Path Coalescence And Its Applications

Name:Aditya Challa
Civic Registration No(Sweden): 892210-P970
Advisor: Prof. Bernhard Mehlig

19 July 2011

Contents

1 Introduction **1**

2 Path Coalescence- System 1 **3**
2.1 Random Velocity Field 3
2.2 Phase transition and lyapunov exponent 4
2.3 The Stationary State and Formation of Caustics 6
2.4 Implementation of the path coalescence 8

3 Path Coalescence- System 2(Discrete Random walk) **9**

4 Equality between System 1 and System 2 **10**

5 Applications of Path Coalescence and Summary **13**

A Appendix 1 **15**

B Appendix 2 **17**

C Appendix 3 **18**
C.1 Path Coalescence System 1 18
C.2 Lyapunov Exponent For System 1 20
C.3 Mean Final Velocities System 1 23
C.4 Rate of Crossing Caustics 26
C.5 Path Coalescence System 2 29
C.6 Lyapunov Exponent System 2 31

1 Introduction

When Particles are suspended in a fluid, with turbulence and significant effects of inertia, it has been observed that the particles aggregate. This is termed as *Path Coalescence*. This aggregation is not always observed. There is an interesting phase transition illustrated by Figure 1 and Figure 2. The figures show the trajectories of the particles with uniform initial distribution under the influence of a random velocity field in one dimension. The x-axis represents time, and y-axis gives the position of the particle at a given time. In Figure 1 the initial uniform distribution remains uniform, but in Figure 2 the particles coalesce.The aim of this project is to understand and characterise this phase transition.

There are a lot of experiments performed to observe the fluctuations in particle densities by clustering in complex or turbulent flows. Clustering of particles to give a non-uniform final state is observed. There is a subtle difference between aggregation and clustering. Aggregation is a limit of clustering, where the particles stick together. There are only a few lab experiments on aggregation. But one of the physical situations where aggregation is observed is in the process of formation of raindrops. The behaviour of path coalescence is believed to play a major role in this process. Water droplets are initially formed due to condensation of the water vapour. However, if only condensation were to act, then it will take hours for the raindrops to increase their sizes by millions of times. Thus path coalescence should play an important role in the formation of raindrops. In Reference [6], Falkovich.G , Fouxon.A and Stepanov.M.G, derived an expression for the collision rate of the small droplets in such conditions. The collection rate (or coalescence rate, or rate of aggregation of particles) is proportional to the collision rate. This collision rate is also called *Rate of formation of caustics* which is discussed here. The authors conclude from the collision rates that air turbulence will enhance the formation of rain droplets through coalescence. Another physical situation where coalescence plays an important role is in the formation of planetary bodies. This application is discussed in Reference [2] and the references therein.Thus coalescence is an important part of the nature which needs to be understood in detail.

The effect of aggregation was characterised by Deutsch in Reference [5](1985). The equation of motion he considered was

$$m\ddot{x} + \gamma\dot{x} = f(x,t)$$

where,m stands for the mass of the particle, and x stands for the position. The dot on the top, as in $\dot{x}$, represents the time derivative of the quantity. $f(x,t)$ is the random velocity field as discussed in detail later in this report (See the section on *Random Velocity Field*). He observed the path coalescence effect and the phase transition between aggregating and non-aggregating phases. This is surprising because each particle is expected to show

diffusive type of dynamics in a random velocity field. Hence, we expect that the variance will increase with time and will result in a uniform final distribution of the particles. This seems to contradict the observation of particles clustering together. From the discussion in this project report it will be seen that these are just complementary results and the main part of the variance comes from the separation of two different clusters. Deutsch used the method proposed by Klyatskin and Tartarski (In Reference [13]) to obtain the equation for the probability distribution of the separation of particles. He then defined an *order parameter*, as the probability that the separation between two particles will go to zero in the limit of the time going to infinity. If the order parameter is 1, then almost surely the particles aggregate. If the order parameter is 0, then the particles almost surely do not aggregate. He obtained analytical expressions for the order parameter. There is a difference between the method followed by the author and the method adopted here. We use lyapunov exponents to characterise the separation of the particles. It can be easily seen that one implies the other, if we can define the probabilities in terms of lyapunov exponents. In the Reference [5] the random velocity field is simulated using discrete random numbers at specified points on the grid . He defines a 2 dimensional grid whose axes are (x, t), and maps each of the grid points to a random number. For the values in between, the force is calculated using linear interpolation. Another method to generate such random fields is described in this report. The author also points out a behaviour in the iterated maps of a function with random noise which can be interpreted as aggregation. The functional forms of the probabilities in both the cases turn out to be identical. This is a remarkable result.

My aim in this project is to observe, characterise and understand the system which exhibits path coalescence. The system should be ideally dealt in higher (two and three) dimensions. But as a starting point we deal only with one dimensional systems, which are easy to comprehend and visualise, but still have the key properties observed in higher dimensions. The main difference between the one dimensional and higher dimensional cases is that of compressibility. In one dimension, the only incompressible transformations are translations, which are trivial. Hence we must consider compressible fluids in one dimension. However, in higher dimensions there are non trivial incompressible transformations. In the incompressible fluids in higher dimensions, if the fluid compresses in one direction, it must expand in the other. In terms of lyapunov exponents it means that not all lyapunov exponents can be negative. In such systems clustering is observed but aggregation is not possible. This is because, if atleast one of the lyapunov exponents is positive then the distance between the particles can only increase. Thus aggregation can only occur when the fluid is compressible. The clustering in incompressible fluids is observed to give patterns, which are dynamic in nature. J.Bec,in Reference [8], discussed the fractal clustering of the inertial particles in random flows. He makes two important observations. The first one is that fractal clustering is seen even for small stokes number,and the lyapunov dimension (defined in Reference [8]) reduces quadratically with stokes number. The second observation is that there is a critical value of the stokes number, below which the clustering is observed.

In the discussion in this report we concentrate on characterising the the phase transition between aggregating and non-aggregating phases. As mentioned earlier, clustering differs from aggregation in a subtle way. Although clustering is different from aggregation the physical mechanisms resulting in both kinds of behaviour can be assumed to be the same. The reason for clustering appears to be the vorticity in turbulent flows, which will push the particles outward giving a centrifugal force to the particles, which will cause density inhomogeneities in the particle density. This is sometimes referred to as *Preferential Concentration* phenomenon. M.R.Maxey predicted that particles will have low density in regions of high vorticity for this reason. This relation is also observed in the numerical simulations (See Reference [10] and Reference [11]). One of the predictions of this theory is that, the coalescence is maximum when the stokes number is approximately unity (See Reference [12]). Numerical simulations confirm this prediction. The authors in Reference [9] give numerical evidence confirming the above results.

There is also an alternate explanation. The turbulence will randomly compress and expand the volume elements in the compressible fluids. This will once again results in the inhomogeneities of particle densities. This motivates us to use the concept of lyapunov exponents (defined later) to characterise the phase transition. Note that this theory would not explain the clustering in random incompressible flows. The method we follow here is proposed by B.Mehlig and M.Wilkinson in Reference [14] , where the authors use a similar model to Deutsch's model. The authors also extended this model in two dimensions (Reference [3]) and three dimensions (Reference [4]) and derived the expressions for the lyapunov exponents under some approximations. The results are verified with some numerical experiments. See Reference [3] and Reference [4].

In what follows, we discuss path coalescence transition in one-dimension in detail. The system 1 we consider here is the motion of suspended particles in random velocity fields. We also describe path coalescence in a simplistic model(system 2), one dimensional random walk, and show that in the limit of infinite inertia, system 1 reduces to system 2 , atleast qualitatively.

2 Path Coalescence- System 1

We start with describing the system and give the equations of motion where the above mentioned behaviour is observed. The system consists of spherical particles (can be water droplets) suspended in turbulent fluid. An example of such a system is water droplets suspended in turbulent atmosphere, or any aerosols. The turbulence is approximated by a random velocity field.Isotropy and homogeneity of the velocity field is also assumed. The particles will then experience a drag force (friction) which is given by the stokes law. The equation (1) and equation (2) give the trajectories of a single particle. But we are interested in a system of particles, where we have particle-particle interactions. We assume, for the sake of simplicity, that no such interactions exist, or the interactions can be ignored. The equations of motion are

$$\dot{x(t)} = \frac{p(t)}{m} \quad (1)$$

$$\dot{p(t)} = f(x,t) - \gamma p(t) \quad (2)$$

In the above equations, $x(t)$ stands for the position of the particle at time t, $p(t)$ gives the momentum at time t, $f(x,t)$ is the random velocity field at $x(t)$ and at time t, γ gives the viscous damping. The dot on top of a quantity indicates the time derivative of the quantity. Hence $\dot{x(t)}$ indicates velocity, and $\dot{p(t)}$ indicates force. Figure 1 and Figure 2 show the 2 states. Figure 2 gives the parameter values where we can observe coalescence and Figure 1 gives the parameter values where we do not observe coalescence.

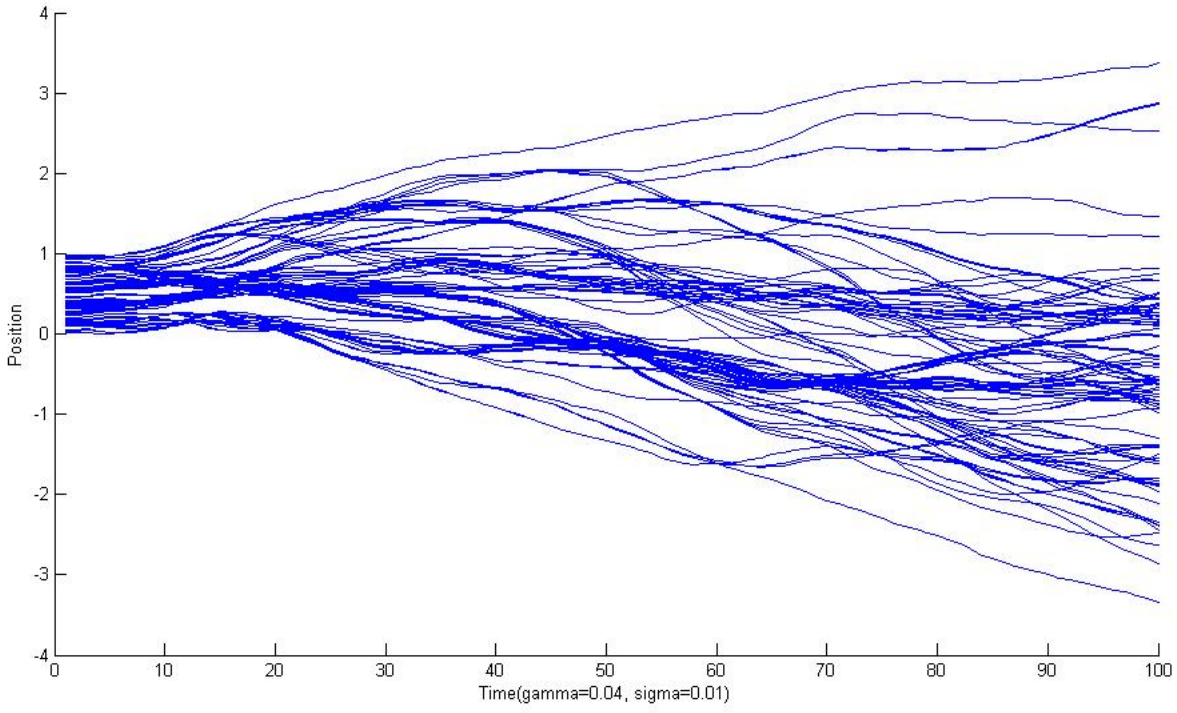

Figure 1: The above plot shows the trajectories of the particles in 1-dimension with time. The x-axis indicates time, and y-axis indicates the position at a given time. The parameter values are ξ=0.05, τ=1, σ=0.01, γ=0.04.

