis[j,i+1] ¿ ub){
xdis1[j,i+1] = ub;
ydis1[j,i+1] = ydis1[j,i] + ydis[j,i+1]/2;
}else{
if(xdis1[j,i] + xdis[j,i+1] ¡ lb){
xdis1[j,i+1] = lb;
ydis1[j,i+1] = ydis1[j,i] + ydis[j,i+1]/2;
}else{
if( ydis1[j,i] + ydis[j,i+1] ¿ ub){
ydis1[j,i+1] = ub;
xdis1[j,i+1] = xdis1[j,i] + xdis[j,i+1]/2;
} else{if(ydis1[j,i] + ydis[j,i+1] ¡ lb){
ydis1[j,i+1] = lb;
xdis1[j,i+1] =xdis1[j,i] + xdis[j,i+1]/2;
}else{
xdis1[j,i+1] = xdis1[j,i] + xdis[j,i+1];
ydis1[j,i+1] = ydis1[j,i] + ydis[j,i+1];
}}}}}}
count = 0;
for(i in 1:N){
if((xdis1[1,i] − xdis1[2,i])^2 + (ydis1[1,i] − ydis1[2,i])^2 <= 2*radius){
break; }
count = count + 1;}
x1= rep(0,count); x2 = rep(0,count); y1=rep(0,count); y2 = rep(0,count);
for(i in 1:count){
x1[i]=xdis1[1,i]; x2[i]= xdis1[2,i];
y1[i]=ydis1[1,i]; y2[i]= ydis1[2,i];}
```

