Mixing in Microchannels

Martin A Ewart

Imperial College, London, SW7 2AZ

22 Sedgeley Avenue,
Rochdale,
Lancashire,
OL16 4TZ,
United Kingdom

mail@martinewart.co.uk

Supervisor: Dr Jean-Luc Thiffeault

June 2004

Abstract

The ability to mix efficiently has many uses, particularly in chemistry and biochemistry. A promising area involves mixing in microchannels—narrow grooves of about 100μm in width where fluid travels in so-called "lab-on-a-chip" applications [11].

A microchannel which mixes fluids is called a micromixer. There are two types of mixing methods in microchannels: active and passive. In active mixing, fluid is pumped into the channel in such a way that mixing is induced, say by a time-dependent forcing. However this is difficult to do on such tiny scales as micrometres, and is also undesirable from a manufacturing standpoint because the design involves moving parts. In passive mixing, the shape of the micromixer is designed to create flow patterns that naturally mix. For example, this is achieved by placing grooves on the walls of the channels which cause chaotic motions of fluid particles. An alternative approach, used in present work, is electro-osmosis, where electric fields are used to push the fluid [1].

In this project passive mixing is studied. Many studies of passive mixing have been made using numerical approaches (such as finite differences, finite elements, etc.) to model flows in microchannels [9]. However in this project the feasibility of using an analytic model for the velocity field of certain simple configurations is investigated. The benefit of having an analytical solution is that a more accurate computation of the flow is performed and its mixing properties are thus more readily assessed.

Acknowledgements

I would like to thank my supervisor, Dr Jean-Luc Thiffeault for the help and support he has given me throughout the project. In addition to my work on the project, attending the talks to which Dr Thiffeault invited me was also interesting. These gave me a fantastic insight into some of the current fluid mechanics topics. In addition, the fluids courses by Dr Craster, Dr Crowdy, Dr Mestel and Dr Thiffeault have been invaluable in helping my understanding of fluids.

Information

There is a CD which accompanies this document. The main program files that have been used have been printed in an appendix at the back of this document. These files plus other additional files that were used for creating certain plots and visualisations are on the CD. In addition there is an animation on the CD to which the project refers. There is a readme file with details about using the CD in the root directory.

Contents

Chapter 1

Introduction

1.1 Background

A microchannel is a channel that has a width and height in the order of micrometers (μm). Conventional mixing methods for larger volumes of fluid are often not practical at such tiny scales, so micromixing requires a separate area of research. Many recent studies have been performed on passive mixing in microchannels [5, 6, 9, 10, 11, 12]. There are different methods of generating chaotic flows in channels, varying from placing obstacles in the channel [12] to a very popular and effective solution called the "staggered herringbone" system [9].

The "staggered herringbone" system is where grooves in a herringbone shape are carved into the floor of the channel (see figure 1.1). The herringbone pattern is periodic in the x-direction and half way through each period the herringbone is flipped around, hence creating a staggered pattern. In this system the efficiency of mixing is far greater than the mixing from diffusion alone [6].

An alternative to actually carving grooves into the floor of the channel (which is hard to manufacture due to the tiny size) is to use electro-osmosis. This is where an electric field is used to induce flow on the floor of the channel [9]. Many different electric fields can be created on the base of the channel including an equivalent to the "staggered herringbone" pattern.

1.2 Project Aims

The aim of this project is to analytically model flows induced by electro-osmosis in microchannels. The electric field used is a step-function in the x-direction in the form of figure 1.2 and is placed in a "non-staggered herringbone" (or simply herringbone) pattern like in figure 1.3. Analytically it is

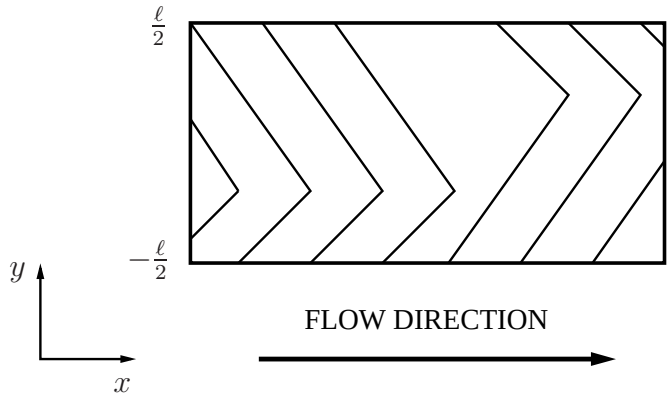


Figure 1.1: A top down view of the staggered herringbone pattern on the floor of the channel

much more difficult to model the staggered system; this is a future area of investigation beyond this project. Previous simulations of flows in microchannels have used a numerical method like finite elements to calculate the velocity field in the channel [5]. Due to only having the knowledge of the velocity field at set points, there are inaccuracies in the velocity of particles as interpolation is required between the points. This also makes it difficult to preserve incompressibility. The reward for finding an analytical solution for the velocity field is that computationally it is much easier to model particle trajectories with a known velocity field. This enables the modelling of many different herringbone configurations to find the best mixing configurations as well as accurate derivatives to enable the calculation of finite time Lyapunov exponents (FTLEs) which provide an indication of "mixing time".

In this project different specifications of mixers are analysed. The cross-section is modelled as in figure 1.4. h and ℓ are the height and width of the channel respectively. Also the micromixers are periodic in the x-direction with period length L. The microchannels we model have a width of 100μm with heights varying from 10 to 50μm, so there is a low aspect ratio between height and width. Different flow rates (u) in the x-direction of 100μm/s and 500μm/s are studied. For simplicity we scale all values with respect to $\ell = 100\mu$m. Therefore $h \in [0.1, 0.5]$ and $u \in [1, 5]$. In addition we set $L/\ell = 2\pi$ so $L \sim 628\mu$m.

Micromixers can have varying velocities from low rate 100μm/s to high rate $10,000\mu$m/s. Low rate mixers are generally much better than high rate mixers [6]. This is due to the low Reynolds number which is defined

$$\mathrm{Re} = \frac{uL}{\nu}$$

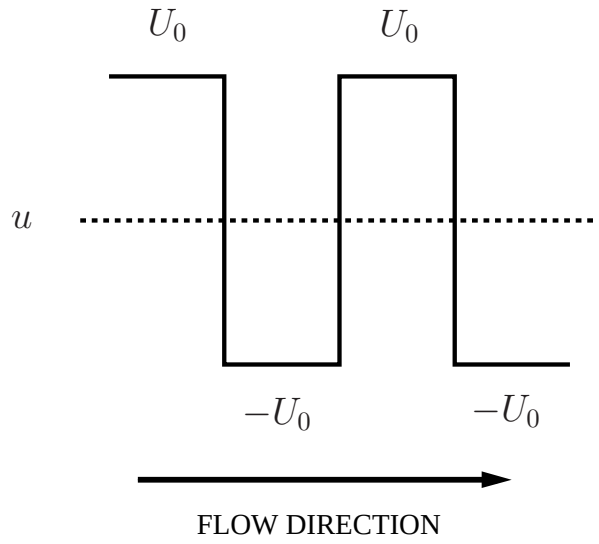


Figure 1.2: Step function for velocity in x-direction at the bottom of the channel.

where ν is the viscosity and we have used the period length L as the length. Assuming that the substance to be mixed has a viscosity similar to water (i.e. $\nu = 10^{-6}\mathrm{m}^2/\mathrm{s}$) then the mixers we study have Reynolds numbers of 0.063 to 0.314. Larger velocities like $10,000\mu\mathrm{m/s}$ have Reynolds number similar to 6.28 which makes mixing more difficult.

In addition to the analytical model and a program to simulate flows, we also analyse the efficiency of mixing within the channels. Mixing for different configurations is studied and some good mixers are found. Poincaré sections, Dispersion plots, Residence times and Lyapunov exponents are all studied to try to find good mixing properties.

It is difficult to define what is meant by good mixing. The aim of mixing in microchannels is to fully mix the fluid in the cross-section in the shortest possible distance down the channel. To get an idea, we look at mixing for DNA. DNA has diffusivity coefficient $\kappa = 10^{-10}\mathrm{m}^2/\mathrm{s}$. Therefore a typical mixing time to diffuse the width of our channel is

$$T_{\mathrm{diff}} = \frac{\ell^2}{\kappa} = \frac{(10^{-4}\mathrm{m})^2}{10^{-10}\mathrm{m}^2/\mathrm{s}} = 100\mathrm{s}\,.$$

At our slowest velocity of $10^{-4}\mathrm{m/s}$ this would take a distance of 1 cm to just diffuse the width of the channel and for the faster velocity of $5 \times 10^{-4}\mathrm{m/s}$ it would take 5 cm. This is a large distance when dealing with tiny amounts of DNA. Mixing needs to be performed in a distance in the order of millimetres rather than centimetres. Therefore we do not want to rely on diffusion alone which motivates a study of ways to improve mixing.

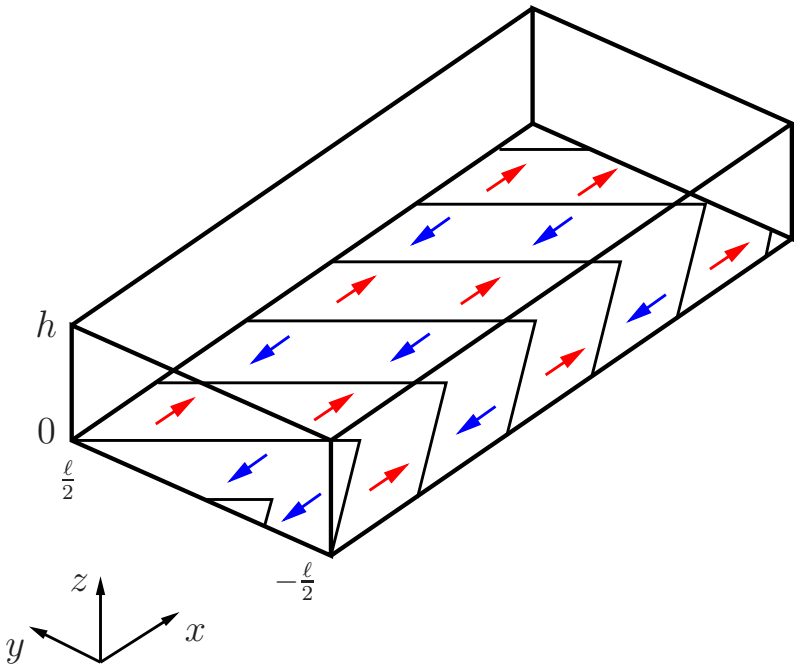


Figure 1.3: A 3D representation of the micromixer. The red arrows represent a forwards flow of velocity U_0 and the blue arrows represent backwards flow of velocity U_0 on the floor of the channel. In addition there is an overall velocity in the x-direction.