2.1 Random Velocity Field

Note that we approximated the turbulence with the random velocity field. This can be done because few of the statistical properties of the random field match with the properties of turbulence (see Reference [7]). However this does NOT mean that they are the same.To understand the motion of the particles, we need to analyse this random field thoroughly. The most important properties of this random field are:-
1)$< f(x,t) >= 0$, i.e the mean of the random velocity at a position at a given time is 0. This means that there

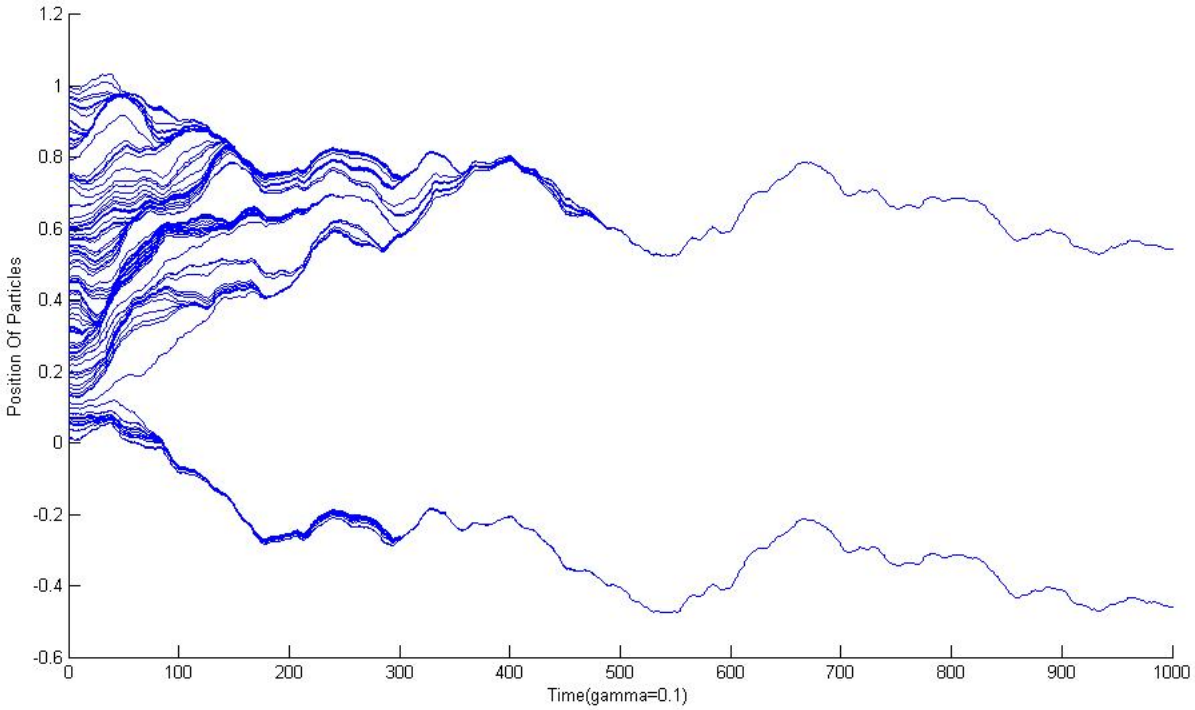

Figure 2: The above plot shows the trajectories of the particles in 1-dimension with time. The x-axis indicates time, and y-axis indicates the position at a given time. The parameter values are ξ=0.05, τ=1, σ=0.001, γ=0.1.

is no net drift of the distribution of the particles.
2)Given variance (σ), correlation length (ξ) and time correlation (τ) the correlation is given by

$$< f(x_1,t_1)f(x_2,t_2)^* >= \sigma^2 e^{-\frac{(x_1-x_2)^2}{2\xi^2}} e^{-\frac{|t_1-t_2|}{\tau}}$$

The variance (σ) gives the strength of the velocity field. We discuss only the case where time correlation is considered to be very small ($\tau \to 0$), that is, the velocity fluctuates very rapidly. We also define the dimensionless strength to be

$$\chi = \frac{\sigma\tau^2}{\xi m}$$

Although we defined the random velocity in one dimension, this can easily be extended to higher dimensions. (See Reference [7]). In the simulation of such a random field we need a distribution with peak at 0 and the variance σ^2. We take the distribution to be normal.See the section on simulation for details.
In Reference [7] (chapter 3) the author discusses and mentions several similarities and differences between the random velocity flow and turbulent flow. One of the most important difference is known as 'intermittent effect'. It is observed that when the velocity component in a turbulent flow has occasional peaks of high frequencies, which will result in a velocity distribution wider than the gaussian distribution we assumed. It is also observed that both random velocity field and turbulent flow exhibits a fractal distribution of the particles suspended under some conditions. This is one of the similarities between the two flows.

2.2 Phase transition and lyapunov exponent

As mentioned earlier a phase transition is observed. The phase transition occurs when the coalescing stops, i.e. when the probability that the nearby particles coalesce becomes very small. *Lyapunov Exponent* characterises this property of the system. The lyapunov exponent is defined as(See Reference [1])

$$\Lambda = \lim_{t\to\infty,\delta x(0)\to 0} \frac{1}{t} < ln \left| \frac{\delta x(t)}{\delta x(0)} \right| >$$

Contrast this to the order-parameter Deutsch used to describe phase transition. The lyapunov exponent compares the final distance between the two particles with the initial distance, and gives a number to characterise this change in distances. Thus lyapunov exponent is continuous. The order parameter is either 0 or 1. It just

gives whether the phase is coalescing or non coalescing. Since lyapunov exponent is continuous, it helps us to point out the transition exactly. Thus the phase transition happens when the lyapunov exponent vanishes. In the following we calculate and analytical expression (not a closed form) for the lyapunov exponent and compare it with the simulation results obtained (see Figure 3) .

Calculation of Lyapunov exponent

We first linearise the equation (1) and equation (2). We get

$$\dot{\delta x(t)} = \frac{\delta p(t)}{m} \tag{3}$$

$$\dot{\delta p(t)} = \Delta f(x,t) - \gamma \delta p(t) \tag{4}$$

For small $\delta x(t)$ (small compared to the correlation length of the field), we can approximate $\Delta f(x,t) = \frac{\partial f(x,t)}{\partial x}\delta x(t)$. Then we have that

$$\dot{\delta p(t)} = \frac{\partial f(x,t)}{\partial x}\delta x(t) - \gamma \delta p(t)$$

If we take $X = \frac{\delta p(t)}{\delta x(t)}$, we have that

$$\dot{\delta x(t)} = \frac{1}{m}\delta p(t) = \frac{X}{m}\delta x(t)$$

Note that the dot on the top denotes time derivative of the quantities. Then we have that

$$\begin{aligned}
\dot{X} &= \frac{\dot{\delta p(t)}\delta x(t) - \dot{\delta x(t)}\delta p(t)}{\delta x(t)^2} \\
&= \frac{\frac{\partial f(x,t)}{\partial x}\delta x(t)^2 - \gamma \delta p(t)\delta x(t)}{\delta x(t)^2} - \frac{\frac{\delta p(t)}{m}\delta p(t)}{\delta x(t)^2} \\
&= \frac{\partial f(x,t)}{\partial x} - \gamma X - \frac{X^2}{m}
\end{aligned}$$

So the final equations which we work with are

$$\dot{\delta x(t)} = \frac{1}{m}\delta p(t) = \frac{X}{m}\delta x(t) \tag{5}$$

$$\dot{X} = \frac{\partial f(x,t)}{\partial x} - \gamma X - \frac{X^2}{m} \tag{6}$$

To calculate the lyapunov exponent, we need to consider the stationary state. It is found that the stationary state here is characterised by the stationary distribution of X. Now is X does not change with time, the solution to equation (6) is like

$$\delta x(t) = e^{\frac{X}{m}t}\delta x(0)$$

So the lyapunov exponent is given by

$$\begin{aligned}
\Lambda &= \lim_{t\to\infty,\delta x(0)\to 0} \frac{1}{t} < ln \left| \frac{\delta x(t)}{\delta x(0)} \right| > \\
&= \lim_{t\to\infty,\delta x(0)\to 0} \frac{1}{t} < ln \left| e^{\frac{X}{m}t} \right| > \\
&= \lim_{t\to\infty,\delta x(0)\to 0} \frac{1}{t} < \frac{X}{m} > \\
&= \frac{< X >}{m}
\end{aligned}$$

Now we have to find the stationary distribution of X. Suppose $P(X,t)$ is the distribution of X, we have that $P(X,t)$ follows the Fokker plank equation. This can be seen by noting that the equation (6) describes a stochastic differential equation, and the process has markov property.

$$\frac{\partial P(X,t)}{\partial t} = -\frac{\partial}{\partial X}(v(X)P(X,t)) + \frac{\partial^2}{\partial X^2}(D(X)P(X,t)) \tag{7}$$

From the above equations it can be seen that

$$v(X) = -\gamma X - \frac{X^2}{m} \qquad D(X) = -\frac{1}{2}\frac{\partial^2}{\partial x^2}\int_{-\infty}^{\infty} \sigma^2 e^{-\frac{(x)^2}{2\xi^2}} e^{-\frac{|t|}{\tau}} dt\Big|_{x=0} = \frac{\sigma^2\tau}{2\xi^2}$$

The above quantities are calculated using

$$v(X) = \lim_{\Delta t\to 0}\frac{<\Delta X>}{\Delta t} \qquad D(X) = \lim_{\Delta t\to 0}\frac{<(\Delta X)^2>}{\Delta t}$$

If we have that X has a stationary distribution, then $\frac{\partial P(X,t)}{\partial t} = 0$. If we call $J = v(X)P(X,t) - D\frac{\partial P(X,t)}{\partial X}$, then we have that

$$\frac{\partial J}{\partial X} = 0 \Longrightarrow \frac{\partial P(X,t)}{\partial X} = \frac{v(X)P(X,t)}{D} + J(Constant)$$

Observe that $D(X)$ is independent of X. We can solve the above differential equation (see Appendix 2). If we transform the variables

$$P(X) = \alpha\varrho(Z,\Omega) \qquad Z = \frac{X}{(mD)^{\frac{1}{3}}} \qquad \Omega = \frac{\gamma m^{\frac{2}{3}}}{D^{\frac{1}{3}}}$$

Then the solution is

$$\varrho(Z,\Omega) = e^{-\Phi(Z,\Omega)}\int_{-\infty}^{z} e^{\Phi(Y,\Omega)}dY \tag{8}$$

where

$$\Phi(Z,\Omega) = -\frac{1}{D}\int v(X)dX = \frac{Z^3}{3} + \frac{\Omega Z^2}{2}$$

Note that we changed the variables after integration to get the final result. α is a suitable normalizing parameter. Thus

$$\int P(X)dX = 1 \Longrightarrow \alpha = \frac{1}{\int \varrho(Z,\Omega)(mD)^{\frac{1}{3}}dZ}$$

The result is the same as in Reference [14]. Hence the lyapunov Exponent is

$$\begin{aligned}\Lambda = \frac{<X>}{m} &= \frac{1}{m}\int_{-\infty}^{\infty} XP(X)dX \\ &= \frac{1}{m}\int_{-\infty}^{\infty} Z\alpha\varrho(Z,\Omega)(mD)^{\frac{2}{3}}dX \\ &= \left(\frac{D}{m^2}\right)^{\frac{1}{3}}\frac{\int_{-\infty}^{\infty} Z\varrho(Z,\Omega)}{\int_{-\infty}^{\infty}\varrho(Z,\Omega)}\end{aligned}$$

So the Lyapunov exponent is

$$\Lambda = \left(\frac{D}{m^2}\right)^{\frac{1}{3}}\frac{\int_{-\infty}^{\infty} Z\varrho(Z,\Omega)}{\int_{-\infty}^{\infty}\varrho(Z,\Omega)} \tag{9}$$

Figure 3 shows the plot of the empirical and theoretical values of the lyapunov exponent. This confirms the result we just obtained theoretically. Now the phase transition occurs when the lyapunov exponent is 0. The value of Ω at which the lyapunov exponent vanishes is found to be *0.82*. . . .