```
plot(x1, y1, type=”l”,main=”trajectory of two praticle before collsion”, xlab=”x-displacement”,ylab=”y-displacement”,col=”green”)
lines(x2, y2, col =”blue”)
count; this returns 2707 which gives number of steps before two particle fianlly hit each other in this domain
```

Reference

Brownian Motion, Peter Morter and Yuval Peres, Cambridge Press, 2010.

Brownian Motion and Stochastic Calculus, Ioannis Karatzas and Steven E. Shreve, Second Edition, Springer Verlag Press, 1991.

An introduction to Stochastic Modeling, Howard M.Taylor and Samuel Karlin, Third Edition, Academic Press, 1998.

Continous Martingales and Brownian Motion, Daniel Revuz and Marc Yor, Third Edition, Springer Press, 1999.

Explosion Effect Modelling and Simulation

Zhijun Yang
Faculty Adivisor: David Aldous

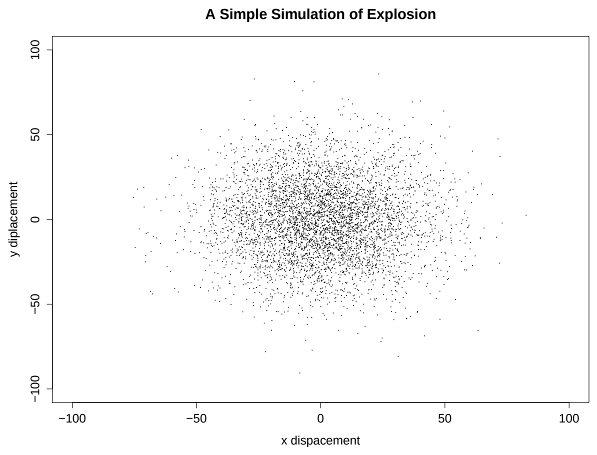
In a physical explosion situation, the particles "emiited" from a compact box into a larger space. Each individual particles are forced to move outward direction and possibly hitting each other in their path. In the real situation, the exact trajctory of each particles in explosion is too complicated to describle due to the complexity of explosion and vast amount of particles react with each other. So here, we give a probabilistic method to approximate the expected final position after some time interval of the explosion.

To simplify our model, let consider that all of our particles have initial position at the origin and all of them are massless. And let's consider our situation in a space without any other physical interaction. we expect that the distribution of displacement position of our each partivles roughly follows a normal distribution. Thus after a certian time steps N, our individual particles follows from a Brownian Motion Path.