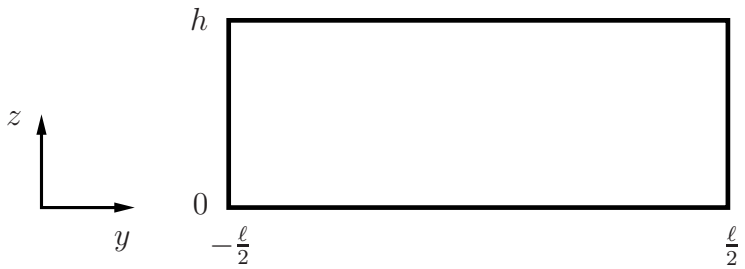


Figure 1.4: Cross-section of the micromixer where the flow direction is out of the page.

1.3 Project Summary

The following are included:

- An analytical solution of the velocity field.
- A program for modelling the flow.
- Trajectory plots.
- Poincaré sections.
- A measure of mixing.
- Dispersion plots.
- Residence times.
- Lyapunov exponents.

Chapter 2

Equation Derivations

2.1 Governing Equations

The governing equations of the fluid motion are the Navier-Stokes equation and the continuity equation,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \nabla^2 \boldsymbol{u}\,, \tag{2.1}$$

$$\nabla \cdot \boldsymbol{u} = 0\,. \tag{2.2}$$

where $\boldsymbol{u} = (u, v, w)$, ρ is constant density and ν is the kinematic viscosity. For our problem the flow is steady due to the low Reynolds number, so $\partial \boldsymbol{u}/\partial t = 0$. Also, for relatively low Reynolds number, for flow in a microchannel with a large ratio of channel length to width, the inertia effects can be neglected [9]. Therefore we neglect $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$. In addition, as ρ and ν are constants, we rescale p such that $p = p/\rho\nu$. The Navier-Stokes equation then reduces to the Stokes equation

$$\nabla p = \nabla^2 \boldsymbol{u}\,, \quad \nabla^2 p = 0\,. \tag{2.3}$$

There are also the following boundary conditions on the flow in the microchannel.

$$u(x, y, 0) = U(x, y)\,, \quad v(x, y, 0) = w(x, y, 0) = 0\,, \tag{2.4a}$$

$$u(x, y, h) = v(x, y, h) = w(x, y, h) = 0\,, \tag{2.4b}$$

$$v\left(x, -\tfrac{\ell}{2}, z\right) = v\left(x, \tfrac{\ell}{2}, z\right) = 0\,, \tag{2.4c}$$

where $U(x, y)$ is the function of x and y which defines the flow on the base of the channel.

These equations satisfy the condition that there is no through flow on the walls of the channel and the no-slip conditions on the floor and ceiling of the channel. However we have not applied no-slip boundary conditions on the sides of the channel, because of the simplification in the model. We impose a function at the base of the channel that is just shifted with respect to a herringbone and not one which is 0 at the sides and hence at $y = -\ell/2$ and $\ell/2$ the velocity u will not be 0. This simplification should not affect the result too much, as the channel has a large ratio between width and height and the effects of not applying no-slip at the side walls will change very little to the flow in most regions of the channel.

2.2 Velocity Field

Due to the low aspect ratio between h and ℓ, 'Lubrication theory' or 'Thin Layer theory' can be applied to (2.3) to obtain equations for u, v and w and hence an approximate analytical solution for velocity field can be found.

We begin by setting $\varepsilon = h/\ell$. As ε is small, the changes in z are small compared to those in x and y. Therefore using Lubrication theory we rescale z such that $z \to \varepsilon z$ and hence $\partial/\partial z \to \varepsilon^{-1}\partial/\partial z$. For the same reason $w \to \varepsilon w$. Thus equation (2.3) becomes

$$u_{xx} + u_{yy} + \frac{1}{\varepsilon^2} u_{zz} = p_x \,, \tag{2.5a}$$

$$v_{xx} + v_{yy} + \frac{1}{\varepsilon^2} v_{zz} = p_y \,, \tag{2.5b}$$

$$\varepsilon \left(w_{xx} + w_{yy} + \frac{1}{\varepsilon^2} w_{zz} \right) = \frac{1}{\varepsilon} p_z \,, \tag{2.5c}$$

and

$$p_{xx} + p_{yy} + \frac{1}{\varepsilon^2} p_{zz} = 0 \,, \tag{2.5d}$$

and (2.2) becomes

$$u_x + v_y + w_z = 0 \,. \tag{2.5e}$$

Note that $w \to \varepsilon w$ has cancelled the ε^{-1} given from $\partial/\partial z$.

We define vertical averaging as

$$\overline{A} = \frac{1}{h} \int_0^h A \, \mathrm{d}z \,. \tag{2.6}$$

From this we find that $\overline{u_x} = \overline{u}_x$, $\overline{v_y} = \overline{v}_y$. Also, $\overline{w_z} = 0$ due to the boundary conditions. Hence $\overline{\nabla \cdot \boldsymbol{u}} = 0$ implies

$$\overline{u}_x + \overline{v}_y = 0\,. \tag{2.7}$$

This means that the vertically averaged velocity is incompressible.

We now assume that $\partial/\partial x$ and $\partial/\partial y$ are order 1. Therefore as $\varepsilon^2 \ll 1$, ε^{-2} is very large and hence equations (2.5a),(2.5b) and (2.5c) reduce to

$$\frac{1}{\varepsilon^2}\, u_{zz} = p_x\,, \tag{2.8a}$$

$$\frac{1}{\varepsilon^2}\, v_{zz} = p_y\,, \tag{2.8b}$$

$$w_{zz} = p_z\,. \tag{2.8c}$$

In order to have a non-zero flow, equations (2.8a) and (2.8b) imply that the pressure must be of order ε^{-2} at leading order, but must not depend on z in order to satisfy (2.8c). Therefore

$$p = \frac{1}{\varepsilon^2} P_0(x,y) + \tilde{p}(x,y,z) \tag{2.9}$$

where $\tilde{p}(x,y,z)$ is of order unity.

Now equations for u, v and w are found. Equations (2.8a), (2.8b) and (2.9) imply:

$$u(x,y,z) = \tfrac{1}{2} P_{0_x} z^2 + a_1(x,y)z + a_0(x,y)\,, \tag{2.10a}$$

$$v(x,y,z) = \tfrac{1}{2} P_{0_y} z^2 + b_1(x,y)z + b_0(x,y)\,. \tag{2.10b}$$

First we solve equation (2.10b). Using the boundary conditions, (2.4a) implies $b_0 = 0$ and (2.4b) implies

$$\tfrac{1}{2} P_{0_y} h^2 + b_1 h = 0 \quad \Longrightarrow \quad b_1 = -\tfrac{1}{2} P_{0_y} h\,.$$

Therefore equation (2.10b) becomes

$$v(x,y,z) = \tfrac{1}{2} P_{0_y} z(z-h)\,. \tag{2.11}$$

To satisfy the boundary condition at the side wall (2.4c), we require

$$P_{0_y}\left(x, \pm\tfrac{l}{2}, z\right) = 0\,. \tag{2.12}$$

Now we solve equation (2.10a). At $z = 0$, (2.4a) implies $u(x, y, 0) = a_0(x, y) = U(x, y)$. At $z = h$, (2.4b) implies

$$u(x, y, h) = \tfrac{1}{2}P_{0_x}h^2 + a_1 h + U = 0 \quad \Longrightarrow \quad a_1 = -\tfrac{1}{h}\left(\tfrac{1}{2}P_{0_x}h^2 + U\right) ,$$

therefore

$$u(x, y, z) = \tfrac{1}{2}P_{0_x}z^2 - \tfrac{1}{h}\left(\tfrac{1}{2}P_{0_x}h^2 + U\right)z + U ,$$

which becomes

$$u(x, y, z) = (h - z)\left(\frac{U}{h} - \frac{P_{0_x}z}{2}\right) . \tag{2.13}$$

We now derive w using u and v by substituting (2.11) and (2.13) into (2.5e) to get

$$w_z = (z - h)\left(\frac{U_x}{h} - \frac{P_{0_{xx}}z}{2}\right) - \frac{1}{2}P_{0_{yy}}z(z - h) .$$

w_z is now integrated w.r.t. z to obtain

$$w = \left(\frac{h}{4}z^2 - \frac{1}{6}z^3\right)\left(P_{0_{xx}} + P_{0_{yy}}\right) + \frac{U_x z^2}{2h} - U_x z , \tag{2.14}$$

where the constant value has been set to 0 due to the boundary condition (2.4a). We now have equations for u, v and w which only depend on x, y, z, P_0 and U.

2.3 Pressure Analysis

In order to use the u, v and w equations, we now need to find a solution for P_0. To solve for P_0 we use the vertically averaged equation (2.7) (i.e. $\bar{u}_x + \bar{v}_y = 0$). The vertical average of v and u can be easily found

$$\bar{v} = \tfrac{1}{2h}P_{0_y}\int_0^h \left(z^2 - hz\right) dz = -\tfrac{1}{12}P_{0_y}h^2 . \tag{2.15}$$

Also

$$u = \frac{U}{h^2}(z - h)(\tau z - h) ,$$

where $\tau = \frac{P_{0_x}h^2}{2U}$ and therefore

$$\bar{u} = \tfrac{1}{h^3}U\int_0^h \left(\tau z^2 - h\tau z - hz + h^2\right) dz = \tfrac{1}{2}U - \tfrac{1}{12}P_{0_x}h^2 . \tag{2.16}$$

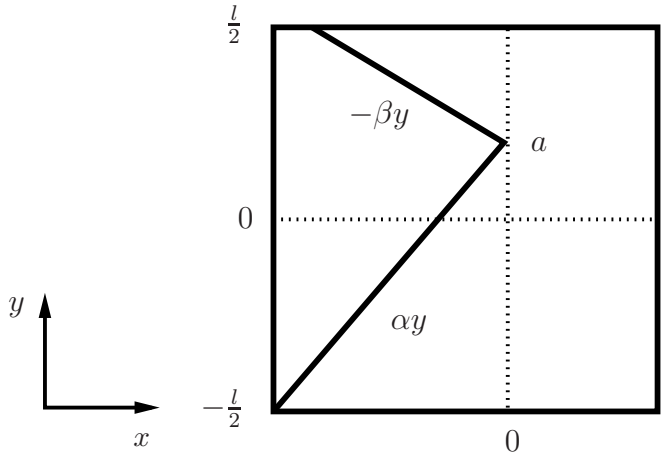


Figure 2.1: A diagram of the herringbone shape with respect to y.