2.3 The Stationary State and Formation of Caustics

In the analysis above we assumed that the quantity X will have a stationary distribution. From equation (6), we see that if X is large compared to the noise, then the equation (6) will reduce to

$$\dot{X} = -\gamma X - \frac{X^2}{m}$$

The solution to the above equation looks like

$$X = \frac{\gamma}{e^{\gamma t + c} - \frac{1}{m}}$$

Where c is a arbitrary constant. This tells us that X will then go to $-\infty$ in finite time, and this gives rise to caustics (discussed later). So at the stationary state, we will have that $X \to -\infty$ with a flux J as is seen from

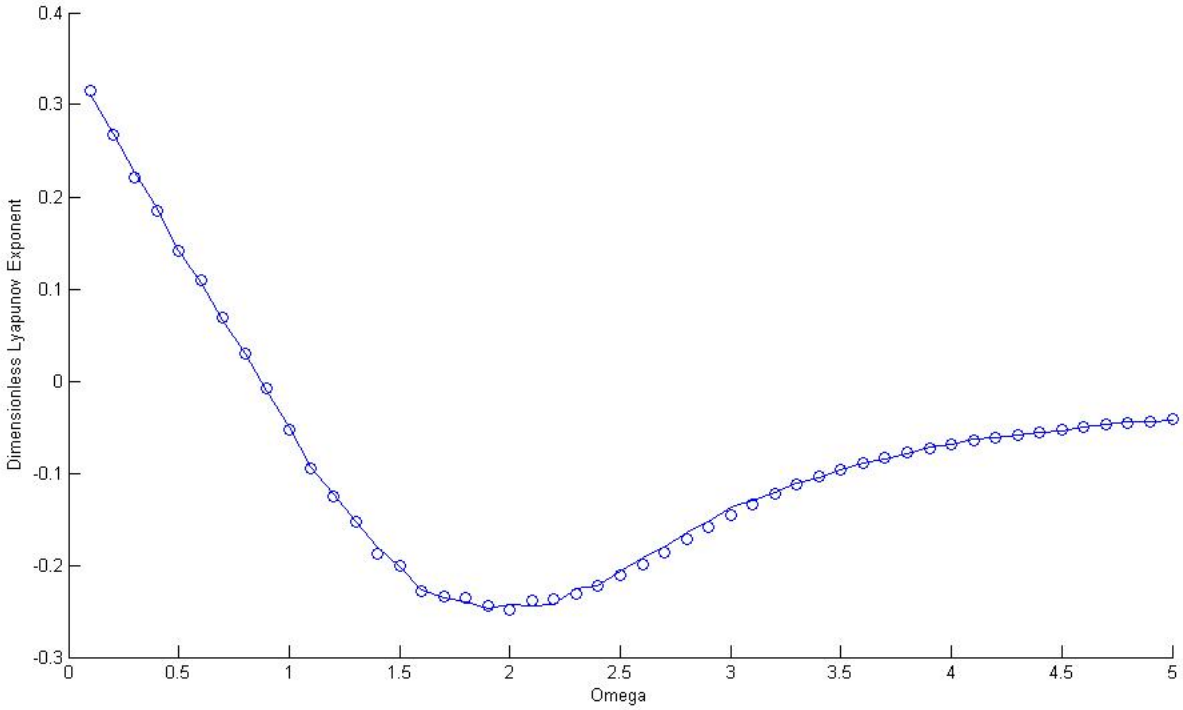

Figure 3: The theoretical and empirical plot of the dimensionless lyapunov exponent($\Lambda \left(\frac{D}{m^2}\right)^{\frac{-1}{3}}$). The dots indicate the empirical value, and the line gives the theoretical value.

the equations above.

In Figure 4 we plotted the mean of the final relative velocity in the stationary state versus the stokes parameter Ω. We see that it follows the same trend as the flux as we shall see in the next section. This is expected because as Ω increases, the viscous coefficient increases and the particles tend to change their velocities slowly compared to the change when Ω is smaller. Thus the relative velocity goes to zero as the inertial effects increase to infinity.

In the above discussion of the stationary state we stated that there will be a flux and X goes to $-\infty$. This means that the relative distance between 2 particles will go to 0. Once they separate again, X will be at $+\infty$. This is referred to as caustics. Thus,in one-dimension caustics appear when 2 particles cross each other. Caustics are important because it is a first step in the formation of aggregates. Thus the rate of formation of caustics can give us insight into the rate of aggregation (coalescing) of particles.

$$Rate\ Of\ Formation\ Of\ Caustics\ =\ J\ =\left(\frac{D}{m^2}\right)^{\frac{1}{3}}\frac{1}{\int_{-\infty}^{\infty}\varrho(Z,\Omega)} \tag{10}$$

The above expression for the flux can be easily verified by using one of the previously obtained equation

$$J \quad = \quad v(X)P(X,t) - D(X)\frac{\partial P}{\partial x}.$$

and substituting the values of

$$v(X) = -\gamma X - \frac{X^2}{m} \qquad and \qquad D(X) = \frac{\sigma^2\tau}{2\xi^2}.$$

in the equation. Note that once again we need to change the variables accordingly, as we did before, to get the final result.

In the phase diagram formation of caustics mean that the phase plot ($\dot{x}$ Vs x) is multivalued, that is 2 particles can be at the same position but with different velocities. Figure 5 gives a comparison of the empirical and theoretical results of the flux. Observe that as Ω increases, the rate goes down to zero rapidly.

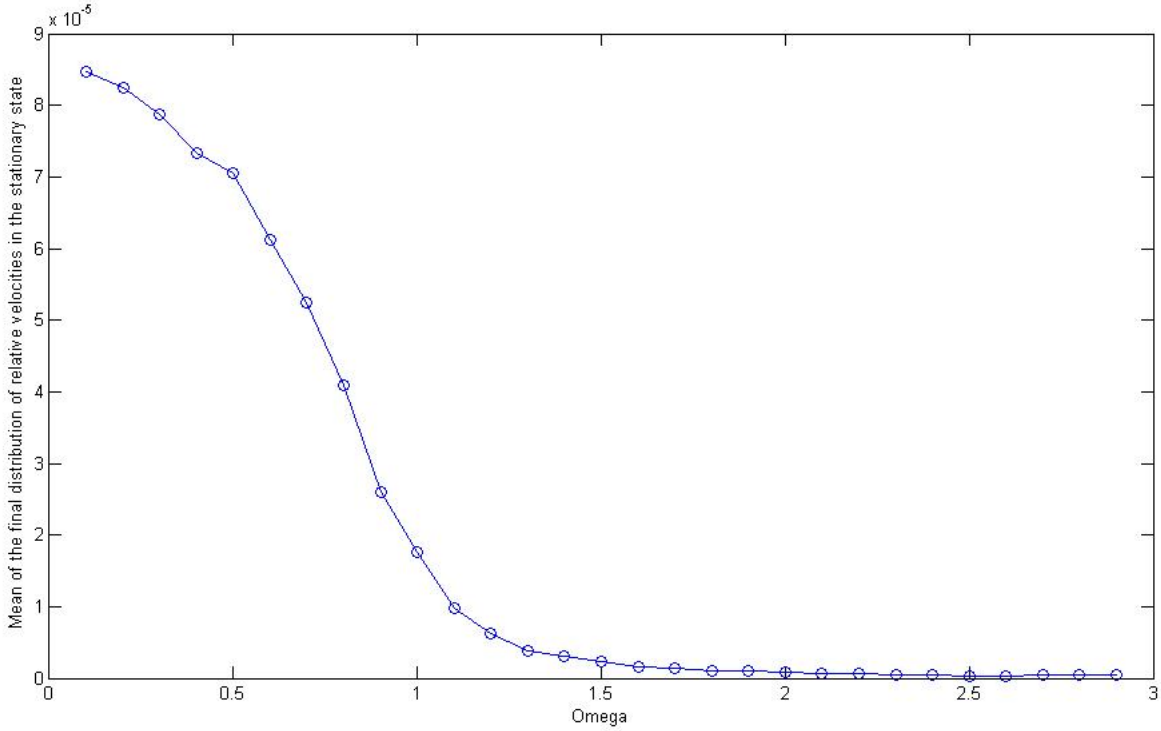

Figure 4: Mean of the final (relative)velocity in the stationary state Vs Ω

2.4 Implementation of the path coalescence

We now discuss the simulation of the above system, and procedures used to get the numerical values given above.Only the most important aspects are discussed here.For further details and the source code see Appendix 3.To simulate the equation (1) and equation (2) we need to first discretize the equations so as to enable for discrete computations. We discretize them as

$$\begin{aligned} x(t+\delta t) &= x(t) + \delta t \frac{p(t)}{m} \\ p(t+\delta t) = &= p(t) + \delta t (\int_t^{t+\delta t} f(x,t) - \gamma p(t)) \end{aligned}$$

We use $\int_t^{t+\delta t} f(x,t)$ instead of the usual $f(x,t)\delta t$ because of the assumption that the correlation time tends to 0, and hence the velocity field fluctuates rapidly. If we write

$$F(x) = \int_t^{t+\delta t} f(x,t).$$

Then from the properties of the random velocity field (discussed above) we see that the following properties should follow:-
1)$< F(x) >$=0. The mean change in the position of a particle at x is zero.
2)We also see that the correlation function of $F(x)$ should be

$$< F(x_1)F(x_2) >= \sigma^2 \tau \delta t e^{-\frac{(x_1-x_2)^2}{2\xi^2}}$$

This property is important because it shows that the dynamics are second ordered, i. e the variance varies linearly with time. As mentioned earlier, we make an assumption of small correlation times. So, we need to solve the dynamics in the limit $\tau \to 0$. This presents a problem that, $< F(x_1)F(x_2) > \to 0$ as $\tau \to 0$, as given by the equation above. This can easily be sorted out by multiplying $F(x)$ with a factor 'g', such that

$$< (gF)^2 > \to finite\ limit\ as\ \tau \to 0.$$

This gives us that $g \propto \frac{1}{\tau}$. For the sake of simplicity, we do not use g in further equations from now on. However we do note that the equations will remain the same, except in the limiting case for which we need the extra

factor. So, the equations are correct for the case of small but non zero τ.
Now we have to simulate $F(x)$. The suitable formula to simulate $F(x)$ is found to be

$$F(x) = \sigma\sqrt{\tau\delta t}\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi, n\in\mathbb{Z}} a_k e^{ikx - \frac{\xi^2 k^2}{4}}$$

where a_k are complex gaussian random numbers with properties:-
1)$< a_k >= 0$
2)$< a_k a_k^* >= 1$
3)$\overline{a_{-k}} = a_k$
It can be easily seen that this will satisfy the requirements of $F(x)$. See the appendix 1 for the proofs and explanation of the above results.
Once again contrast this with the method Deutsch used (see Reference [5]) to simulate the random velocity field as described in the introduction. The random velocity field in Reference [5] is piece-wise linear, continuous, but rarely differentiable, whereas the random velocity field as described above is infinitely smooth.