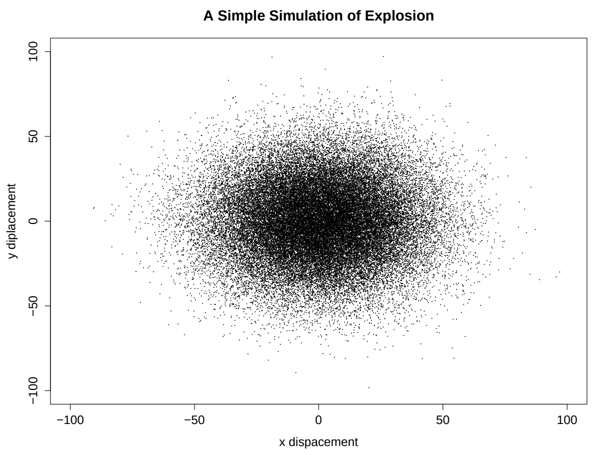
Note here is that those assumpition only hold true at a very short time interval after explosion, since in this time, the particles are very dense at a small ball around the orignal, so each particles react with each other constantly like the situation from our previous cases. But soon after a certian time, the distance of individual particles are large enough that they are not intereact with each other that often so the model are not accurate in taht case.

A Simple Example of Explosion Simulation

In this simulation, we assume our space in two dimension, and with all the assumption we developed in our discusssion above. and let's consider the case that the time steps are 500, and number of particles are relatively small, let's say 5000. And after running the code, the graph is produced as following..



And we do the simulation again with a much larger number of particles, let's taken the number to be 50000 instead of 5000, and the graph is given by the following.

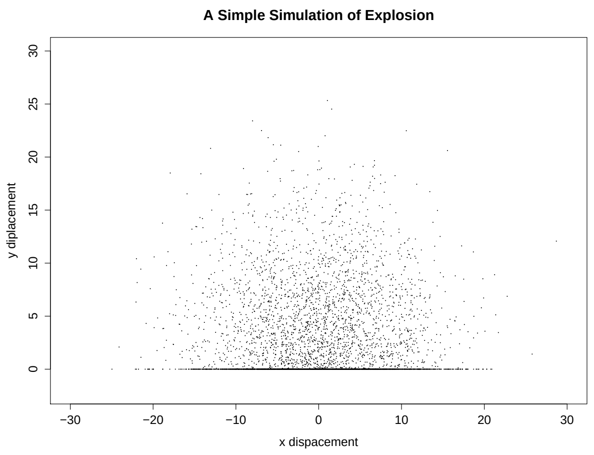


I found it pretty interesting that those graph not only looks like the explosion in the initial state but also looks like the distribution of possible position of an electron with an atom. However, due to the complexity I am not discuss about simulation of quantum particles in our simulation case, but leave this for interested readers.

Another Example of Explosion Simulation