Next we substitute (2.15) and (2.16) into the vertically averaged equation (2.7) to obtain the condition

$$P_{0_{xx}} + P_{0_{yy}} = 6\frac{U_x}{h^2}\,. \tag{2.17}$$

This can now be solved for P_0 and hence analytical solutions can be obtained for u, v and w. However first we simplify w by substituting (2.17) into (2.14) to get

$$w = \left(\tfrac{h}{4}z^2 - \tfrac{1}{6}z^3\right) 6\frac{U_x}{h^2} + \frac{U_x z^2}{2h} - U_x z\,,$$

which reduces to

$$w = -\frac{z}{h^2}\left(h - z\right)^2 U_x\,. \tag{2.18}$$

Hence w does not explicitly depend on P_0.

In order to solve (2.17) for P_0 the specific form of U is needed. Figure 2.1 shows how the herringbone depends on y within the channel. U is different for $y \leq a$ and $y > a$. To represent the herringbone we define a function $\phi(y)$ such that

$$\phi(y) = \begin{cases} \alpha(y-a)\,, & y \leq a; \\ -\beta(y-a)\,, & y > a. \end{cases}$$

Therefore $\phi(y)$ is continuous in y. We now take a Fourier expansion of U in x, varying the phase in y following the herringbone pattern,

$$U = U_0 + \sum_{n=1}^{\infty} U_n^{(\mathrm{s})} \sin(k_n x + \phi(y)) + \sum_{n=1}^{\infty} U_n^{(\mathrm{c})} \cos(k_n x + \phi(y)) \tag{2.19}$$

where $k_n = 2\pi n/L$. This equation is a sum of terms. Because equations (2.5a) to (2.5e) are linear, we now look for a solution for each term, which can then be superimposed once they have been found. U_0 is considered later. We now look at

$$U = U_n \sin(k_n x + \phi(y) + \varphi) \tag{2.20}$$

where $\varphi = 0$ gives us the sin term where $U_n = U_n^{(\mathrm{s})}$ and $\varphi = \pi/2$ gives us the cos term where $U_n = U_n^{(\mathrm{c})}$. $U_n^{(\mathrm{s})}$ and $U_n^{(\mathrm{c})}$ depend on the step function at the base of the channel. We may expand (2.20) as

$$U = U_n \left[\sin(k_n x + \varphi)\cos(\phi(y)) + \cos(k_n x + \varphi)\sin(\phi(y))\right] . \tag{2.21}$$

We further divide this into two terms of the form

$$U = U_n \sin(k_n x + \varphi + \chi)\cos(\phi(y) - \chi) , \tag{2.22}$$

where for $\chi = 0$ we get the 'sin cos' term and for $\chi = \pi/2$ we get the 'cos sin' term in (2.21). Again we will superimpose the solutions. We have thus reduced the problem of solving for the boundary condition (2.19) to four boundary conditions of the form (2.22), with $(\varphi, \chi) = (0, 0)$, $(0, \pi/2)$, $(\pi/2, 0)$, $(\pi/2, \pi/2)$.

Now we let

$$P_0 = \hat{P}(y)\cos(k_n x + \varphi + \chi)$$

We now substitute P_0 and U into equation (2.17) to get

$$\left(\hat{P}''(y) - k_n^2 \hat{P}(y)\right)\cos(k_n x + \varphi + \chi) = \frac{6}{h^2} U_n k_n \cos(k_n x + \varphi + \chi)\cos(\phi(y) - \chi) ,$$

therefore

$$\hat{P}_n''(y) - k_n^2 \hat{P}_n(y) = \frac{6}{h^2} U_n k_n \cos(\phi(y) - \chi) . \tag{2.23}$$

We now define

$$\gamma = \begin{cases} \alpha , & y \le a ; \\ -\beta , & y > a , \end{cases}$$

so that

$$\hat{P}_n''(y) - k_n^2 \hat{P}_n(y) = \frac{6}{h^2} U_n k_n \cos(\gamma(y - a) - \chi) . \tag{2.24}$$

We look for a particular solution of the form

$$Q(y,\gamma,\chi) = q\cos(\gamma(y-a)-\chi)\,.$$

With the choice $\hat{P}_n = Q$, equation (2.24) becomes

$$-\gamma^2 q\cos(\gamma(y-a)-\chi) - k_n^2 q\cos(\gamma(y-a)-\chi) = \frac{6}{h^2}U_n k_n\cos(\gamma(y-a)-\chi)\,,$$

which implies

$$q = \frac{-6U_n k_n}{h^2(\gamma^2+k_n^2)}\,.$$

Therefore

$$Q(y,\gamma,\chi) = \frac{-6U_n k_n\cos(\gamma(y-a)-\chi)}{h^2(\gamma^2+k_n^2)}\,. \tag{2.25}$$

Additionally, the complementary solution for $\hat{P}(y)$ is

$$\hat{P}(y) = A\cosh(k_n y) + B\sinh(k_n y)\,,$$

so the solutions for $\hat{P}(y)$ when $y \le a$ and $y > a$ are

$$\hat{P}_{\mathrm{I}} = A_{\mathrm{I}}\cosh(k_n y) + B_{\mathrm{I}}\sinh(k_n y) + Q(y,\alpha,\chi)\,, \qquad y \le a\,, \tag{2.26a}$$

$$\hat{P}_{\mathrm{II}} = A_{\mathrm{II}}\cosh(k_n y) + B_{\mathrm{II}}\sinh(k_n y) + Q(y,-\beta,\chi)\,, \quad y > a\,. \tag{2.26b}$$

There are 4 χ-dependent constants to find; A_{I}, A_{II}, B_{I} and B_{II}. These constants are independent of φ. There are four conditions that we need to satisfy to find the constants

$$P_{\mathrm{I}}'(-\frac{\ell}{2}) = 0 \tag{2.27a}$$

$$P_{\mathrm{II}}'(\frac{\ell}{2}) = 0 \tag{2.27b}$$

$$P_{\mathrm{I}}(a) = P_{\mathrm{II}}(a) \tag{2.27c}$$

$$P_{\mathrm{I}}'(a) = P_{\mathrm{II}}'(a) \tag{2.27d}$$

These conditions ensure that the pressure is continuous over the microchannel and that there is no through flow at $y = -\ell/2$ and $\ell/2$.

We now only need to solve for the four constants to use our equations for u, v and w, which we solve for using Maple. The worksheet can be seen in Appendix A.1.

2.4 Pressure Gradient and Constant Velocity Solution

In order for there to be an overall mean flow in the positive x-direction, there must be either a pressure gradient or a nonzero mean velocity induced on the floor of the channel, in addition to the herringbone induced velocities. On a global scale the herringbone velocities average out because for every forward flow herringbone there is an equal and opposite backwards flow herringbone due to the step function imposed. Having a constant velocity on the base of the channel is equivalent to having a moving channel floor, which is only feasible for electro-osmotically driven flow. This would be impossible with a grooved floor which would need to be pressure driven.

The U_0 term in the expansion for U is the constant velocity. The pressure gradient would only be in the x-direction. Therefore we define the pressure for pressure driven flow as $P_0 = -rx$ where r is positive. This will create a flow in the positive x-direction as the pressure gradient $(-r)$ is negative, so the pressure reduces as x increases and hence fluid will flow from high to low pressure areas.

$P_0 = -rx$ and $U = U_0$ which implies $P_{0_x} = -r$, $P_{0_y} = 0$, $P_{0_{xx}} = 0$ and $U_x = 0$. Therefore the pressure equation (2.17) holds and the equations for v and w ((2.11) and (2.18)) reduce to $v = 0$ and $w = 0$. However the equation for u (2.13) becomes

$$u = \frac{(h-z)}{h}U_0 + \frac{(h-z)z}{2}r\,.$$

The velocity (u) generated by electro-osmosis is linear in z (Couette flow) whereas the constant pressure gradient flow is quadratic in z (Poiseuille flow). For small z, this implies that the velocity generated by the constant electro-osmosis is much greater than pressure-driven flow. From a manufacturing stand-point the requirement for a high pressure to drive the flow is undesirable and electro-osmosis is much easier and effective to use. As stated in the introduction, this project will not look at pressure-driven flows. We set $r = 0$ and therefore the global flow is created by U_0 only. However, it is useful to know that the model can easily be changed to model pressure driven flow by setting $U_0 = 0$ and r to the non-zero value of the pressure gradient. We label the flow in the x-direction due to U_0 as u_e:

$$u_e = \frac{(h-z)}{h}U_0 \tag{2.28}$$

This equation will be superimposed with the solution found for u due to the herringbones.

2.5 Herringbone Step Function and Gibbs Phenomenon

Once the solution for U has been found, we still need solutions for values for $U_k^{(c)}$ and $U_k^{(s)}$. U_0 can just be defined as a constant of our choice, which is the speed of the flow (i.e. 1=100 μm/s or 5=500 μm/s etc.). U_0 is the overall driving force of the channel and we don't want it to overpower the step function at the base of the channel as this would not be good for mixing. Therefore we define our step function relative to U_0,

$$\text{step}(x) = \begin{cases} S_F U_0\,, & 0 < x < \frac{L}{2}\,; \\ -S_F U_0\,, & \frac{L}{2} < x < L\,, \end{cases}$$

where S_F is the "step function value" (i.e. the ratio between U_0 and the step function velocity). Unless otherwise stated, $S_F = 1$ throughout this project. We are more interested in setting $S_F = 1$ than any other value as this is good for practical manufacturing purposes. The negative velocity herringbones will exactly cancel out U_0, so when making the channels, the coating used to create the electro-osmotically induced flow would need to be placed on forwards herringbones and the backwards herringbones wouldn't need coating. For $U_0 = 1$ the backwards herringbones would induce no velocity whereas the forwards herringbones would induce a velocity of 2 (i.e. 200μm/s). For $S_F \neq 1$ the manufacturing is more complicated and expensive, with different amounts of coating being required on forwards and backwards herringbones.