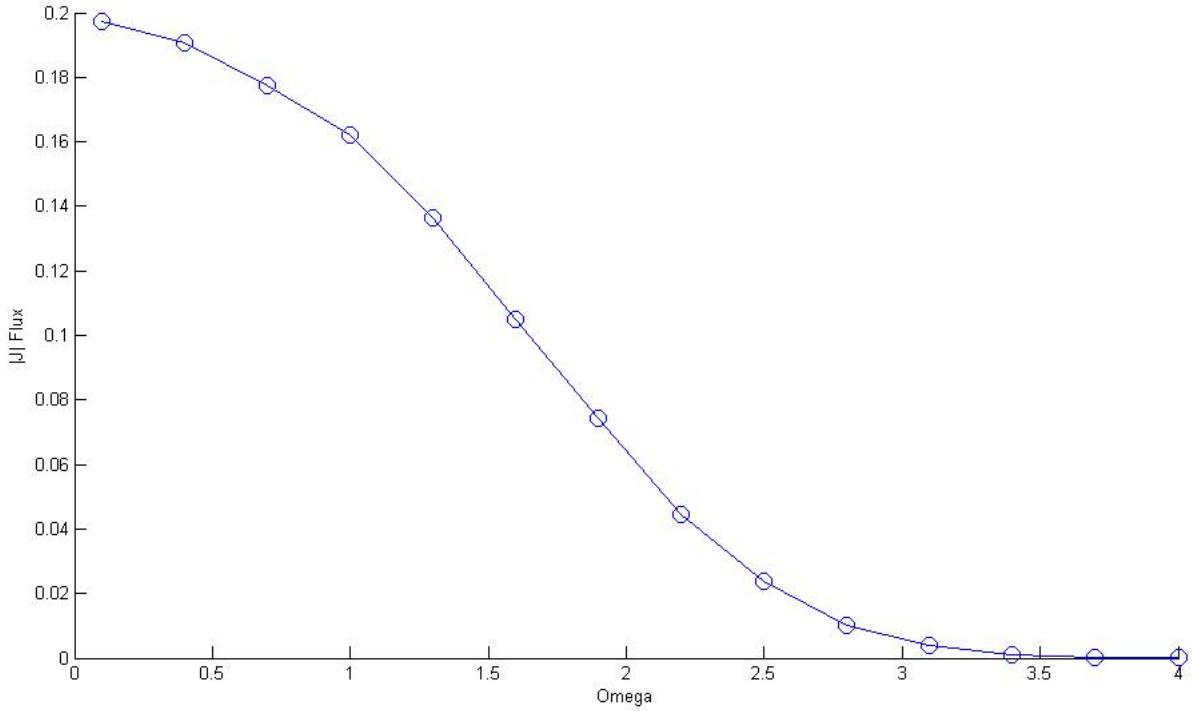

Figure 5: The theoretical and empirical plot of the dimensionless Flux. The circles indicate the empirical value, and the line gives the theoretical value.

3 Path Coalescence- System 2(Discrete Random walk)

In the previous section we discussed a system which exhibits path coalescence. The simplest system which exhibits path coalescence is the 1-dimensional Random walk given by the equation

$$x(t + \delta t) = x(t) + f(x(t), t) \tag{11}$$

where $f(x, t)$ has similar properties as the random velocity field discussed above. This differs from the random velocity field in system 1, in the sense that this velocity field does not have a time correlation, whereas the previous one has time correlation. This is simulated by using the formula

$$f(x,t) = \sigma\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi} a_k e^{ikx - \frac{\xi^2 k^2}{4}}$$

Figure 6 shows the non-coalescing phase while Figure 7 shows the coalescing phase,with the parameter values as shown below the figures. As we will soon illustrate, the behaviour in system-1, in the limit of high viscous

damping (large γ) is the same as in the system-2. We proceed along the same lines as with system to analyse system 2. We start with calculating the lyapunov exponent, and then show that the expressions for the lyapunov exponents in both the systems are the same. This shows analytical equality of lyapunov exponents. Since lyapunov exponents describe the aggregation of particles, we can say that these systems have the same behaviour (in the appropriate limits).

Calculation of Lyapunov Exponent.

We start with linearising equation (11) as

$$\delta x(t+\delta t) = \delta x(t) + f(x_1(t),t) - f(x_2(t),t) \quad where\ \delta x(t) = x_1(t) - x_2(t)$$

Now we can write

$$\begin{pmatrix} f(x_1(t),t) \\ f(x_2(t),t) \end{pmatrix} \sim Normal\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \sigma^2 \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right)$$

where

$$\rho = e^{-\frac{(x_1(t)-x_2(t))^2}{\xi^2}}$$

From this we get that

$$f(x_1(t),t) - f(x_2(t),t) \sim Normal(0, 2\sigma^2(1-\rho))$$

For $\delta x(t)$ small, we have that $\rho \sim 1 - \frac{(\delta x(t))^2}{2\xi^2}$, which gives

$$f(x_1(t),t) - f(x_2(t),t) \sim Normal(0, \sigma^2\frac{(\delta x(t))^2}{\xi^2}) \sim \delta x(t) Normal(0, \frac{\sigma^2}{\xi^2})$$

If we call $F'(x) \sim Normal(0, \frac{\sigma^2}{\xi^2})$, then

$$\begin{aligned} \delta x(t+\delta t) &= \delta x(t)(1+F'(x(t))) \\ \Rightarrow \frac{\delta x(t+\delta t)}{\delta x(0)} &= \frac{\delta x(t)}{\delta x(0)}(1+F') \\ \Rightarrow \frac{\delta x(n\delta t)}{\delta x(0)} &= (1+F')^n \end{aligned}$$

We used the fact that distribution of F' is independent of the position. Now Lyapunov Exponent =

$$\begin{aligned} \Lambda &= \lim_{n\to\infty, \delta x(0)\to 0} \frac{1}{n\delta t} < ln \left|\frac{\delta x(n\delta t)}{\delta x(0)}\right| > \\ &= \lim_{n\to\infty, \delta x(0)\to 0} \frac{1}{n\delta t} < ln\,|(1+F')^n| > \\ &= \frac{1}{\delta t} < ln\,|(1+F')| > \end{aligned}$$

We verify the result by plotting the empirical and theoretical values of the lyapunov exponent in Figure 8.

4 Equality between System 1 and System 2

We will now show that in the limit of $\gamma \to \infty$, we have that the behaviour of equation (1) and equation (2) will match with the behaviour of equation (11) . Firstly under this limit,in equation (1) and equation (2), we have that $\dot{p(t)} = 0$. This is because the frictional force is so high that the particles will be *advected*. This can be seen by looking at the mean relative velocity plot in Figure 4, where the relative velocity drops to 0. This has the implication that if two particles are at the same position then they will stick together in the future.

So under the limit, we have that $\dot{p(t)} = 0$, which gives

$$p(t) = \frac{1}{\gamma} f(x,t) \Rightarrow \dot{x(t)} = \frac{1}{m\gamma} f(x,t)$$

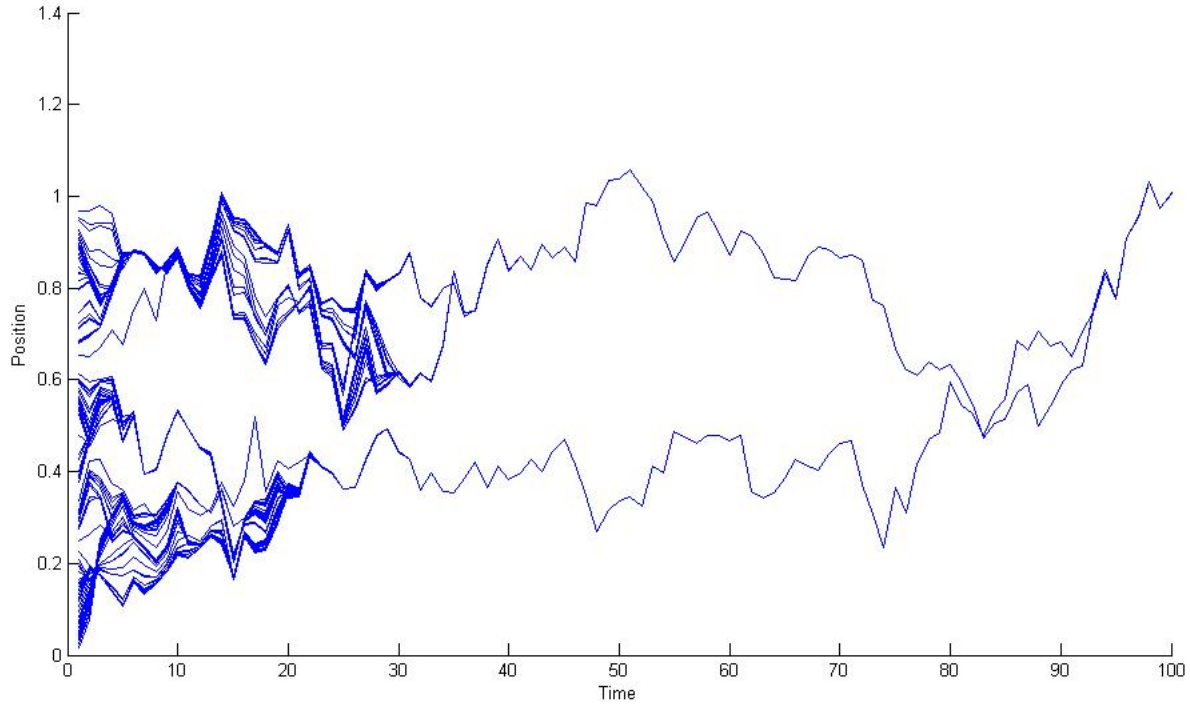

Figure 6: The above plot shows the trajectories of the particles in 1-dimension with time. The x-axis indicates time, and y-axis indicates the position at a given time. The parameter values are $\sigma = 0.05$ $\xi = 0.05$

Discretizing the above equation, we have that

$$x(t+\delta t) = x(t) + \int_t^{t+\delta t} \frac{1}{m\gamma} f(x,t)dt \tag{12}$$

Observe that this equation is the same as the discrete random walk if we take

$$F_1(x) = \frac{1}{m\gamma}\int_t^{t+\delta t} f(x,t) = \frac{1}{m\gamma}F(x)$$

where $F(x) = \int_t^{t+\delta t} f(x,t)$ is as defined in the section on simulation. So we have that

$$Var(F_1) \approx \frac{\sigma^2\tau\delta t}{\gamma^2 m^2}$$

which is obtained from the correlation function of F(x) (in the section on simulation) which is given by

$$< F(x_1)F(x_2) >= \sigma^2\tau\delta t e^{-\frac{(x_1-x_2)^2}{2\xi^2}}$$

and using $x_1 = x_2$. Now, if we identify equation (12) with equation (11) then $F'(x)$ defined in the previous section is equal to $\frac{1}{\xi}F_1$. Substituting the corresponding terms in the expression of lyapunov exponent derived in the last section,we get

$$\Lambda = \frac{1}{\delta t} < ln\left|(1+\frac{1}{\xi}F_1)\right| >$$

In the limit of high γ, we have that $Var(\frac{1}{\xi}F_1)$ is less and hence we can approximate $< ln(1+\frac{1}{\xi}F_1) >\approx -\frac{< F_1^2 >}{2\xi^2}$. Therefore, the lyapunov exponent is approximately

$$\Lambda = \frac{-1}{\delta t}\frac{< F_1^2 >}{2\xi^2} = -\frac{\sigma^2\tau}{2\gamma^2 m^2\xi^2}$$