Now let's consider the case where the explosion takes place in ground, that is any particles hitted the ground (y-position = 0) will be stayed at that level, only able to move in the x-direction, since the external force will still push it outward and sometimes inward. We will ignore the gravitation force at this time, since it largely increase the complexity of our code. And we take our time steps to be less N = 50 for quickly generating the simulation plot due to large number of calcaulation. Interested readers can modify the code to write a much effcient simulation.

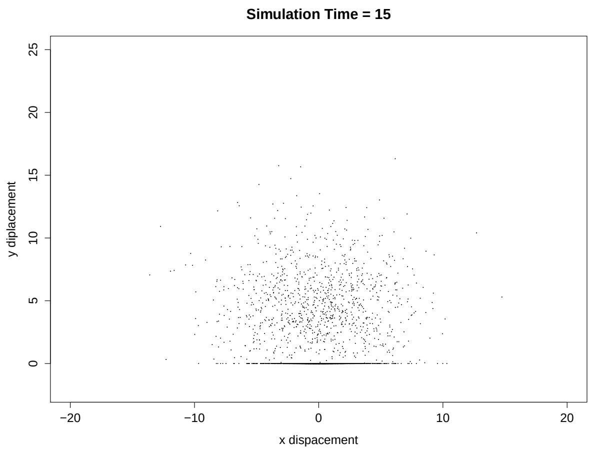
And let's plot the graph as before using plot function recursively, and we obtain the following And the black dots lies on the x-axis are are particles which reach the ground at this stage of the time.



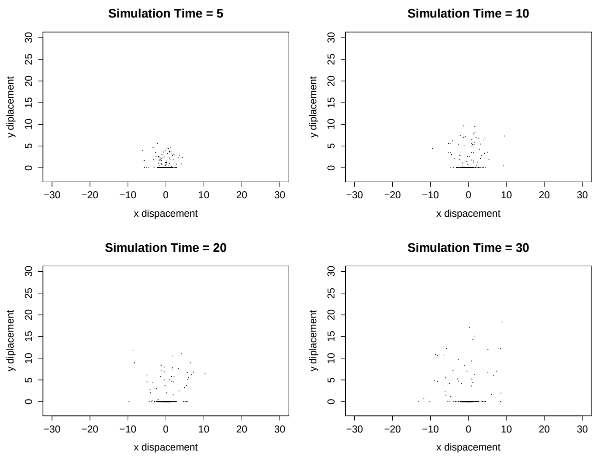
Explosion Simulation with Gravity

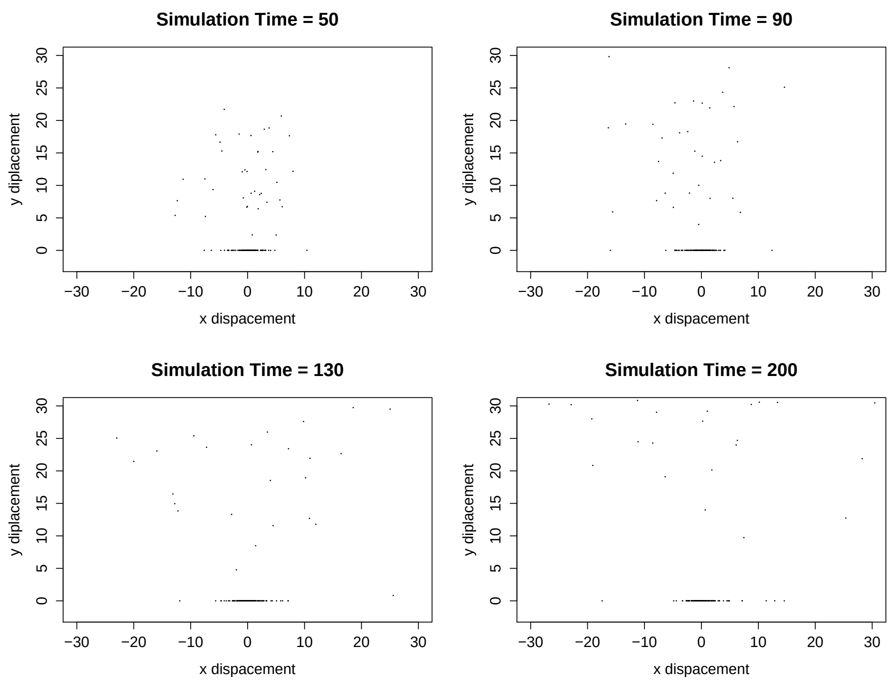
Now let's consider a model silimar as the examples before but with additional gravitation take place in our simulation. The gravity acts on each particle by changing its moving direction towards the ground. And once the particle hitted the ground, it no longer moving for the rest of the time. Again to simplfy this situation, let's introduce the gravitaion displacement, which acts on quantities of displacement of y axis of each particles constantly. We expected eventually most particles hit the ground..

After ploting the graph after a time interval N = 200, we see from the graph that some of the particles are stayed at the ground, and leaving some particles away from the origin. Such situation is quiet realistic indeed for this time stage.