$U_k^{(c)}$ and $U_k^{(s)}$ are just the Fourier coefficients

$$\begin{aligned} U_n^{(c)} &= \frac{1}{L}\int_0^L \cos\left(k_n x\right)\text{step}(x)\mathrm{d}x \\ U_n^{(s)} &= \frac{2}{L}\int_0^L \sin\left(k_n x\right)\text{step}(x)\mathrm{d}x \end{aligned}$$

where $k_n = 2\pi n/L$ and step(x) is the step function. Since cosine is even and step(x) is odd over the interval $[0, L]$, $U_n^{(c)} = 0$ for all k_n. However $U_n^{(s)}$ becomes

$$\begin{aligned} U_{k_n}^{(s)} &= S_F U_0 \frac{2}{L}\int_0^{\frac{L}{2}} \sin(k_n x)\mathrm{d}x - S_F U_0 \frac{2}{L}\int_{\frac{L}{2}}^L \sin(k_n x)\mathrm{d}x \\ &= S_F U_0 \frac{4}{L}\int_0^{\frac{L}{2}} \sin(k_n x)\mathrm{d}x = S_F U_0 \frac{4}{L}\frac{1}{k_n}\left[-\cos(k_n x)\right]_0^{\frac{L}{2}} \\ &= S_F U_0 \frac{2}{\pi n}\left[-(-1)^n + 1\right]\,. \end{aligned}$$

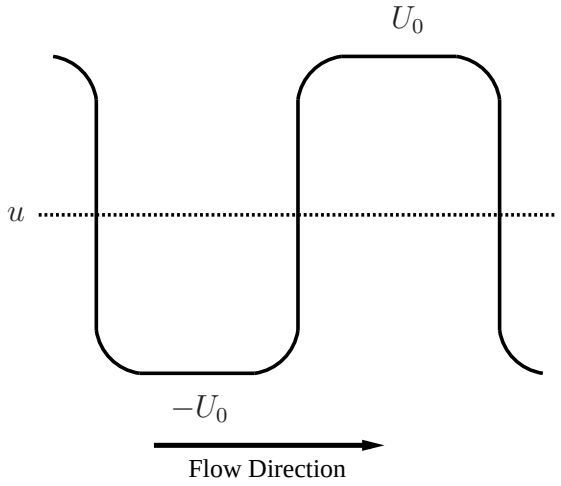


Figure 2.2: Adapted step function with smoothing parameter.

Therefore

$$U_n^{(s)} = \begin{cases} 0\,, & n \text{ even}\,; \\ \frac{4}{\pi n} S_F U_0\,, & n \text{ odd}\,. \end{cases} \tag{2.29}$$

Therefore the only nonzero values for $U_n^{(s)}$ occur when n is odd. Hence, there is no need to solve for P_0 for $\varphi = \frac{\pi}{2}$ as all cos terms are 0.

As we have applied a Fourier transform to a step function, we will experience large oscillations in the truncated Fourier series near to discontinuities in the step function. This is called the Gibbs Phenomenon. To avoid this, we force the Fourier solutions for large n to decay to 0 quickly by imposing an exponential damping factor on the solution. For odd n we let

$$U_n^{(s)} = \frac{4}{\pi n} S_F U_0 \mathrm{e}^{-\lambda(n-1)^2}\,, \tag{2.30}$$

where λ is a smoothing parameter. We do not want λ to be too large as this will cause too many high modes to be approximately 0 and the flow on the floor of the channel will be like a sine wave rather than a step function. However, λ should be large enough to stop the Gibbs phenomenon from occurring. The value of λ depends on the desired number of Fourier modes we would like to use and is determined from numerical experiments later in the project. The correct choice of λ will cause the step function to stay very similar but have rounded edges (see figure 2.2).

Chapter 3

Program Creation

In this section the steps taken to create the program are briefly explained. In addition tests are performed to ensure that our solutions are correct and continuous.

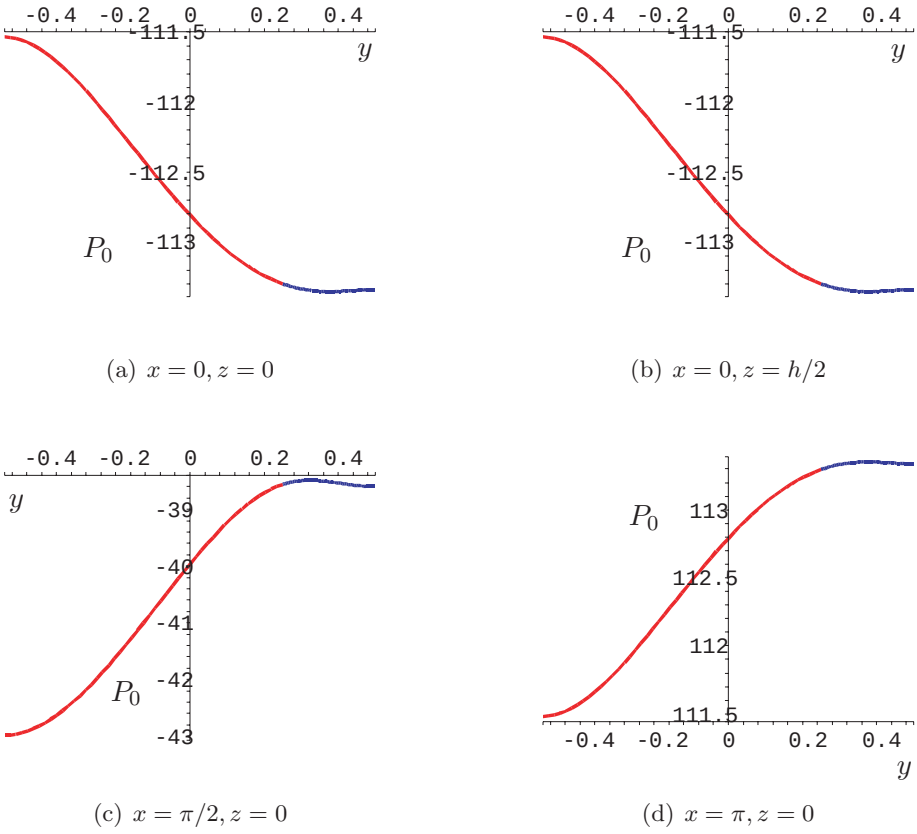
3.1 Preliminary Calculations

Before we created the program, we solved for the constants using Maple. We also performed the same calculations in Mathematica to confirm the results. We then used the constants in a program created in C++ along with our equations for u, v, w and P_0. The Maple sheet can be seen in appendix A.1. In addition to solving for the constants we also verify that the conditions (2.4) and (2.27) are satisfied.

The equations are dependent on the configuration of the channel. As stated in the introduction, in this project we always use $\ell = 1$ and $L = 2\pi$. This implies that from now on $k_n = n$. For this test problem we just use $n = 1$ (i.e. the first Fourier mode) and therefore $U_1 = 4/\pi$. As a result of only 1 mode being used, the base of the channel has a sine wave rather than a step function in line with the herringbones. The other parameters have been set to $\alpha = 1$, $\beta = 2$, $a = 0.25$ and $h = 0.25$.

In order to verify that (2.4) and (2.27) are satisfied, we plotted graphs of u, v, w and P_0 over the width of the channel for different values of x and z. The code can again be seen in appendix A.1. In figures 3.1 to 3.5 are plots of P_0, P_0', u, v and w against y for different values of x and z. The red line is for $y = -\ell/2$ to a and the blue line is for $y = a$ to $\ell/2$.

From all of the plots we can see that P_0, P_0', u, v and w are continuous from $y = -\ell/2$ to $y = \ell/2$. In figure 3.1 we can see for $x = 0$ that the graph for P_0 is the same for $z = 0$ and $z = h/2$.



(a) $x = 0, z = 0$

(b) $x = 0, z = h/2$

(c) $x = \pi/2, z = 0$

(d) $x = \pi, z = 0$

Figure 3.1: Plots of P_0 across the width of the channel at different positions in x and z

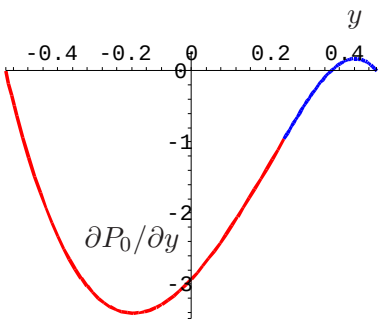
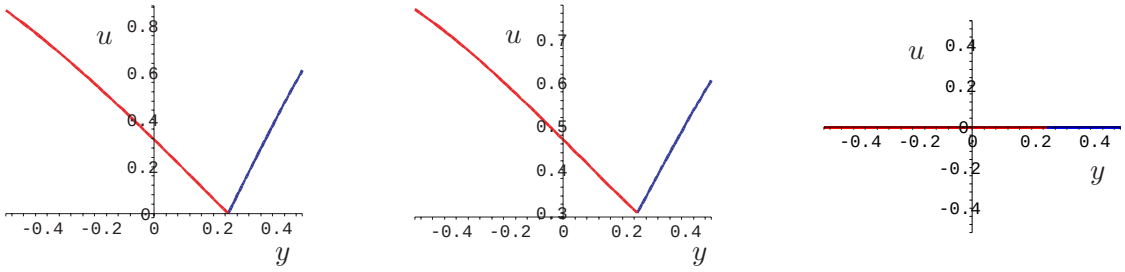
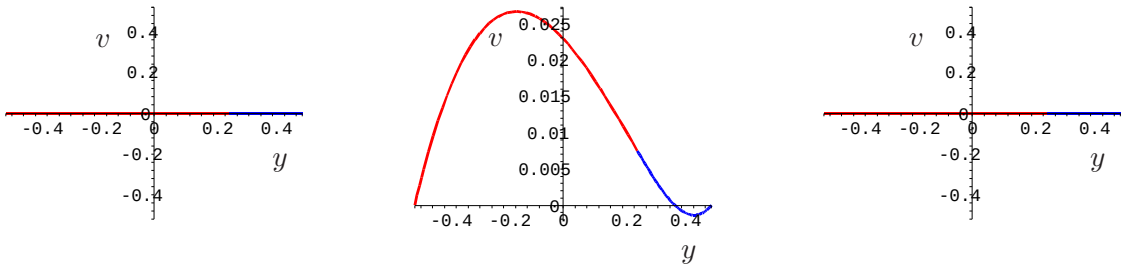


Figure 3.2: Plot of $\partial P_0/\partial y$ across the width of the channel at $x = 0$ and $z \in [0, h]$.