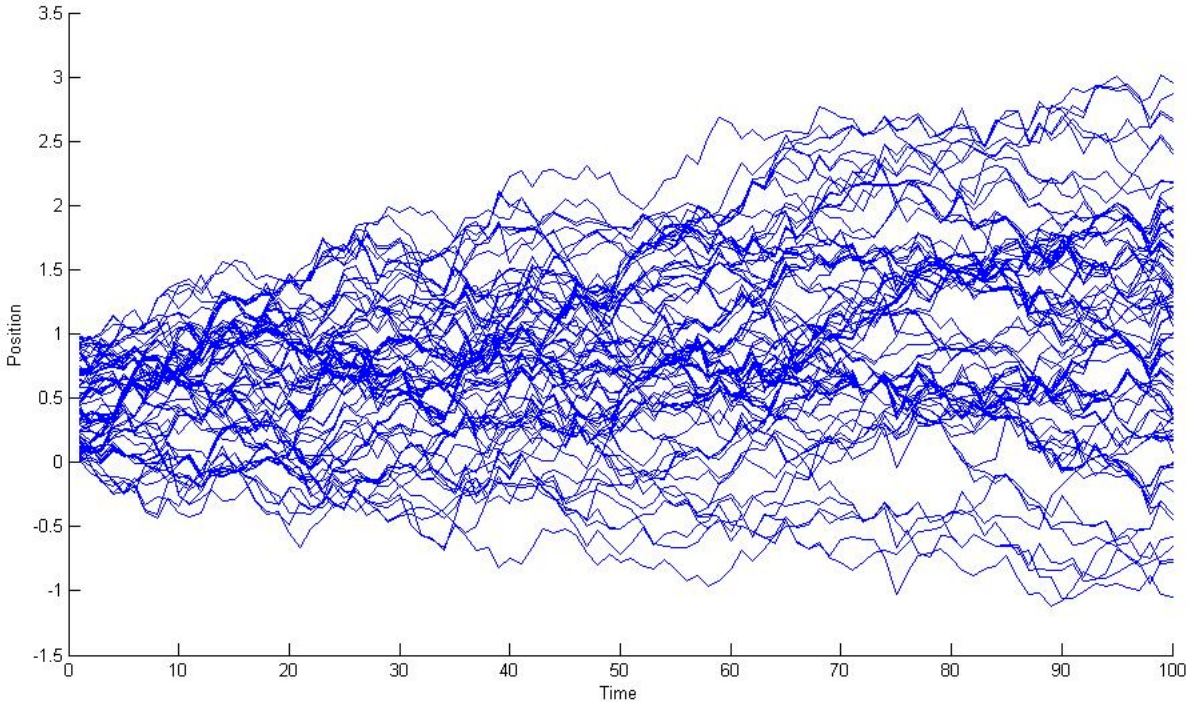

Figure 7: The above plot shows the trajectories of the particles in 1-dimension with time. The x-axis indicates time, and y-axis indicates the position at a given time. The parameter values are $\sigma = 0.1$ $\xi = 0.02$

Another method to calculate lyapunov exponent is to use the limit,$\Omega \to \infty$ in the expression for the lyapunov exponent for system 1 as in equation (9), that is

$$\Lambda = \left(\frac{D}{m^2}\right)^{\frac{1}{3}} \frac{\int_{-\infty}^{\infty} Z\varrho(Z,\Omega)}{\int_{-\infty}^{\infty} \varrho(Z,\Omega)}$$

We defined

$$\Phi(Z,\Omega) = \frac{Z^3}{3} + \frac{\Omega Z^2}{2}$$

then the solution is

$$\varrho(Z,\Omega) = e^{-\Phi(Z,\Omega)} \int_0^z e^{\Phi(Y,\Omega)} dY$$

In the limit of high Ω,the integral $\int_0^z e^{\Phi(Y,\Omega)} dY$ is approximately constant in the region $Z \in [-\Omega, \frac{\Omega}{2}]$. Let the constant be β. So,

$$\begin{aligned} \varrho(Z,\Omega) &\approx \beta e^{-\Phi(Z,\Omega)} \\ &= \beta e^{-\frac{\Omega Z^2}{2}} e^{-\frac{Z^3}{3}} \\ &\approx \beta e^{-\frac{\Omega Z^2}{2}} [1 - \frac{Z^3}{3} + Order(Z^4)] \end{aligned}$$

Now,

$$\begin{aligned} \int_{-\infty}^{\infty} Z\varrho(Z,\Omega) &\approx \int_{-\infty}^{\infty} Z\beta e^{-\frac{\Omega Z^2}{2}} [1 - \frac{Z^3}{3} + Order(Z^4)] \\ &\approx \int_{-\infty}^{\infty} Z\beta e^{-\frac{\Omega Z^2}{2}} - \int_{-\infty}^{\infty} \frac{Z^4}{3} e^{-\frac{\Omega Z^2}{2}} \\ &\approx 0 - \frac{1}{\Omega^2} \end{aligned}$$

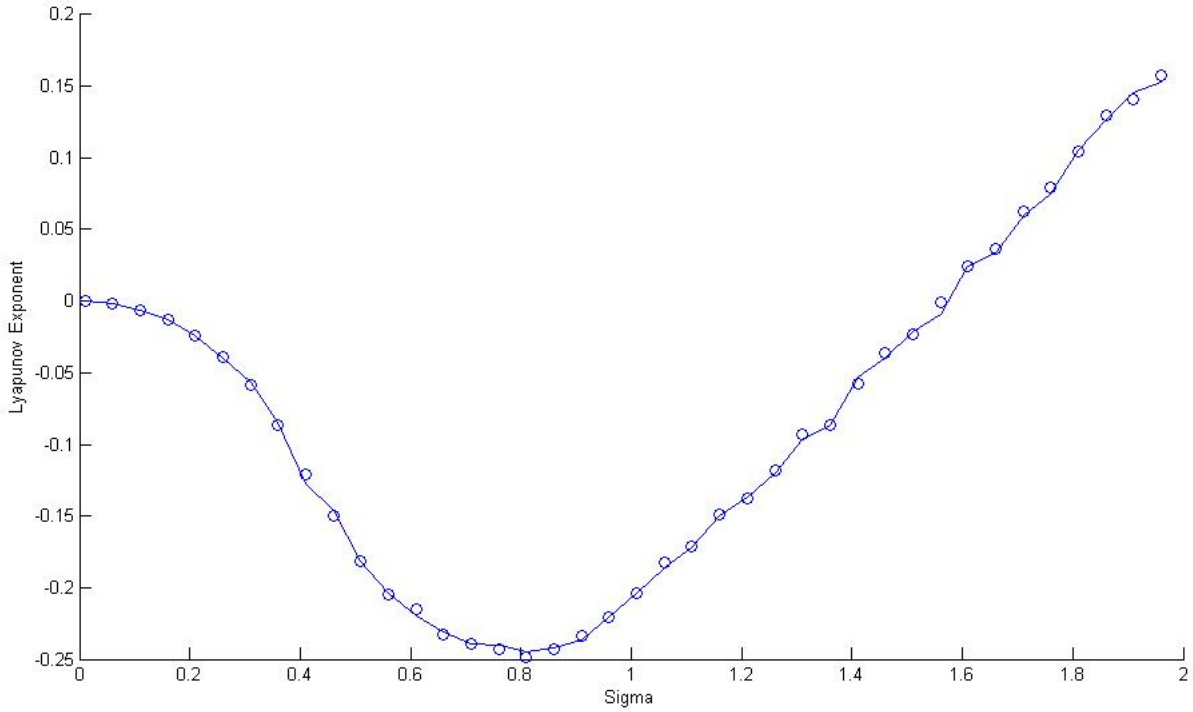

Figure 8: Theoretical(Line) and Empirical(dots) Values of Lyapunov exponents Vs σ

Also the denominator is approximately 1. Hence the lyapunov exponent is approximately equal to

$$\begin{aligned}
\Lambda &= \left(\frac{D}{m^2}\right)^{\frac{1}{3}} \times -\frac{1}{\Omega^2} \\
&= -\frac{D^{1/3}}{m^{2/3}\Omega^2} \\
&= \frac{D}{\gamma^2 m^2} \\
&= -\frac{\sigma^2 \tau}{2\gamma^2 m^2 \xi^2} \quad (Substituting\ D\ from\ before)
\end{aligned}$$

So the lyapunov exponents in both the calculations match analytically in the limit of high γ. This checks that the behaviour of both syatem 1 and system 2 are the same in this limit.

5 Applications of Path Coalescence and Summary

Path coalescence is am example of diffusive type of dynamics ,and can be expected to be observed in many natural situations. Even though the discussion above was done keeping 'particles in a fluid' in mind, the applications are not restricted to this. In the references stated below many applications have been discussed. The applications include, the motion of small droplets on a randomly contaminated surface, motion of energetic electrons in disordered solids,migrating patterns of animals.

The equations of motion can also relate to different scenarios in day to day life. The example below illustrates one such hypothetical but still plausible relation. The equations of motion are

$$\begin{aligned}
\dot{x(t)} &= \frac{p(t)}{m} \\
\dot{p(t)} &= f(x,t) - \gamma p(t)
\end{aligned}$$

Take a society with a lot of people producing (say) cotton. The aim of the game is to decide how much to produce. There is an optimal quantity which everybody should produce(different for each producer) so that they gain maximum profit. Say $x(t)$ denotes the produce of a producer at time t. So the first of the above equations

tells us that the change in his production is given by $p(t)$ which in turn depends on the market(characterised by $f(x,t)$). γ reflects his stubbornness to change.

The assumption that the market is characterised by $f(x,t)$ is valid because:-
1)There is a length correlation ξ, which means that the amount of change for 2 producers producing similar amounts will be similar. This is true intuitively.
2)There is a variance σ^2 which accounts for the fluctuations in the produce.
3)The time correlation takes into consideration how quickly the market can change.

The above analysis then tells us that in such situations there is a possibility that all the producers will produce same amounts of cotton or form clusters depending on how much they produce. An interesting question would then be to look at which producers cluster, and how much would they produce.

In summary, we started with a literary review of the articles related to the path coalescence transition and its applications. Path coalescence appears to be first noted by Deutsch (Reference [5]). Bernard Mehlig and Michael Wilkinson used the same model and characterised it using lyapunov exponent in Reference [14]. They also extended the model to higher dimensions and used perturbation analysis to find out the phase transition, as in Reference [3] and Reference [4]. We also noted some of the work done to understand collisions in such systems. We then restricted our attention to one dimensional system, and described the phase transition it in detail. We used lyapunov exponent to characterise the transition and found out an analytical expression for this. We also described a spatially correlated random walk which exhibits path coalescence (system 2), and showed that in the limit of high inertial effects both these systems have the same qualitative behaviour. We then described a possible application of the system in economics and discussed other applications with it.

References

[1] *Chaos in dynamical systems.* Cambridge University Press, 2002.

[2] A.Provenzale A.Bracco, P.H.Chavanis and E.Spiegel. Particle aggregation in a turbulent keplerian flow. *Phys.Fluids*, 1999.

[3] B.Mehlig and M.Wilkinson. Coagulation by random velocity fields as a kramers problem. *Phys.Rev.Lett.92*, 2004.

[4] K.Duncan T.Weber B.Mehlig, M.Wilkinson and M.Ljunggren. On the aggregation of inertial particles in random flows. *Phys.Rev.E*, 2005.

[5] J M Deutsch. Aggregation-disorder transition induced by fluctuating random forces. *J. Phys. A*, 1985.

[6] Stepanov.M.G Falkovich.G, Fouxon.A. Acceleration of rain initiation by cloud turbulence. *Nature*, 2002.

[7] Kristian Gustaffson. *Inertial Collisions in random flows.* PhD thesis, University Of Gothenberg, 2011.

[8] J.Bec. Fractal clustering of inertial particles in random flows. *Phys. Fluids*, 2003.

[9] M.Cencini A.Lanotte S.Musacchio J.Bec, L.Biferale and F.Toschi. Heavy particle concentration in turbulence at dissipative and inertial scales. *Phys.Rev.Lett.98*, 2007.

[10] L.P.Wang and M.R.Maxey. Settling velocity and concentration distribution of heavy particles in homogeneous isotropic turbulence. *J.Fluid Mech*, 1993.

[11] M.R.Maxey. The gravitational settling of aerosol particles in homogeneous turbulence and random flow fields. *J.Fluid Mech.*, 1987.

[12] S.Ostlund M.Wilkinson, B.Mehlig and K.P.Duncan. Unmixing in random flows. *Phys. Fluids*, 2007.

[13] D Gurarie V I Klyatskin. Coherent phenomena in stochastic dynamical systems. *PHYS-USP*, 1999.

[14] M. Wilkinson and B. Mehlig. Path coalescence transition and its applications. *Phys. Rev. E*, 2003.