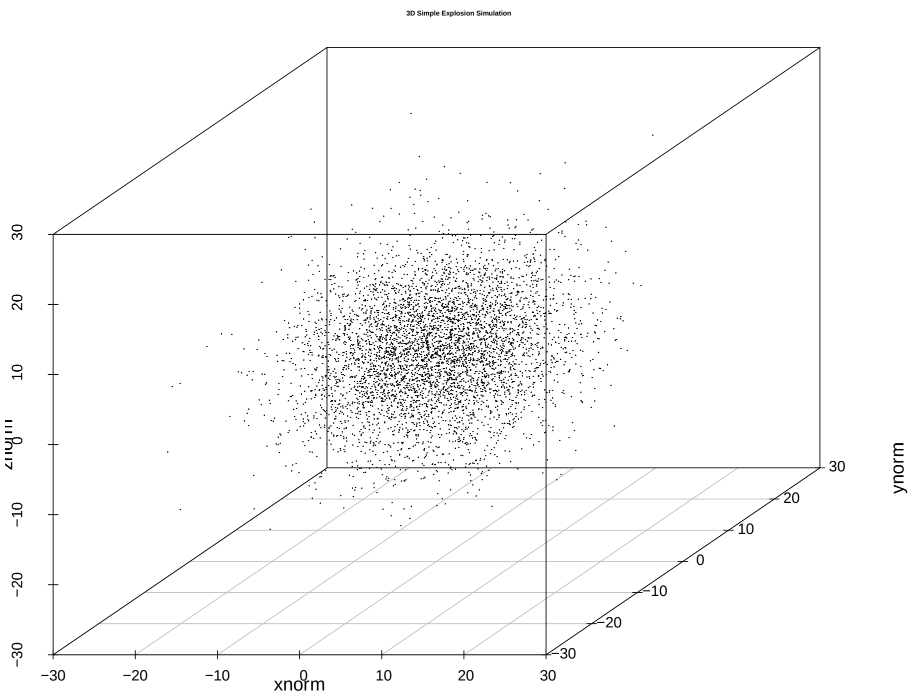
Now giving those seems simply but actually powerful code, we are able to examine for each time stage, the position of all exploded particles, and view this process a whole. The following pieces graph examines this explosion simulation at different time stages, where we take our number of particles P = 200 and time steps to be 5, 10, 20, 30, 50, 90, 130, 200 respectively. For quickly simulation of the graph, we take our number of particles to be 200 (reatively small number) for each graph.





A Simple Example of Explosion Simulation in Three Dimesnsion

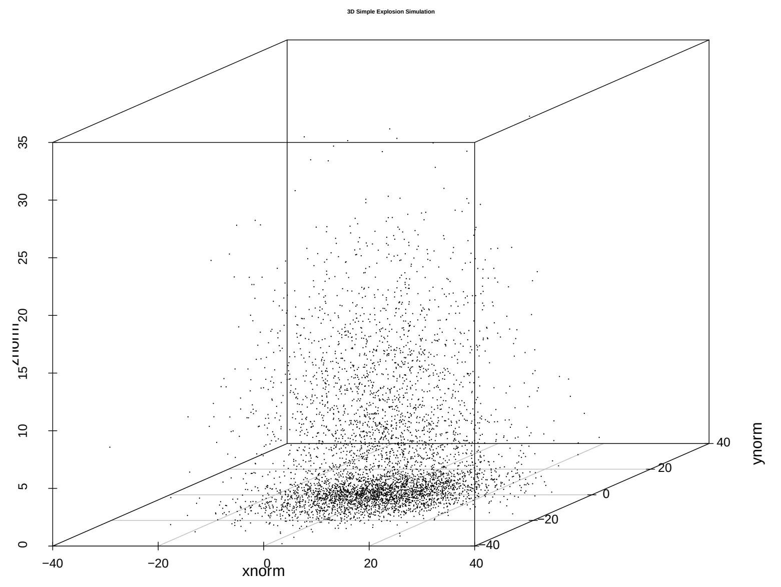
Like what we have done before, let's generalize the case into three dimension. We first consider the simple case corresponding to what we done in the first example of two dimensional case.

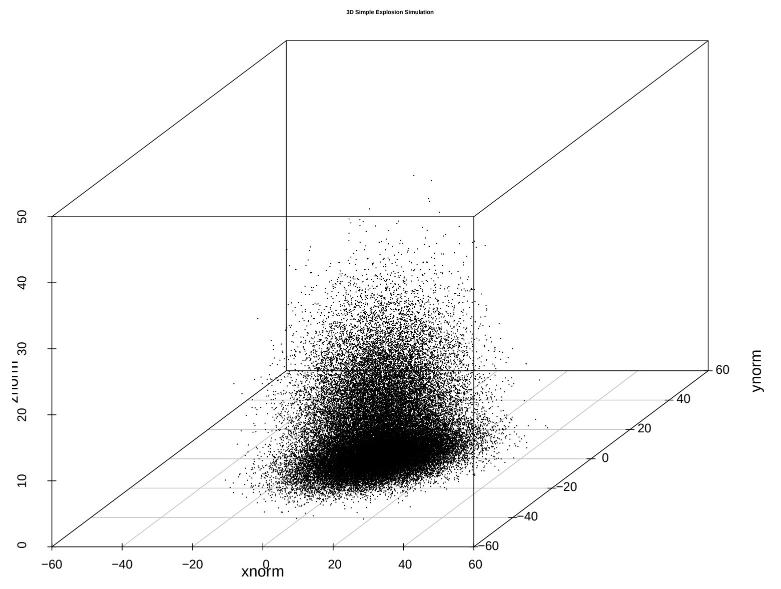


Another Example of Explosion Simulation in 3D

Now, we adjust the case to explosion on the ground as before, that is, after any particles the hited the ground, it remains there for the rest of the time.

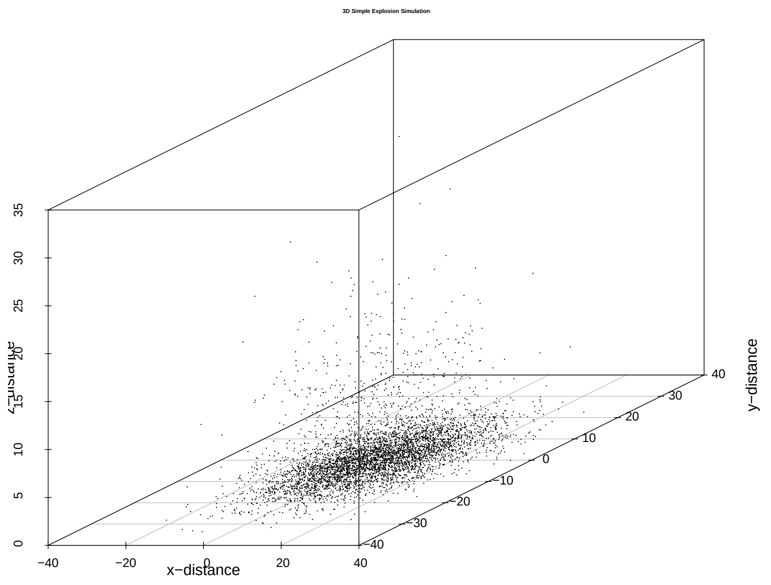
And the graph produced is given as following, we see it is a very realistic simulation of the explosion. Where I take the amount of particles to be 5000 and 50000 to produce the following two graphs.





Example of Explosion Simulation with Gravity in 3D

Now we continue our modification of simulation as before, where we take the gravitation into account. Again we introduce our gravitation displacement constant G = -9.8*0.01 into our code and modify the part it affects on.