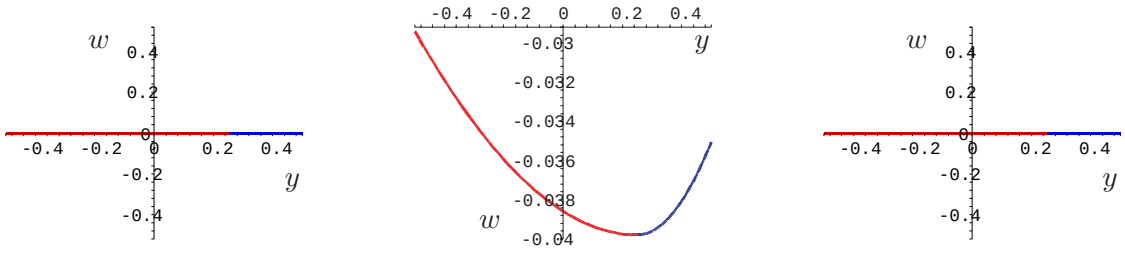
(a) $x = 0, z = 0$ (b) $x = 0, z = h/2$ (c) $x = 0, z = h$

Figure 3.3: Plots of u across the width of the channel at different positions in x and z



(a) $x = 0, z = 0$ (b) $x = 0, z = h/2$ (c) $x = 0, z = h$

Figure 3.4: Plots of v across the width of the channel at different positions in x and z



(a) $x = 0, z = 0$ (b) $x = 0, z = h/2$ (c) $x = 0, z = h$

Figure 3.5: Plots of w across the width of the channel at different positions in x and z

In fact for any given x value, P_0 is the same for all z. This is as expected as P_0 doesn't depend on z. Figure 3.1 also shows plots for $z = 0$ and $x = \pi/2$ and $x = \pi$. For $x = 0$ and $\pi/2$, P_0 is negative, whereas for $x = \pi$, P_0 is positive. This suggests that for this flow, there would probably be a change of flow direction from $x = 0$ and $\pi/2$ to $x = \pi$. This again is as expected since we imposed a sin wave on the floor of the channel with the period 2π.

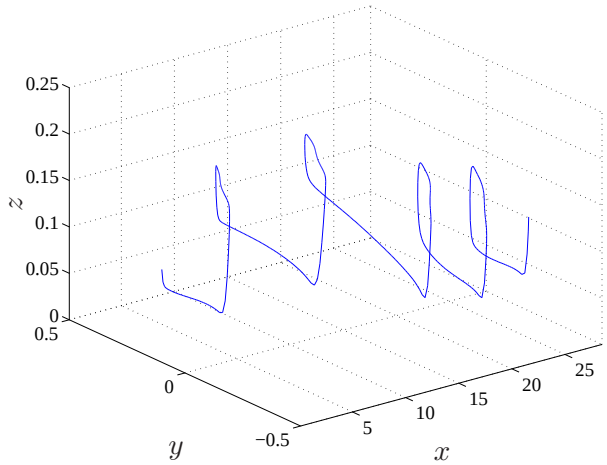
Figure 3.2 shows that P_0' is continuous over y. Also $P_0'(-\ell/2) = P_0'(\ell/2) = 0$. Therefore the conditions (2.27) are all satisfied. In addition for figures 3.3, 3.4 and 3.5 we can see that $v = w = 0$ at the base of the channel, $u = v = w = 0$ at the ceiling of the channel and $v = 0$ at the side walls of the channel. In addition figure 3.3 shows that at the floor of the channel at $x = 0$, u is a function of y. From these results it is clear that the boundary conditions (2.4) are all satisfied. Also from the plots we can see that u, v and w all vary between $z = 0$ and h which is as we hoped, otherwise the flow would be very simple.

3.2 Velocity Field Code

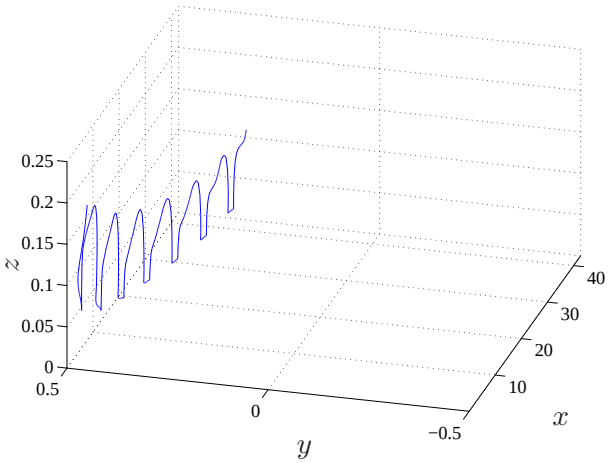
In order to analyse the flow we created a program in C++ for integrating the system from a starting position for the length of time of our choice. We created a function which contains the velocity field solution. This contains the equations for the constants and the equations for u, v and w. This function can be seen in appendix B.1 and the file is called `velmmix.hpp`.

In section 4 there are programs which use `velmmix.hpp` to obtain different results which we can use to analyse mixing properties. First we test the the velocity field by integrating the the system using the adaptive Runge-Kutta Cash-Karp method from a few different starting positions to obtain some particle trajectories. The code is called `velmmix_particle.cpp` and can be seen in appendix B.3.

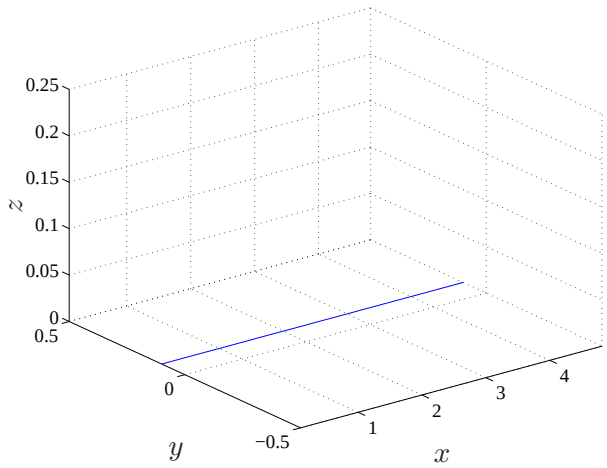
In figure 3.6 there are four different trajectories for different starting positions. We have used the same herringbone configuration as in section 3.1. However, instead of using just one Fourier mode, we have used fifty Fourier modes, so we have a step function (refer to section 3.3 for more information about the configuration of the step function). Also there is now a mean flow down the channel as the U_0 term has been superimposed on the u velocity. In 3.6(a) is a random trajectory of a particle which appears to be spiraling as it moves along the channel. In 3.6(b) we see a trajectory of a particle which starts close to the side wall. This particle stays close to the wall as there is no flow through the wall, however it does slowly move towards the middle of the channel. 3.6(c) is the trajectory of a particle which starts on the floor. As there is no v or w velocity, the particle



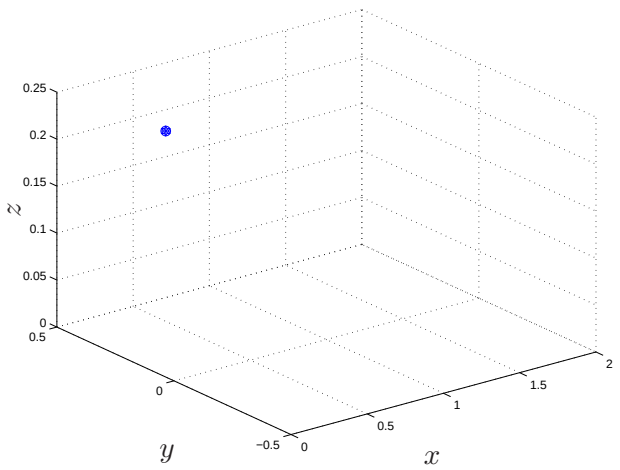
(a) $x = 0.1$, $y = 0.1$, $z = 0.1$



(b) $x = 0.1$, $y = 0.45$, $z = 0.2$



(c) $x = 0.1$, $y = 0.1$, $z = 0$



(d) $x = 0.1$, $y = 0.1$, $z = 0.25$

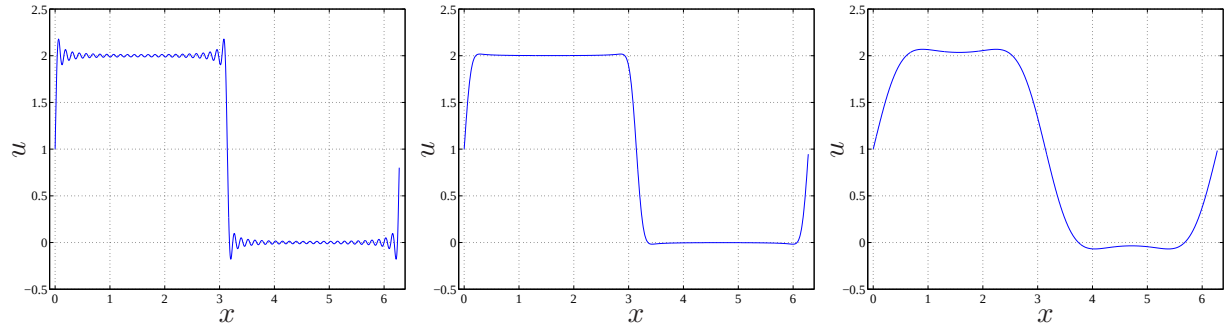
Figure 3.6: Different trajectories for different starting positions in a channel with configuration $\alpha = 1$, $\beta = 2$, $a = 0.25$, h=0.25, $U_0 = 1$ for 50 Fourier modes. The figures are labelled with the starting position of the particle.

just travels in a straight line in the x-direction, staying on the floor of the channel. Finally 3.6(d) shows that a particle which begins on the ceiling of the channel does not move at all, which satisfies no-slip on the ceiling.

From all four plots in figure 3.6, the trajectories are behaving as they should be near to the boundaries and no trajectories leave the channel. In addition a particle which starts not too close to a boundary moves with a circular motion down the channel which suggests that there is some "swirling" motion occurring. These tests suggest that the velocity field function is working correctly and we can now continue to study some properties of the flow.

3.3 λ values

In section 2.5 the Gibbs phenomenon was discussed. We will now find the λ values for different numbers of Fourier modes so that there is a smooth step function on the floor of the channel. We created a program called `velmmix_test.hpp` which can be seen in Appendix B.2. This program outputs the velocity along the floor of the channel for the length $L = 2\pi$.