A Appendix 1

In this appendix we prove that if we define

$$F(x) = \sigma\sqrt{2\tau\delta t}\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi, n\in\mathbb{Z}} a_k e^{ikx - \frac{\xi^2 k^2}{4}}$$

where a_k are complex gaussian random numbers with properties:-
1)$< a_k >= 0$
2)$< a_k a_k^* >= 1$
3)$\overline{a_{-k}} = a_k$
Then the following properties follow:-
1)$< F(x) >= 0$
2)The second moment is given by

$$< F(x_1)F(x_2) >= \sigma^2 2\tau\delta t e^{-\frac{(x_1-x_2)^2}{2\xi^2}}$$

Proof of 1:

$$\begin{aligned}
< F(x) > &= < \sigma\sqrt{2\tau\delta t}\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi, n\in\mathbb{Z}} a_k e^{ikx - \frac{\xi^2 k^2}{4}} > \\
&= \sigma\sqrt{2\tau\delta t}\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi, n\in\mathbb{Z}} < a_k > e^{ikx - \frac{\xi^2 k^2}{4}} \\
&= \sigma\sqrt{2\tau\delta t}\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi, n\in\mathbb{Z}} 0 \; e^{ikx - \frac{\xi^2 k^2}{4}} \\
&= 0
\end{aligned}$$

Hence Proved.

Proof of 2:

$$\begin{aligned}
< F(x_1)F(x_2)^* > &= \sigma^2 2\tau\delta t\xi\sqrt{2\pi} \sum_{k_i=2n_i\pi, n_1, n_2\in\mathbb{Z}} < a_{k_1} a_{k_2}^* > e^{i(k_1x_1 - k_2x_2) - \frac{\xi^2(k_1^2+k_2^2}{4}} \\
&= \sigma^2 2\tau\delta t\xi\sqrt{2\pi} \sum_{k_i=2n_i\pi, n_1, n_2\in\mathbb{Z}} \delta_{k_1k_2} e^{i(k_1x_1 - k_2x_2) - \frac{\xi^2(k_1^2+k_2^2}{4}} \\
&= \sigma^2 2\tau\delta t\xi\sqrt{2\pi} \sum_{k=2n\pi, n\in\mathbb{Z}} e^{ik(x_1-x_2) - \frac{\xi^2 k^2}{2}} \\
&= \sigma^2 2\tau\delta t\xi\sqrt{2\pi} \sum_{k=2n\pi, n\in\mathbb{Z}} e^{ik(x_1-x_2) - \frac{\xi^2 k^2}{2}}
\end{aligned}$$

Now consider the summation term. We can approximate the sum by

$$\begin{aligned}
\sum_{k=2n\pi, n\in\mathbb{Z}} e^{ik(x_1-x_2) - \frac{\xi^2 k^2}{2}} &= \int_{n=-\infty}^{n=\infty} e^{ik(x_1-x_2) - \frac{\xi^2 k^2}{2}} dn \\
&= \frac{\sqrt{2\pi}}{\xi} \int_{n=-\infty}^{n=\infty} \frac{1}{\frac{\sqrt{2\pi}}{\xi}} e^{ik(x_1-x_2) - \frac{\xi^2 k^2}{2}} \frac{1}{2\pi} dk \\
&= \frac{1}{\xi\sqrt{2\pi}} e^{\frac{(x_1-x_2)^2}{\xi^2}}
\end{aligned}$$

This is because, for small ξ, the variance is large and the sum can be approximated by the integral. The other way to see this is using the poission summation formula. We know that if the variance a gaussian distribution is high, then the variance of its Fourier transform(which is also gaussian) is less, and the only the first term counts. That is, if $f'(l)$ is the Fourier transform of $f(k) = e^{ik(x_1-x_2) - \frac{\xi^2 k^2}{2}}$, then poission summation states that

$$\sum_{k\in\mathbb{Z}} f(k) = \sum_{l\in\mathbb{Z}} f'(l)$$

So here

$$\begin{aligned} f'(l) &= \int_{n=-\infty}^{n=\infty} e^{ik(l+(x_1-x_2))-\frac{\xi^2k^2}{2}}dn \\ &= \frac{\sqrt{2\pi}}{\xi}\int_{n=-\infty}^{n=\infty}\frac{1}{\frac{\sqrt{2\pi}}{\xi}}e^{ik(l+x_1-x_2)-\frac{\xi^2k^2}{2}}\frac{1}{2\pi}dk \\ &= \frac{1}{\xi\sqrt{2\pi}}e^{-\frac{(l+(x_1-x_2))^2}{2\xi^2}} \end{aligned}$$

Since ξ is small,we need to consider only $l = 0$. Hence

$$\sum_{k=2n\pi,n\in\mathbb{Z}} e^{ik(x_1-x_2)-\frac{\xi^2k^2}{2}} = \frac{1}{\xi\sqrt{2\pi}}e^{-\frac{(x_1-x_2)^2}{2\xi^2}}$$

Substituting in $< F(x_1)F(x_2) >$, we get

$$< F(x_1)F(x_2) >= \sigma^2 2\tau\delta t e^{\frac{(x_1-x_2)^2}{2\xi^2}}.$$

Hence Proved.

B Appendix 2

In this appendix we solve the partial differential equation which gives the distribution for the stationary state.The equation we have is

$$\frac{\partial P(X,t)}{\partial X} = \frac{v(X)P(X,t)}{D} + J$$

we first assume that $P(X,t) = P_1(X,t)P_2(X,t)$. Then we have that

$$\begin{aligned} & \frac{\partial P(X,t)}{\partial X} = \frac{\partial P_1(X,t)}{\partial X}P_2(X,t) + \frac{\partial P_2(X,t)}{\partial X}P_1(X,t) \\ \Rightarrow \quad & \frac{\partial P_1(X,t)}{\partial X}P_2(X,t) + \frac{\partial P_2(X,t)}{\partial X}P_1(X,t) = \frac{v(X)P_1(X,t)P_2(X,t)}{D} + J \end{aligned}$$

If we can find $P_1(X,t)$ and $P_2(X,t)$ such that

$$\frac{\partial P_1(X,t)}{\partial X} = \frac{v(X)P_1(X,t)}{D} \quad \textit{and} \tag{13}$$

$$\frac{\partial P_2(X,t)}{\partial X} = \frac{J}{P_1(X,t)} \tag{14}$$

Then we see that the above equation is easily solved.We identify the equation as simple first order differential equation in X with the coefficient also depending on X.The solution to this equation is

$$P_1(X,t) = e^{\frac{1}{D}\int v(X)}$$

Remember that the integral is an indefinite integral.One can easily verify that this is indeed the solution by substituting the equation for $P_1(X,t)$ in the differential equation (13), and verifying that LHS and RHS match.Once we have the solution for $P_1(X,t)$ the solution to the differential equation (14) is just

$$P_2(X,t) = \int J e^{-\frac{1}{D}\int v(X)}$$

So, the final solution for the distribution is

$$\begin{aligned} P(X,t) &= P_1(X,t)P_2(X,t) \\ &= J e^{\frac{1}{D}\int v(X)} \int e^{-\frac{1}{D}\int v(X)} \end{aligned}$$

The expression for J is obtained as shown in the section on *The Stationary State and Formation of Caustics*. See equation (10). The dimensionless expressions for $v(X)$ and D are substituted in the above equation. Also, the expression for J from equation (10) is substituted and simplified to get the final expression for $P(X,t)$ as shown in (8).

C Appendix 3

In this appendix, all the programs, and source codes used to simulate and obtain the results used in the above text are given. All the programs are written in *C++* and i used *Microsoft Visual C++* to compile the programs. In all the programs i used the *Mersenne Twister* random number generator. It is observed that this is sufficient for our purposes. The other more complex random number generators did not give better results compared to the Mersenne Twister. The naming of the variables is straight forward and can be understood from the context.

C.1 Path Coalescence System 1

The following program is used to simulate the system 1. The main aspect of the program is to simulate the random velocity field. The function *velocity_function* is written for this purpose. The equation used to simulate the random velocity is

$$F(x) = \sigma\sqrt{\tau\delta t}\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi, n\in\mathbb{Z}} a_k e^{ikx - \frac{\xi^2 k^2}{4}}$$

This has a complex part in the equation. Using the properties of a_k, we can reduce the above equation by using $a_k = \frac{1}{\sqrt{2}}(a_{k,1}, a_{k,2})$. Also, since $a_k = a_{-k}^{-}$, we can combine these two terms. Then we get

$$F(x) = \sigma\sqrt{2\tau\delta t}\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi\geq 0, n\in\mathbb{Z}} e^{-\frac{\xi^2 k^2}{4}} [a_{k,1} cos(kx) - a_{k,2} sin(kx)]$$

This form of the equation is used to simulate the random velocity field. Note that $a_{k,1}$ and $a_{k,2}$ are real gaussian numbers. n is chosen between[-500, 500].
Initially at t=0, we generate random initial positions for the 100 particles we considered. Then at each time step, we first generate the random numbers a_k, to fix the random velocity field. Then we use the discrete version of equation (1) and equation (2) derived in the section on simulation to carry out the dynamics. Then the positions are stored in a text file, and we proceed to the next time step. The source code is given below.

```
#include <iostream>
#include <string>
#include <fstream>
#include <time.h>
#include "randomc.h"
#include <math.h>
#include "stocc.h"
#include "mersenne.cpp"
#include "userintf.cpp"
#include "stoc1.cpp"

using namespace std;
int seed = (int)time(0);
StochasticLib1 sto(seed);
CRandomMersenne RanGen(seed);


double Coeff[1000];
#define Pi 3.14159265358979323846264 3
double velocity_function(double x,double sigma,double zeta,int index)
{
 int n=0;
 if(index==1)
 {
  for(n=0; n<1000; n++)
 {
  Coeff[n]=sto.Normal(0, 1);
 }
 }
```

```
31   int k=0;
32   double xtmp=x;
33   double ytmp=0;
34   double y;
35   double g1,g2;
36   for (k=0;k<500;k++)
37   {
38    g1=Coeff[2*k];
39    g2=Coeff[2*k+1];
40    ytmp=ytmp+exp(-1*pow(zeta*Pi*k,2))*(g1*cos(2*Pi*k*xtmp)-g2*sin(2*Pi*k*xtmp));
41   }
42   y=ytmp*sigma*sqrt(2*zeta*sqrt(2*Pi));
43   return y;
44  }
45
46  int main()
47  {
48   double sigma=0.1;
49   double zeta= 0.02;
50   int index=0;
51   int i = 0;
52   double t=0,k1=0;
53   double time=1000;
54   int noOfParticles=100;
55   double *particles;
56   particles=new double[noOfParticles];
57   fstream data;
58   data.open("data1.txt");
59   for(int n=0; n<noOfParticles;n++)
60   {
61    particles[n]=RanGen.Random();
62   }
63   for(int n=0; n<noOfParticles;n++)
64   {
65    data<<particles[n]<<"\n";
66   }
67   for(t=0;t<time;t++)
68   {
69    index=1;
70    for(int n=0;n<noOfParticles;n++)
71    {
72     particles[n]=particles[n]+ velocity_function(particles[n],sigma,zeta,index);
73     index= 0 ;
74    }
75    for(int n=0; n<noOfParticles;n++)
76     data<<particles[n]<<"\n";
77   }
78  data.close();
79  return 0;
80  }
```

C.2 Lyapunov Exponent For System 1

This program calculates the lyapunov exponent in system 1. In the previous program we simulated the dynamics of system 1. We simulated the random velocity field using the function *velocity_function*. We use the same function here. There are a lot of ways to calculate lyapunov exponents. We use a straightforward approach here.
We start with two particles whose distance is of the order $d_0 = 10^{-5}$. Then we run the dynamics for some time. We define two thresholds,D_u, the upper threshold, and D_l, the lower threshold such that $D_l < d_0 < D_u$. When the distance $d(t)$ between the particles crosses any of these thresholds,say at time t_1 we define

$$\alpha_1 = \frac{d(t_1)}{d_0}$$

Then we scale the distance so that the particles are once again at a distance d_0. Then we repeat the above process several times, and find out $\alpha_2, \alpha_3, \cdots, \alpha_k$, for k very large. We also keep track of the time elapsed from the start to the end. , say T. The lyapunov exponent is got by

$$\Lambda = \frac{1}{T}\sum_{i=1}^{k} log(\alpha_i)$$

Numerical lyapunov exponent is calculated in the above fashion. The theoretical lyapunov exponent is calculated by approximating the integrals in equation (9). The function *ExpectedLyapunov* is used to approximate the integrals. The source code is given below.