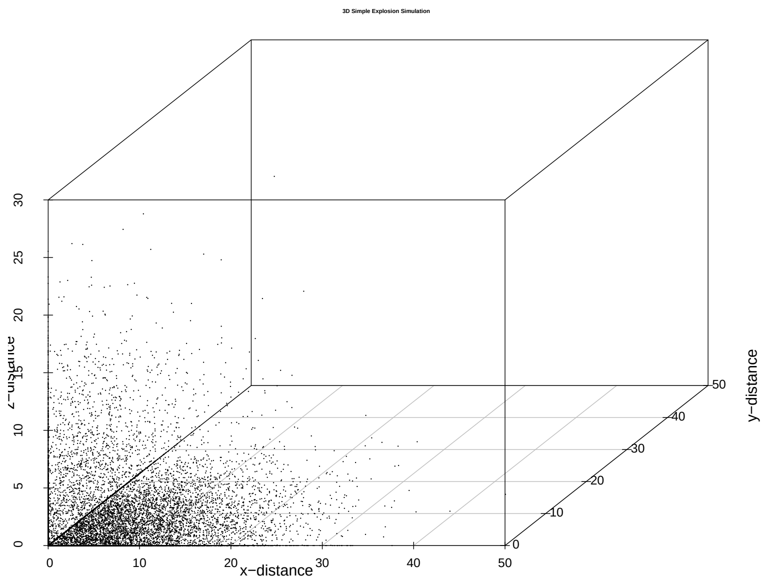


As we can see from the graph comparing to our correponding one (N = 5000) before, there are evidently less particles flowing over the air. This consistent with our intuition that graviaty does acts on those particles and eventually pull all of them on the ground. (Ignoring air resistency in all our case).

Example of Explosion Simulation in a Corner with Gravity in 3D

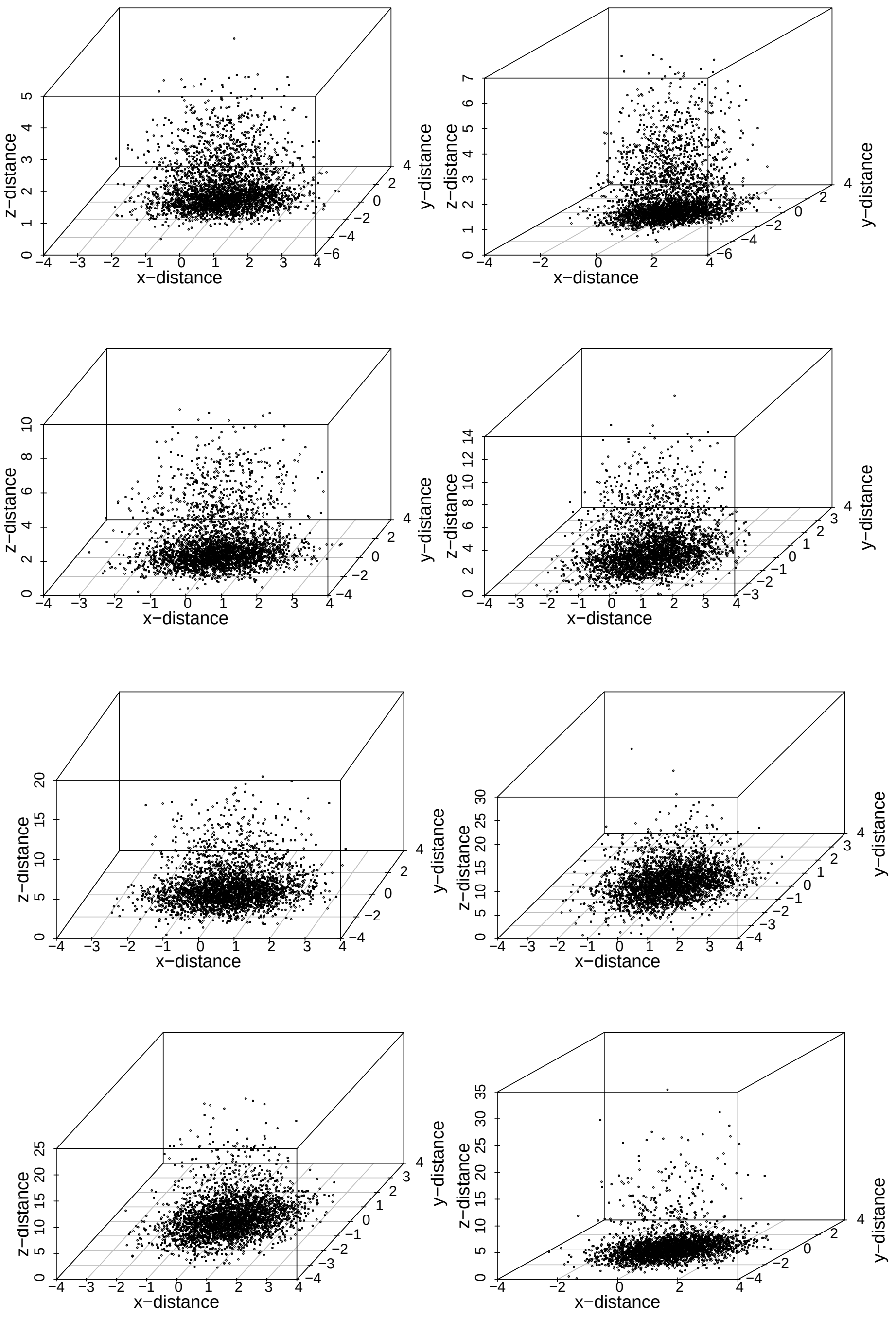
Note the code I produce is easy to modify to produce different effect of explosion situation, for example if I modify my code as following, we are able to produce an effect of explosion in a corner.

And the graph produced is given as following. It is a very nice simulation of explosion effect that takes place in a corner after some time interval. There are many other situation you can simulate by modify the code a little bit, but I am not to examine those situation for now.



Example of Explosion Simulation in Each Stage with Gravity in 3D

For our final part of this section, like in the two-dimensional case, I am going to examine the simulation effect in a sequence of time interval, that is for each time stage, we are going to plot the positions of all exploded particles, and view this process a whole. The following pieces graph examines this explosion simulation at different time stages, where we take our number of particles P = 3000 and time steps to be 2, 5, 10, 20, 30, 50, 90, 150 respectively. Together, these following graphs give us a general idea of how the explosion evolves in each time stage.



Appendix: Code

1. Simple Explosition Code