(a) $\lambda = 0$ (b) $\lambda = 0.005$ (c) $\lambda = 0.1$

Figure 3.7: Plots of the step function from $x = 0$ to 2π for different λ values with 50 modes used and $U_0 = 1$.

In figure 3.7 we see how the step function varies depending on the λ value. In plot (a) the λ value is too small so large oscillations occur near the discontinuities, whereas plot (c) has the value too high so the function is more like a sine wave than a step function. Plot (b) has a good λ value and it is an acceptable step function for the fifty Fourier mode system.

Throughout the analysis in the next few sections we will use different numbers of Fourier modes depending on the accuracy and the desired length of time for the calculation to be performed. As

Modes	λ
1	N/A
5	0.2
10	0.035
20	0.01
50	0.005

Table 3.1: λ values for $U_0 = 1$ and $S_F = 1$ obtained by using `velmmix_test.hpp`.

a rough estimate, a calculation using 10 modes will take 10 times longer than a 1 mode calculation. Table 3.1 shows some good λ values for different numbers of Fourier modes, when $U_0 = 1$ and $S_F = 1$.

Figure 3.8 shows how better step functions can be obtained by using more Fourier modes. For 1 mode it doesn't matter what value of λ is used as not enough modes are being used to create a step function. We will try to use 50 modes for most calculations in the project, however for some of the longer calculations, a lower number is required as the calculation would take too long with 50 modes. In figure 3.8 we see that for 20 modes the step function is not much worse than for 50 modes, so we can drop the number of modes without losing too much accuracy.

λ values have not been stated for when either U_0 or S_F are not 1. However the test program can be used for any combination of U_0 and S_F. For all cases in the project where U_0 or S_F isn't 1, an appropriate value for λ has been used in the calculations.

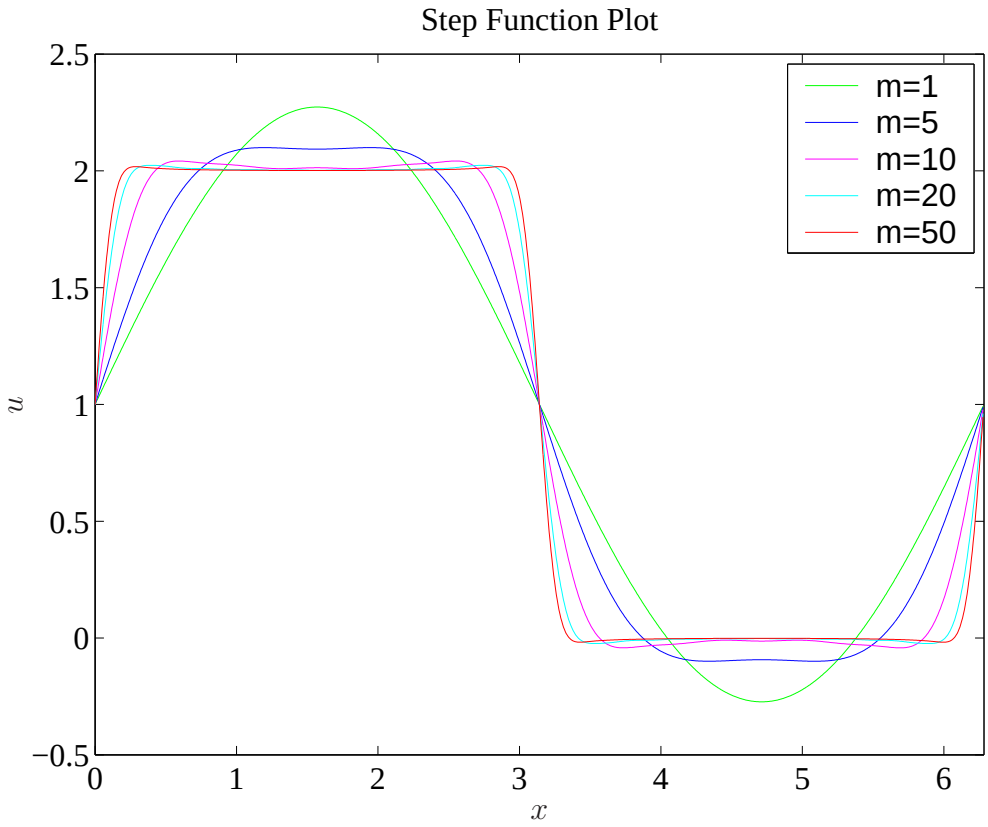


Figure 3.8: Plot of step functions for different numbers of Fourier modes with λ values used from table 3.1. m = number of Fourier modes.

Chapter 4

Mixing Analysis

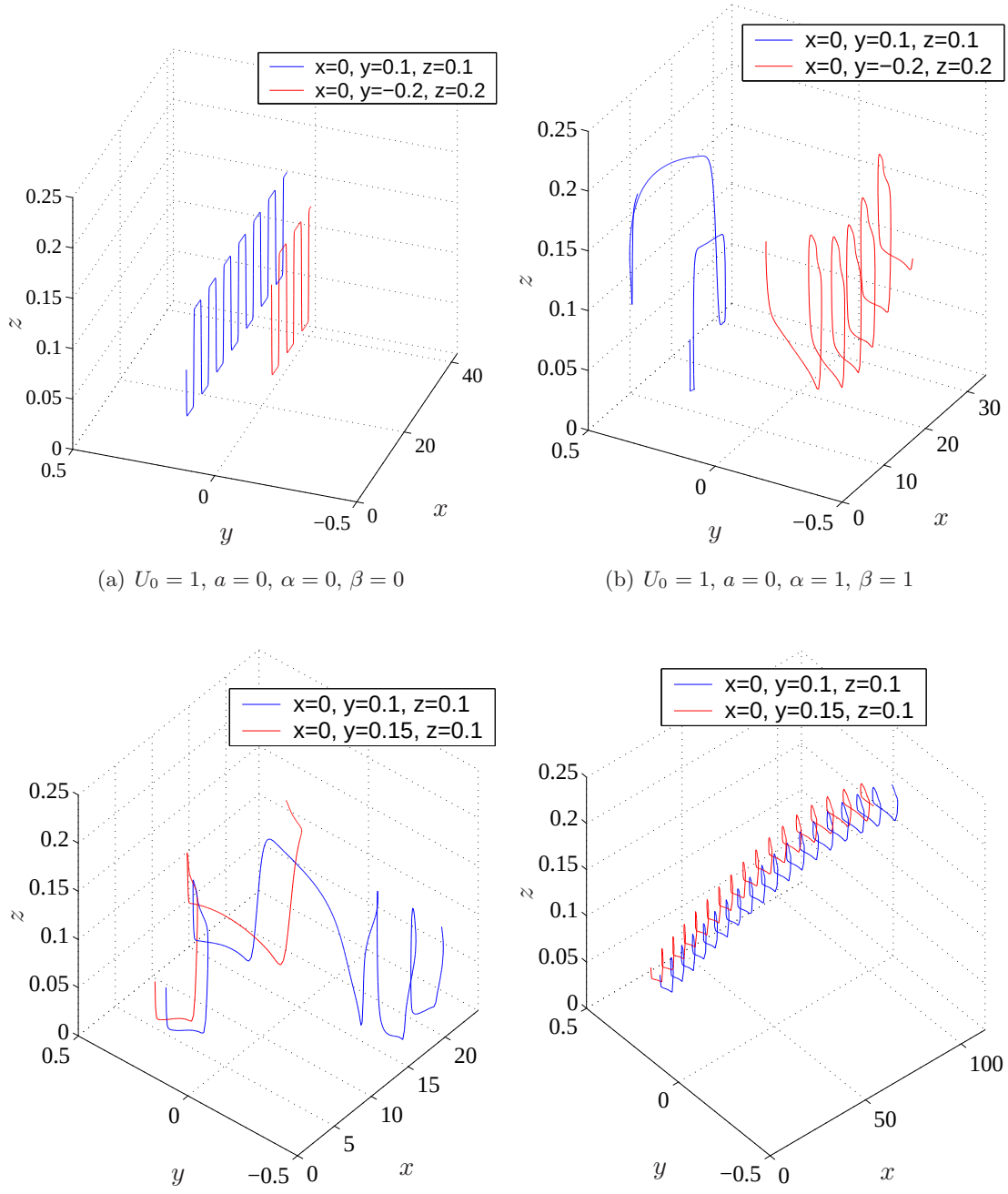
In this section we look at some different flows for different configurations of channels. The aim is to try to find some good mixing configurations. However, it is very difficult to define good mixing. Ideally mixing will occur in the cross-section of the channel (i.e. lateral mixing), but dispersion will also occur longitudinally in the channel which we would like to keep to a minimum to avoid longitudinal spreading. We look at some different properties of various configurations and highlight the signs of chaotic flows which is needed for good mixing. We will first look at some different trajectories and then study Poincaré sections in some detail before studying dispersion within the channels.

4.1 Trajectories

First we look at a few different trajectories of particles for different channel configurations. We will look at configurations with $h = 0.25$, and a few different combinations of U_0, α, β and a. An extensive study of different configurations is performed later in section 4. To obtain the different trajectories we used `velmmix_particle.cpp` which can be seen in appendix B.3.

A sign of good mixing is chaotic trajectories of particles. If two particles begin near to each other, diverging paths is a possible sign of chaos. However if two particles never get close to each other, there are possibly two regions in the fluid which never mix. Thus we look for chaotic trajectories that cover the whole channel when looking at trajectory plots.

Figure 4.1 has four plots of trajectories for different configurations. Figure 4.1(a) is an example of a very poor mixer. The configuration is $\alpha = 0$ and $\beta = 0$, so this is the step function which is positioned in straight lines rather than a herringbone pattern, and hence there is no lateral flow.



(a) $U_0 = 1$, $a = 0$, $\alpha = 0$, $\beta = 0$

(b) $U_0 = 1$, $a = 0$, $\alpha = 1$, $\beta = 1$

(c) $U_0 = 1$, $a = 0.25$, $\alpha = 1$, $\beta = 2$

(d) $U_0 = 5$, $a = 0.25$, $\alpha = 1$, $\beta = 2$ with $S_F = 0.2$

Figure 4.1: Plots of some trajectories for some different channel configurations. The legend states the starting positions of each trajectory.

The two particles do not interact at all as they only move in the x and z direction. The blue particle moves further down the channel than the red particle in the same length of time. This is because the red particle begins at a higher point in the channel and due to the driving force being generated on the floor of the channel, particles are generally moving faster in the bottom half of the channel than the top which has no-slip boundary conditions on the ceiling.