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <time.h>
5  #include "randomc.h"
6  #include <math.h>
7  #include "stocc.h"
8  #include "mersenne.cpp"
9  #include "userintf.cpp"
10 #include "stoc1.cpp"
11
12 using namespace std;
13
14 int seed = (int)time(0);
15 StochasticLib1 sto(seed);
16 CRandomMersenne RanGen(seed);
17
18
19 double Coeff[1000];
20 #define Pi 3.14159265358979323846264 3
21
22 double ExpectedLyapunov (double Omega)
23 {
24 double si1=0,si0=0,si1tmp=0,si0tmp=0;
25 for(double z=-30;z<30;z=z+0.01)
26 {
27 si1tmp=0;
28 for(double y=-30;y<z;y=y+0.01)
29  si1tmp=si1tmp+0.01*(exp(((pow(y,3)-pow(z,3))/3+Omega*((pow(y,2)-pow(z,2))/2))))
       ;
30 si1=si1+si1tmp*z*0.01;
31 si0=si0+si1tmp*0.01;
32 }
33 double ele=si1/si0;
34 return ele;
35 }
36 double velocity_function(double x,double sigma,double zeta,double tau, double
     deltat, double g,int index)
```

```
37 {
38  int n=0;
39  if(index==1)
40  for(n=0; n<1000; n++)
41    Coeff[n]=sto.Normal(0, 1);
42  int k=0;
43  double xtmp=x;
44  double ytmp=0;
45  double y;
46  double g1,g2;
47  ytmp=ytmp+Coeff[0];
48  for (k=1;k<500;k++)
49  {
50  g1=Coeff[2*k];
51  g2=Coeff[2*k+1];
52  ytmp=ytmp+exp(-1*pow(zeta*Pi*k,2))*(g1*cos(2*Pi*k*xtmp)-g2*sin(2*Pi*k*xtmp));
53  }
54  y=ytmp*g*sigma*sqrt(2*zeta*tau*deltat*sqrt(2*Pi));
55  return y;
56 }
57
58 int main()
59 {
60  double sigma=0.001,xi=0.05,gamma=0.001,tau=1,deltat=1,mass=1;
61  double g=pow(tau,-0.5);
62  double diffusion=pow(sigma,2)*tau*pow(g,2)*deltat/(pow(xi,2));
63  double Omega=gamma*pow(mass,0.6666)/pow(diffusion,0.33333);
64  double deltax=exp(-6.0)*(1-2*RanGen.IRandomX(0,1));
65  double LE=0,ExLE=0;
66  double x1=0,x2=0,vx1=0,vx2=0;
67  double x1tmp=0,x2tmp=0,vx1tmp=0,vx2tmp=0;
68  double t=0,time=500,d=fabs(deltax),d1=fabs(deltax);
69  int index=0,i=0;
70  double n=0;
71  int noOfObs=1000;
72  x1=RanGen.Random();
73  x2=x1+deltax;
74  fstream data;
75  data.open("datafinal1.txt");
76  for(Omega=0.1;Omega<5;Omega=Omega+0.1)
77  {
78   deltax=pow(10.0,-5)*(1-2*RanGen.IRandomX(0,1));
79   x1=RanGen.Random();
80   x2=x1+deltax;
81   vx1=0;
82   vx2=0;
83   LE=0;
84   t=0;
85   i=0;
86   d=fabs(deltax);
87   ExLE=ExpectedLyapunov(Omega);
88   gamma=Omega*pow(diffusion,0.3333333)*pow(mass,-0.6666666);
89   while(i<5*pow(10.0,3) && t<1*pow(10.0,6))
90   {
91    while(d>5*pow(10.0,-8) && d< pow(10.0,-2))
92    {
93     index=1;
94     x1tmp=x1+deltat*vx1/mass;
95     vx1tmp=vx1+deltat*(velocity_function(x1,sigma,xi,tau,deltat,g,index)-gamma*
           vx1);
96     index=0;
```

```
97        x2tmp=x2+deltat*vx2/mass;
98        vx2tmp=vx2+deltat*(velocity_function(x2,sigma,xi,tau,deltat,g,index)-gamma*
              vx2);
99        x1=x1tmp;
100       x2=x2tmp;
101       vx1=vx1tmp;
102       vx2=vx2tmp;
103       t=t+1;
104       d=pow(pow((x2-x1),2)+pow((vx1-vx2),2),0.5);
105     }
106     LE=LE+log(fabs(d/deltax));
107     cout<<Omega<<","<<LE*pow(diffusion,-0.3333333)/t<<","<<ExLE<<","<<t<<","<<endl
            ;
108     x2=x1+deltax;
109     vx1=0;
110     vx2=0;
111     d=fabs(deltax);
112     i++;
113   }
114   LE=LE*pow(diffusion,-0.3333333)/t;
115   cout<<Omega<<","<<LE<<","<<ExLE<<","<<t<<","<<endl;
116   data<<Omega<<","<<LE<<","<<ExLE<<","<<"\n";
117   return 0;
118 }
```

C.3 Mean Final Velocities System 1

In this program we calculate the mean final velocities of the particles in the stationary state. Remember that the stationary state is given by $X = \frac{\delta p}{\delta x}$, which is only again valid in the limits of small δp and δx. So once again we use thresholds to mark the boundaries where the stationary state can be seen. We use the same function as in the previous two programs to simulate the random velocity field. We start with two particles close to each other and run the dynamics for some time(fixed beforehand), and reset the system. We repeat this several times and take the average of the final relative velocities of the particles. This is done for each value of Ω.

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <time.h>
5  #include "randomc.h"
6  #include <math.h>
7  #include "stocc.h"
8  #include "mersenne.cpp"
9  #include "userintf.cpp"
10 #include "stoc1.cpp"
11 using namespace std;
12 int seed = (int)time(0);
13 StochasticLib1 sto(seed);
14 CRandomMersenne RanGen(seed);
15
16
17 double Coeff[1000];
18 #define Pi 3.14159265358979323846264 3
19
20 double velocity_function(double x,double sigma,double zeta,double tau, double
       deltat, double g,int index)
21 {
22
23          int n=0;
24
25          if(index==1)
26          {
27                  for(n=0; n<1000; n++)
28                  {
29                          Coeff[n]=sto.Normal(0, 1);
30                  }
31          }
32          int k=0;
33          double xtmp=x;
34          double ytmp=0;
35          double y;
36          double g1,g2;
37          ytmp=ytmp+Coeff[0];
38          for (k=1;k<500;k++)
39          {
40                  g1=Coeff[2*k];
41                  g2=Coeff[2*k+1];
42                  ytmp=ytmp+exp(-1*pow(zeta*Pi*k,2))*(g1*cos(2*Pi*k*xtmp)-g2*sin
                        (2*Pi*k*xtmp));
43          }
44          y=ytmp*g*sigma*sqrt(zeta*tau*deltat*sqrt(2*Pi))*2;
45          return y;
46 }
47
48 int main()
49 {
```

```
50          double sigma=0.001,xi=0.05,gamma=0.001,tau=1,deltat=1,mass=1;
51          double g=pow(tau,-0.5);
52          double diffusion=pow(sigma,2)*tau*pow(g,2)*deltat/pow(xi,2);
53          double Omega=gamma*pow(mass,0.6666)/pow(diffusion,0.33333);
54          double deltax=exp(-6.0)*(1-2*RanGen.IRandomX(0,1));
55          double finalVelocity=0;
56          double x1=0,x2=0,vx1=0,vx2=0;
57          double x1tmp=0,x2tmp=0,vx1tmp=0,vx2tmp=0;
58          double t=0,time=500,d=fabs(deltax),d1=fabs(deltax);
59          double index=0,i=0;
60          double n=0;
61          int noOfObs=1000;
62
63          x1=RanGen.Random();
64          x2=x1+deltax;
65          fstream data;
66          data.open("data10.txt");
67          for(Omega=0.1;Omega<3;Omega=Omega+0.1)
68          {
69                  deltax=pow(10.0,-5)*(1-2*RanGen.IRandomX(0,1));
70                  x1=RanGen.Random();
71                  x2=x1+deltax;
72                  vx1=0;
73                  vx2=0;
74                  finalVelocity=0;
75                  t=0;
76                  i=0;
77                  d=fabs(deltax);
78                  gamma=Omega*pow(diffusion,0.3333)*pow(mass,-0.6666);
79                  while(i<5*pow(10.0,3) && t<1*pow(10.0,6))
80                  {
81                          while(d>5*pow(10.0,-8) && d< pow(10.0,-3))
82                          {
83                                  index=1;
84                                  x1tmp=x1+deltat*vx1/mass;
85                                  vx1tmp=vx1+deltat*(velocity_function(x1,sigma,xi
                                        ,tau,deltat,g,index)-gamma*vx1);
86                                  index=0;
87                                  x2tmp=x2+deltat*vx2/mass;
88                                  vx2tmp=vx2+deltat*(velocity_function(x2,sigma,xi
                                        ,tau,deltat,g,index)-gamma*vx2);
89                                  x1=x1tmp;
90                                  x2=x2tmp;
91                                  vx1=vx1tmp;
92                                  vx2=vx2tmp;
93                                  t=t+1;
94                                  d=pow(pow((x2-x1),2)+pow((vx1-vx2),2),0.5);
95                          }
96                          finalVelocity=finalVelocity+fabs(vx1-vx2);
97                          x2=x1+deltax;
98                          index=1;
99                          vx1=0;
100                         index=0;
101                         vx2=0;
102                         d=fabs(deltax);
103                         i++;
104                 }
105                 finalVelocity=finalVelocity/i;
106                 cout<<Omega<<","<<finalVelocity<<endl;
107                 data<<Omega<<","<<finalVelocity<<"\n";
108         }
```

```
109             return 0;
110     }
```

C.4 Rate of Crossing Caustics

This Program is used to calculate the rate of crossing caustics, that is J(flux) described in the text above. Note that the flux should be calculated in the stationary state described in the section on stationary state. We defined the stationary state in the limits of small δp and δx. The caustic appears when two particles cross each other or collide with each other.
The random velocity is simulated by the function *velocity_function*. We start with 2 particles close to each other, and run the dynamics until the distance between them becomes too large or too small. In which case we rescale the distance and start the dynamics again. If the total number of caustics is n, and the total time is t, then the rate of appearance of caustics is $\frac{n}{t}$.