```
N = 500; NUMBER OF TIME STEPS
P = 5000; NUMBER OF PARTICLES

xrnorm = matrix(nrow = P, ncol = N);
yrnorm = matrix(nrow = P, ncol = N);
for(i in 1:P){
xrnorm[i,] = cumsum(rnorm(N,0,1));
yrnorm[i,] = cumsum(rnorm(N,0,1));
}

plot(xrnorm[1,N],yrnorm[1,N],xlim = c(-100,100), ylim = c(-100,100),cex=0.1)

for(i in 2:P){
points(xrnorm[i,N],yrnorm[i,N],cex=0.1)
}
```

2. Explostion Simulation on Ground

```
N = 50; NUMBER OF TIME STEPS
P = 5000; NUMBER OF PARTICLES

xrnorm = matrix(nrow = P, ncol = N);
yrnorm = matrix(nrow = P, ncol = N);
for(i in 1:P){
xrnorm[i,] = cumsum(rnorm(N,0,1));
yrnorm[i,] = cumsum(rnorm(N,0,1));
for(j in 1:N){
if(yrnorm[i,j] ¡= 0){
for(k in j:N){
yrnorm[i,j] = 0;
}}}}
```

3. Explosition Simulation on Gound 2

```
N = 15; NUMBER OF TIME STEPS
P = 5000; NUMBER OF PARTICLES
g = -9.8*0.01*1 GRIVATIONAL DISPACEMENT ON PARTICLES

xrnorm = matrix(nrow = P, ncol = N);
yrnorm = matrix(nrow = P, ncol = N);
for(i in 1:P){
xrnorm[i,] = cumsum(rnorm(N,0,1));
yrnorm[i,] = cumsum(rnorm(N,0,1) - g);
for(j in 1:N){
if(yrnorm[i,j] ¡= 0){
for(k in j:N){
xrnorm[i,k] = xrnorm[i,j];
yrnorm[i,k] = 0;
}}}}

plot(xrnorm[1,N],yrnorm[1,N],xlim = c(-20,20), ylim = c(-2,25),cex=0.1,xlab="x dispacement", ylab="y
diplacement", main="Simulation Time = 15")

for(i in 2:P){
points(xrnorm[i,N],yrnorm[i,N],cex=0.1)
}
```

4. Explostion Simulation on 3 Dimension

```
N = 50;
P = 5000;

xnorm = 1:P;
ynorm = 1:P;
znorm = 1:P;

for(i in 1:P){
xnorm[i] = (cumsum(rnorm(N,0,1)))[N];
ynorm[i] = (cumsum(rnorm(N,0,1)))[N];
znorm[i] = (cumsum(rnorm(N,0,1)))[N];
}

library(scatterplot3d)
par(cex = 0.3)
scatterplot3d(xnorm,ynorm,znorm,type=”p”,main=”3D Simple Explosion Simulation”)
```

5. Explostion Simulation on 3 Dimension 2

```
for(i in 1:P){
xnorm[i] = (cumsum(rnorm(N,0,1)))[N];
ynorm[i] = (cumsum(rnorm(N,0,1)))[N];
ztraj = cumsum(rnorm(N,0,1) + G) HERE WE INTRODUCE THE GRAVITATION CONSTANT G
for(j in 1:N){
if(ztraj[j] ¡= 0){
znorm[i] = 0;
{else{
znorm[i] = ztraj[N];
}}}
```

6. Explostion Simulation on 3 Dimension on Corner

```
for(i in 1:P){
xtraj = cumsum(rnorm(N,0,1))
for(j in 1:N){
if(xtraj[j] ¡= 0){
xnorm[i] = 0;
}else{
xnorm[i] = xtraj[N];
}}
ytraj = cumsum(rnorm(N,0,1))
for(j in 1:N){
if(ytraj[j] ¡= 0){
ynorm[i] = 0;
}else{
ynorm[i] = ytraj[N];
}}
ztraj = cumsum(rnorm(N,0,1) + G)
for(j in 1:N){
if(ztraj[j] ¡= 0){
znorm[i] = 0;
}else{
znorm[i] = ztraj[N];
}}}
```

Reference

Brownian Motion, Peter Morter and Yuval Peres, Cambridge Press, 2010.

Brownian Motion and Stochastic Calculus, Ioannis Karatzas and Steven E. Shreve, Second Edition, Springer Verlag Press, 1991.

An introduction to Stochastic Modeling, Howard M.Taylor and Samuel Karlin, Third Edition, Academic Press, 1998.

Continous Martingales and Brownian Motion, Daniel Revuz and Marc Yor, Third Edition, Springer Press, 1999.