Figure 4.1(b) shows two trajectories for the configuration $U_0 = 1$, $a = 0$, $\alpha = 1$, $\beta = 1$. The trajectories are a lot more chaotic than those in figure 4.1(a). The particles move in completely different patterns, but the trajectories seem to be in different regions. It is hard to tell whether these particles would ever get close, so this channel may not be a good mixer.

Figure 4.1(c) shows the configuration $U_0 = 1$, $a = 0.25$, $\alpha = 1$, $\beta = 2$ and has the trajectories of two particles which begin close to each other. These two particles initially follow similar trajectories but then deviate a long way from each other. It is possible that this channel has chaos occurring without there being any large regions that do not mix. Thus this channel potentially may be a good mixer.

Finally figure 4.1(d) shows the same configuration as figure 4.1(c) but with $U_0 = 5$ and $S_F = 0.2$. The particles move much faster down the channel than for when $U_0 = 1$ and only have the same velocity induced by the herringbones as in figure 4.1(c). The particles do not deviate away from each other, like for when $U_0 = 1$. This is because the faster mean flow generated when $U_0 = 5$ dominates over the herringbone velocities, so any lateral forces in the y-direction have less effect on the trajectory of the particle. However for $S_F = 1$ the herringbones will not be overpowered, so the trajectories are like 4.1(c), but with $U_0 = 5$.

From the four plots in figure 4.1 it is very difficult to get any conclusive information about the amount of mixing that the different configurations provide. The channel in 4.1(a) is a poor mixer and the bilateral movement of the particles in both 4.1(b) and 4.1(c) will result in better mixing than 4.1(a), where the particle trajectories are purely unilateral. From the diagrams alone, there is no real method for making a quantifiable comparison between 4.1(b) and 4.1(c). Thus this provides the motivation to study different ways of measuring mixing.

4.2 Poincaré Sections

A popular and useful method to get an idea of mixing is to take Poincaré sections of the flow. The idea of Poincaré sections is to make use of the periodicity of configuration of the channel [8]. That is, if we start two particles with with the same y and z coordinates but with one at $x = 0$ and one

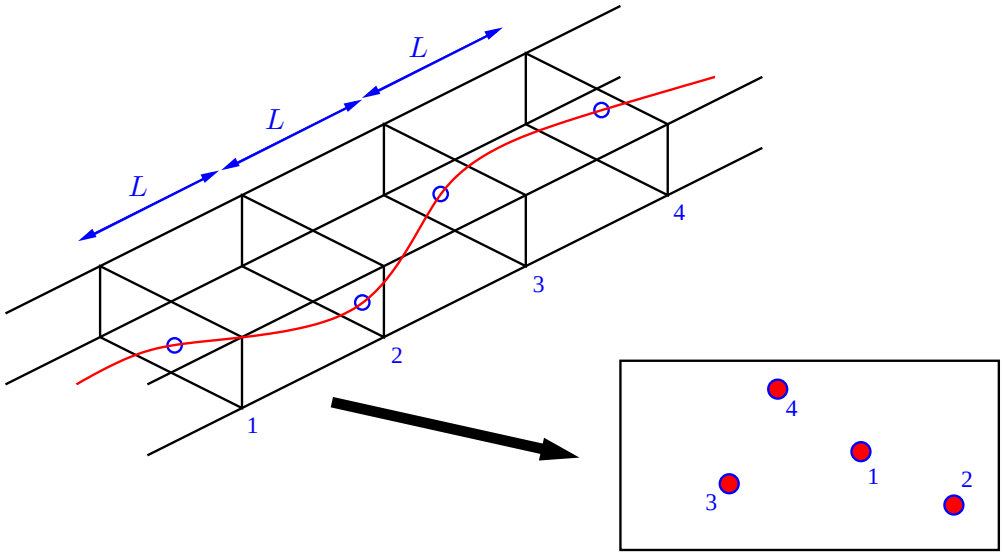
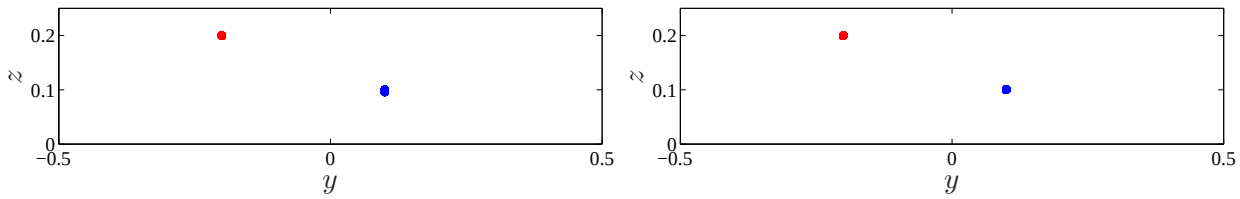


Figure 4.2: Diagram of how the trajectory maps to a Poincaré section.

at $x = L$, the particles will follow exactly the same trajectory except the x coordinates will differ by L. The same follows for any multiple of L and any initial reference point. i.e. a trajectory will be the same if starting at $x = x_0$ or $x = x_0 + L$ or $x = x_0 + 2L$ etc. A Poincaré section is a plot of where the trajectory has passed through the cross-section of the flow at $x = x_0 + nL$ for a set number of periods, n say (i.e figure 4.2). Modelling one particle for many periods is the same as modelling many particles within the same region of the flow for just a few periods. However by modelling one particle we are guaranteed that the particle will stay in one region of the flow.

The more periods we let the particle travel, the more points there are in the Poincaré section. If the Poincaré section has many points for one trajectory spreading over most of the section then it implies that the mixer will be good. However there may be regions or loops which the particle never leaves, and this is not good for mixing. Poincaré sections can include more than one trajectory, with each trajectory represented by different coloured points on the section. Therefore a good idea of how particles cross through the section can be obtained by taking a Poincaré section of a few different trajectories.

We now look at some of the Poincaré sections for the same configurations that we plotted trajectories for in section 4.1. In Appendix B.4 is the C++ code we created for finding the Poincaré sections. It is very similar to the trajectory code, however it stores the position of the particle at the selected sections. First we only look at sections at $x = nL$ and $x = L/2 + nL$.

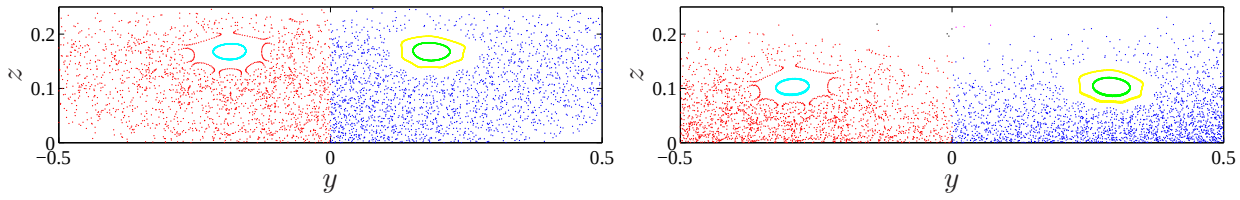


(a) $x = 0$ (b) $x = \frac{L}{2}$

Figure 4.3: Poincaré sections for channel: $U_0 = 1$, $a = 0$, $\alpha = 0$, $\beta = 0$

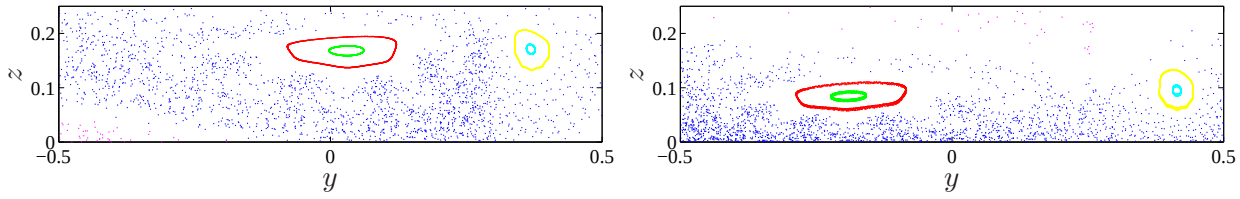
Figures 4.3 to 4.6 are for the Poincaré sections for the same configurations as in figure 4.1. All the Poincaré sections are for 2000 periods of the channel and on the plots, forwards flow is towards the reader. Figure 4.3 has the section generated by the two different particles observed in figure 4.1(a). This plot confirms our observations from the 3D trajectory plot in 4.1(a) that the particles are not laterally moving around the channel. In fact, as there are only two small dots (one blue and one red) the particles are in the same position after every period, so there isn't even any vertical change after a period. This is a very poor mixer.

Figure 4.4 is more interesting. In figure 4.1(b) it is difficult to tell if the channel is a good mixer and how the particles interact. Figure 4.4 makes the situation a lot clearer. Each colour represents one trajectory. There are two clear regions that do not interact on this flow. At $x = 0$ there is a blue side and a red side of the flow. Both of these particles represent forward crossings of the section. The red and blue particles never cross over $y = 0$. However each particle does cover most of its half of the channel. Therefore there are two reasonably well mixed halves of the channel. Also, there are two regions where particles are stuck on loops that they never leave. Very little mixing will occur in these regions. At $x = L/2$ the situation is very similar: the loops have moved slightly but these loops will move around through the channel during each period. The main difference here is that at the top of the channel there are some black and magenta dots which represent the backwards crossings through the cross section at $x = L/2$ of the red and blue particles respectively. There is no backwards flow at $x = 0$. This makes sense because if we refer back to figure 3.1 the pressure is a different sign at $x = 0$ to $x = L/2$. Also we know that at $L/2$ the step function changes from 1 to -1. There is a mean forward flow on the base of the channel at $x = 0$ whereas at $x = L/2$ there is no mean flow on the base of the channel. Therefore it makes sense that there is only backwards flow at $x = L/2$. In addition there are a lot less backwards flow dots than there are forwards flow dots, so the top of the channel does look relatively empty. This is because the mean flow in the middle of the channel is still positive at $x = L/2$ so as the trajectories travel



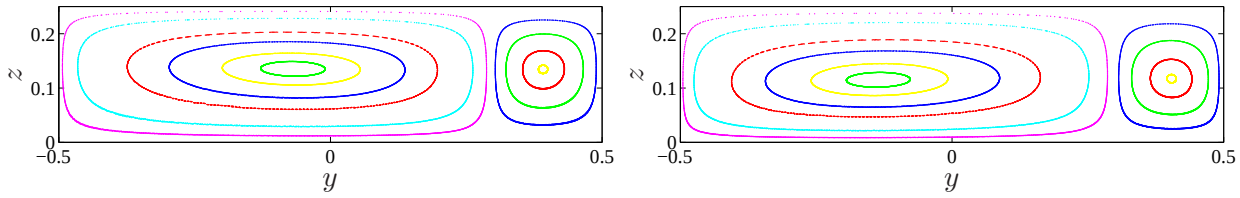
(a) $x = 0$ (b) $x = \frac{L}{2}$

Figure 4.4: Poincaré sections for channel: $U_0 = 1$, $a = 0$, $\alpha = 1$, $\beta = 1$



(a) $x = 0$ (b) $x = \frac{L}{2}$

Figure 4.5: Poincaré sections for channel: $U_0 = 1$, $a = 0.25$, $\alpha = 1$, $\beta = 2$.