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <time.h>
5  #include "randomc.h"
6  #include <math.h>
7  #include "stocc.h"
8  #include "mersenne.cpp"
9  #include "userintf.cpp"
10 #include "stoc1.cpp"
11
12 using namespace std;
13 int seed = (int)time(0);
14 StochasticLib1 sto(seed);
15 CRandomMersenne RanGen(seed);
16
17
18 double Coeff[1000];
19 #define Pi 3.14159265358979323846264 3
20
21 double velocity_function(double x,double sigma,double zeta,double tau, double
       deltat, double g,int index)
22 {
23
24          int n=0;
25
26          if(index==1)
27          {
28                  for(n=0; n<1000; n++)
29                  {
30                          Coeff[n]=sto.Normal(0, 1);
31                  }
32          }
33          int k=0;
34          double xtmp=x;
35          double ytmp=0;
36          double y;
37          double g1,g2;
38          ytmp=ytmp+Coeff[0];
39          for (k=1;k<500;k++)
40          {
41                  g1=Coeff[2*k];
42                  g2=Coeff[2*k+1];
43                  ytmp=ytmp+exp(-1*pow(zeta*Pi*k,2))*(g1*cos(2*Pi*k*xtmp)-g2*sin
                        (2*Pi*k*xtmp));
44          }
45          y=ytmp*g*sigma*sqrt(zeta*tau*deltat*sqrt(2*Pi))*2;
46          return y;
47 }
48
```

```
49 int main()
50 {
51         double sigma=0.001,xi=0.05,gamma=0.001,tau=1,deltat=1,mass=1;
52         double g=pow(tau,-0.5);
53         double diffusion=pow(sigma,2)*tau*pow(g,2)*deltat/pow(xi,2);
54         double Omega=gamma*pow(mass,0.6666)/pow(diffusion,0.33333);
55         double deltax=exp(-6.0)*(1-2*RanGen.IRandomX(0,1));
56         double LE=0,ExLE=0;
57         double x1=0,x2=0,vx1=0,vx2=0;
58         double x1tmp=0,x2tmp=0,vx1tmp=0,vx2tmp=0;
59         double t=0,time=500,d=fabs(deltax),d1=fabs(deltax);
60         int index=0,i=0;
61         double n=0;
62         int noOfObs=1000;
63
64         x1=RanGen.Random();
65         x2=x1+deltax;
66         fstream data;
67         data.open("data8.txt");
68         for(Omega=0.1;Omega<4;Omega=Omega+0.3)
69         {
70                 deltax=pow(10.0,-5)*(1-2*RanGen.IRandomX(0,1));
71                 x1=RanGen.Random();
72                 x2=x1+deltax;
73                 vx1=0;
74                 vx2=0;
75                 t=0;
76                 n=0;
77                 i=0;
78                 d=fabs(deltax);
79                 gamma=Omega*pow(diffusion,0.3333)*pow(mass,-0.6666);
80                 while(i<5*pow(10.0,3) && t<1*pow(10.0,6))
81                 {
82                         while(d>5*pow(10.0,-11) && d< pow(10.0,-2))
83                         {
84                                 index=1;
85                                 x1tmp=x1+deltat*vx1/mass;
86                                 vx1tmp=vx1+deltat*(velocity_function(x1,sigma,xi
                                        ,tau,deltat,g,index)-gamma*vx1);
87                                 index=0;
88                                 x2tmp=x2+deltat*vx2/mass;
89                                 vx2tmp=vx2+deltat*(velocity_function(x2,sigma,xi
                                        ,tau,deltat,g,index)-gamma*vx2);
90                                 t=t+1;
91                                 d=pow(pow((x2-x1),2)+pow((vx1-vx2),2),0.5);
92                                 if((x2-x1)*(x2tmp-x1tmp)<0)
93                                         n=n+1;
94                                 x1=x1tmp;
95                                 x2=x2tmp;
96                                 vx1=vx1tmp;
97                                 vx2=vx2tmp;
98                         }
99                         x2=x1+deltax;
100                        vx1=0;
101                        vx2=0;
102                        d=fabs(deltax);
103                        i++;
104                }
105                cout<<Omega<<","<<n*pow(mass,0.6666)/(pow(diffusion,0.33333)*t)
                       <<","<<ExLE<<","<<"\n";
```

```
106                     data<<Omega<<","<<n*pow(mass,0.6666)/(pow(diffusion,0.33333)*t)
                            <<","<<ExLE<<","<<"\n";
107             }
108             return 0;
109     }
```

C.5 Path Coalescence System 2

The following program is used to simulate the system 2. The main aspect of the program is to simulate the random velocity field. The function *velocity_function* is written for this purpose. The equation used to simulate the random velocity is

$$F(x) = \sigma\sqrt{\xi\sqrt{2\pi}} \sum_{k=2n\pi, n\in\mathbb{Z}} a_k e^{ikx - \frac{\xi^2 k^2}{4}}$$

This has a complex part in the equation. Using the properties of a_k, we can reduce the above equation by using $a_k = \frac{1}{\sqrt{2}}(a_{k,1}, a_{k,2})$. Also, since $a_k = a_{-k}^-$, we can combine these two terms. Then we get

$$F(x) = \sigma\sqrt{\xi 2\sqrt{2\pi}} \sum_{k=2n\pi\geq 0, n\in\mathbb{Z}} e^{-\frac{\xi^2 k^2}{4}} [a_{k,1} cos(kx) - a_{k,2} sin(kx)]$$

This form of the equation is used to simulate the random velocity field. Note that $a_{k,1}$ and $a_{k,2}$ are real gaussian numbers. Also note the difference between this formula and he one used to simulate the random velocity field in the system with inertial effects. n is chosen between[-500, 500].
Initially at t=0, we generate random initial positions for the 100 particles we considered. Then at each time step, we first generate the random numbers a_k, to fix the random velocity field. Then we use equation (11) to carry out the dynamics. Then the positions are stored in a text file, and we proceed to the next time step. The source code is given below.

```
#include <iostream>
#include <string>
#include <fstream>
#include <time.h>
#include "randomc.h"
#include <math.h>
#include "stocc.h"
#include "mersenne.cpp"
#include "userintf.cpp"
#include "stoc1.cpp"

using namespace std;
int seed = (int)time(0);
StochasticLib1 sto(seed);
CRandomMersenne RanGen(seed);


double Coeff[1000];
#define Pi 3.14159265358979323846264 3

double velocity_function(double x,double sigma,double zeta,int index)
{
 int n=0;
 if(index==1)
 {
  for(n=0; n<1000; n++)
 {
  Coeff[n]=sto.Normal(0, 1);
 }
 }
 int k=0;
 double xtmp=x;
 double ytmp=0;
 double y;
 double g1,g2;
 for (k=0;k<500;k++)
 {
  g1=Coeff[2*k];
```

```
39     g2=Coeff[2*k+1];
40     ytmp=ytmp+exp(-1*pow(zeta*Pi*k,2))*(g1*cos(2*Pi*k*xtmp)-g2*sin(2*Pi*k*xtmp));
41    }
42    y=ytmp*sigma*sqrt(2*zeta*sqrt(2*Pi));
43    return y;
44  }
45
46  int main()
47  {
48   double sigma=0.1;
49   double zeta= 0.02;
50   int index=0;
51   int i = 0;
52   double t=0,k1=0;
53   double time=1000;
54   int noOfParticles=100;
55   double *particles;
56   particles=new double[noOfParticles];
57   fstream data;
58   data.open("data1.txt");
59   for(int n=0; n<noOfParticles;n++)
60   {
61    particles[n]=RanGen.Random();
62   }
63   for(int n=0; n<noOfParticles;n++)
64   {
65    data<<particles[n]<<"\n";
66   }
67   for(t=0;t<time;t++)
68   {
69    index=1;
70    for(int n=0;n<noOfParticles;n++)
71    {
72     particles[n]=particles[n]+ velocity_function(particles[n],sigma,zeta,index);
73     index= 0 ;
74    }
75    for(int n=0; n<noOfParticles;n++)
76     data<<particles[n]<<"\n";
77   }
78  data.close();
79  return 0;
80  }
```

C.6 Lyapunov Exponent System 2

This program calculates the lyapunov exponent in system 2. In the previous program we simulated the dynamics of system 2. We simulated the random velocity field using the function *velocity_function*. We use the same function here. There are a lot of ways to calculate lyapunov exponents. We use a straightforward approach here.
We start with two particles whose distance is of the order $d_0 = 10^{-5}$. Then we run the dynamics for some time. We define two thresholds,D_u, the upper threshold, and D_l, the lower threshold such that $D_l < d_0 < D_u$. When the distance $d(t)$ between the particles crosses any of these thresholds,say at time t_1 we define

$$\alpha_1 = \frac{d(t_1)}{d_0}$$

Then we scale the distance so that the particles are once again at a distance d_0. Then we repeat the above process several times, and find out $\alpha_2, \alpha_3, \cdots, \alpha_k$, for k very large. We also keep track of the time elapsed from the start to the end. , say T. The lyapunov exponent is got by

$$\Lambda = \frac{1}{T}\sum_{i=1}^{k} log(\alpha_i)$$

Numerical lyapunov exponent is calculated in the above fashion. The theoretical lyapunov exponent is calculated by using monte-carlo simulations . The function *ExpectedLyapunov* is used to approximate the integrals. The source code is given below.

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <time.h>
5  #include "randomc.h"
6  #include <math.h>
7  #include "stocc.h"
8  #include "sfmt.h"
9
10
11 // The followong are the files i do not know ehther to include or not
12
13 #include "mersenne.cpp"
14 #include "userintf.cpp"
15 #include "stoc1.cpp"
16 #include "sfmt.cpp"
17 using namespace std;
18
19 int seed = (int)time(0);
20 StochasticLib1 sto(seed);
21 CRandomMersenne RanGen(seed);
22
23
24 double Coeff[1000];
25 #define Pi 3.14159265358979323846264 3
26 double velocity_function(double x,double sigma,double zeta,int index)
27
28 {
29
30         int n=0;
31
32         if(index==1)
33         {
34                 for(n=0; n<1000; n++)
35                 {
36                         Coeff[n]=sto.Normal(0, 1);
37                 }
38         }
```

```
39          int k=0;
40          double xtmp=x;
41          double ytmp=0;
42          double y;
43          double g1,g2;
44          for (k=0;k<500;k++)
45          {
46                  g1=Coeff[2*k];
47                  g2=Coeff[2*k+1];
48                  ytmp=ytmp+exp(-1*pow(zeta*Pi*k,2))*(g1*cos(2*Pi*k*xtmp)-g2*sin
                        (2*Pi*k*xtmp));
49          }
50          y=ytmp*sigma*sqrt(2*zeta*sqrt(2*Pi));
51          return y;
52  }
53
54  double ExpectedLyapunov(double sigma)
55  {
56          double a=0;
57          double g1=0;
58          for(int i=0;i<100000;i++)
59          {
60                  g1=sto.Normal(0, sigma);
61                  a=a+log(fabs(1+g1));
62          }
63          a=a/100000;
64          return a;
65  }
66
67  int main()
68  {
69          double sigma=0,xi= 0.05;
70          int index=0;
71          double Ele=0,le=0;
72          fstream data;
73          data.open("data2.txt");
74          double t=0;
75          double x1=RanGen.Random();
76          double deltax=exp(-5.0)*(1-2*RanGen.IRandomX(0,1));
77          double x2=x1+deltax;
78          double d=fabs(deltax);
79          double sigmap=0;
80          int i=0;
81          for(sigma=0.01;sigma<2;sigma=sigma+0.05)
82          {
83                  x1=RanGen.Random();
84                  deltax=exp(-5.0)*(1-2*RanGen.IRandomX(0,1));
85                  x2=x1+deltax;
86                  sigmap=sigma*xi;
87                  t=0;
88                  le=0;
89                  i=0;
90                  d=fabs(deltax);
91                  Ele=ExpectedLyapunov(sigma);
92                  while(i<5000 && t<pow(10.0,6))
93                  {
94                          while(d> exp(-8.0)*4 && d< exp(-4.0))
95                          {
96                                  index=1;
97                                  x1=x1+(velocity_function(x1,sigmap,xi,index));
98                                  index=0;
```

```
 99                                      x2=x2+(velocity_function(x2,sigmap,xi,index));
100                                      t=t+1;
101                                      d=fabs(x2-x1);
102                              }
103                              le=le+log(fabs(d/deltax));
104                              x2=x1+(x2-x1)*deltax/d;
105                              d=fabs(x2-x1);
106                              i=i+1;
107                              //cout<<i<<","<<le/t<<","<<t<<","<<Ele<<endl;
108                      }
109                      le=le/t;
110                      data<<sigma<<","<<le<<","<<Ele<<"\n";
111                      cout<<sigma<<","<<le<<","<<Ele<<endl;
112              }
113
114              data.close();
115
116
117      }
```