(a) $x = 0$ (b) $x = \frac{L}{2}$

Figure 4.6: Poincaré sections for channel: $U_0 = 5$, $a = 0.25$, $\alpha = 1$, $\beta = 2$ with a step function value $S_F = 0.2$. Note however that for $S_F = 1$, the section looks very similar to that in figure 4.5.

through the channel there will be a lot less occasions when they cross the section backwards than when they cross forwards.

Figure 4.5 is definitely the best mixer of the group. This is for the same configuration as for figure 4.1(c). The blue regions cover most of the channel at both $x = 0$ and $x = L/2$. The magenta coloured dots are the backwards crossing of the blue trajectories, so the one trajectory covers all of the channel with the exception of two loop regions where little mixing will occur. Again there is a larger backwards flow region at $x = L/2$ than $x = 0$, however there is some backwards flow at $x = 0$. Overall this configuration appears to be quite a good mixer. There are only two small regions where mixing does not occur. It is these regions that we need to minimise or preferably eliminate completely to find a good mixer. However, for the moment the configuration $a = 0.25$, $U_0 = 1$, $\alpha = 1$, $\beta = 2$ is the best mixer.

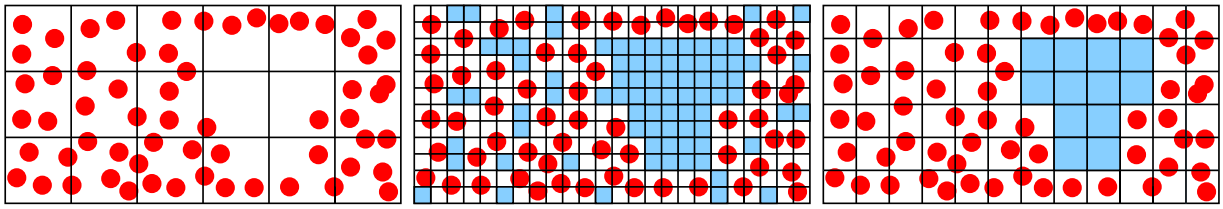
In figure 4.6 is the same configuration as in figure 4.5 except $U_0 = 5$ and $S_F = 0.2$ like in figure 4.1(d). As we expected this has quite bad consequences on the flow as all the particles are stuck on loops so virtually no mixing will occur. However for $S_F = 1$, the flow looks almost identical to figure 4.5. This implies that for an increased velocity the flow doesn't change very much provided $S_F = 1$. As the flow for $U_0 = 5$ is similar to the flow for $U_0 = 1$, we investigate this later once we have found the best mixer for $U_0 = 1$.

Finally, on the CD there is an animation of a flow with configuration $\alpha = 1$, $\beta = 1$, $a = 0.25$ and $U_0 = 1$. Open `animation.html` on the CD to get a web page with a brief explanation of the animation. This animation shows how the "loop" regions move through the flow during each period. This flow is similar to that in figure 4.5 and the configurations are very similar expect β differs. We need a way of measuring mixing to decide which of these two configurations provides the best mixing.

4.3 A Measure of Mixing

Poincaré sections have enabled us to highlight the best configuration of the four channels that we looked at in section 4.1. However we do not know if this configuration is better than that on the CD and there are many different configurations of channels that we can study. The problem is that it is highly impractical to physically look at many different channels and it would be very difficult to highlight the best channel with the human eye. Therefore we devised a method for measuring mixing; similar ideas have been used in other papers (e.g.[5]) although none that we are aware of have actually used this exact same method. The idea is to first make a Poincaré section of one

trajectory; we then split the section into a grid, and count the number of squares containing one or more dots. Therefore areas with large regions of poor mixing will have a lower score than areas with small regions. A system like that in figure 4.4 will have a low score because for only one particle, only half of the channel will be covered. This is quite a simple idea but the score does depend on the size of the grid we use. Too coarse a grid will result in areas of poor mixing not being caught, and too fine a grid will result in areas of good mixing being mistaken for areas of poor mixing (see figure 4.7).



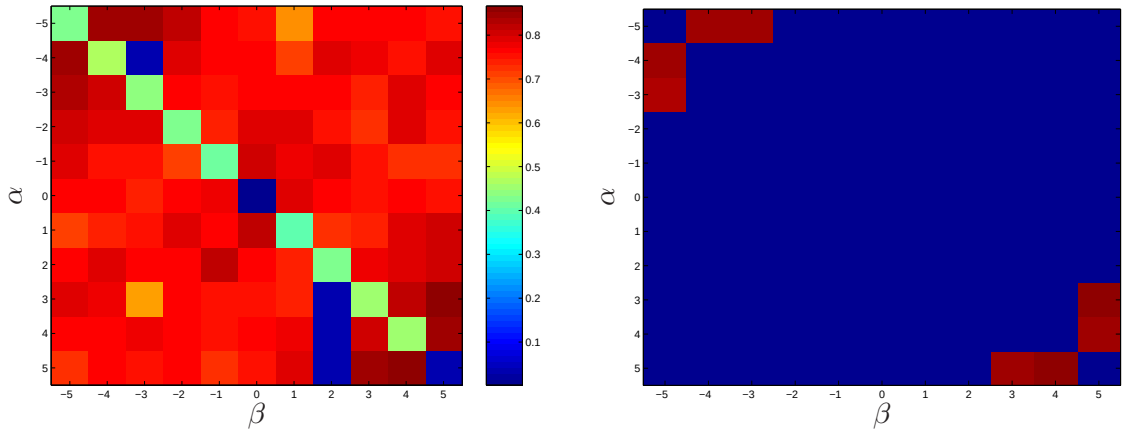
(a) Too Coarse (b) Too Fine (c) Acceptable

Figure 4.7: Examples of how different grid sizes can give different results. The red circles represent particle trajectory points as they cross through the cross-section and the light blue squares represent areas of no mixing detected.

The problem of determining the correct grid size is difficult. There are always going to be small areas that will be mistakingly detected as poor mixing. However because we use over 2000 points (i.e. the particle has travelled through 2000 periods), we can use very high resolution grids and we hope that the results won't change much, provided the grid size is sensible. Next we try some different grid sizes and see if the best configurations remain the same.

Initially we look at channels with $U_0 = 1$, $h = 0.25$, $a = 0$ and vary α and β from -5 to 5 in increments of 1. So this is a total of 121 different configurations. We choose -5 and 5 as the minimum and maximum values of α and β because anything greater than 6 or less than -6 would make a herringbone length close to or greater than the length of the period $L = 2\pi$. We do not want herringbones to span the entire period of the channel as this is unrealistic. We modified the code in appendix B.4 slightly so that it outputs each configurations Poincaré section for 2000 periods at $x = 0$ and $x = L/2$ to a different file. We then created a program in Matlab which reads the $x = 0$ and $x = L/2$ file for each configuration and calculates a mixing score (See appendix C.1). The mixing score is the total number of blocks containing at least one point (for both forwards and backwards crossings) on both the section at $x = 0$ and $x = L/2$ divided by the total number of blocks on each section (i.e twice the size of the grid). So the mix score will be between 0 and

1. The closer to 1 the score is, the better the channel is at mixing. However the scores depend on the grid size, so the top score might be 0.8 for one grid size and 0.9 for another. Therefore the mix score means nothing unless compared to the scores of other configurations for the same grid size.



(a) All mix scores (b) Mix scores above 0.82 only

Figure 4.8: Image plots of the mix scores from $\alpha = -5$ to 5 and $\beta = -5$ to 5 with $a = 0$, $h = 0.25$, $U_0 = 1$. A 48 by 12 grid (y by z) has been used on the Poincaré section. In plot (a) the darker the red, the higher the score. In plot (b) mix scores > 0.82 are red, scores < 0.82 are blue.

In figure 4.8 there are two image plots of how the scores vary for changing α and β. Figure 4.8(a) shows all of the mix scores. It is apparent that for $\alpha = \beta$ (straight slats rather than herringbones) the scores are quite poor, however with the exception of a few anomalies, all the other scores are quite high. The anomalies are probably due to the initial position not being in the main region of mixing. As we can not choose the starting point for many different configurations, this was just chosen randomly and is not likely to be good for every configuration. Figure 4.8(b) shows only the scores above 0.82. Now we can see that there is a pattern. All of the highest scores are combinations of $\alpha = 3$, $\beta = 5$ or $\alpha = 4$, $\beta = 5$ and their rotations or reflections.

Table 4.1 shows the results for three different grid sizes. The results are fairly consistent and agree with figure 4.8. For all three grid sizes, the best eight configurations are the same. Not only that, they are actually the same two configurations which are either reflected or rotated. The actual order of the best eight does vary slightly, however we did expect slight variations in the results. The fact that for three different grid sizes, the best eight results are all the same and they are all variations of two configurations suggests that these are the two best configurations. The other important fact is that it appears that the direction of the herringbone makes very little difference

(a) 40 by 16

α	β	Mix Score
-5	-4	0.81797
-5	-3	0.81797
-4	-5	0.81328
-3	-5	0.80547
3	5	0.83281
4	5	0.81953
5	3	0.83125
5	4	0.82578

(b) 48 by 12

α	β	Mix Score
-5	-4	0.84028
-5	-3	0.84115
-4	-5	0.83941
-3	-5	0.82899
3	5	0.86198
4	5	0.85243
5	3	0.84896
5	4	0.86632

(c) 60 by 10

α	β	Mix Score
-5	-4	0.84917
-5	-3	0.83917
-4	-5	0.85583
-3	-5	0.82667
3	5	0.86417
4	5	0.84333
5	3	0.85583
5	4	0.85167

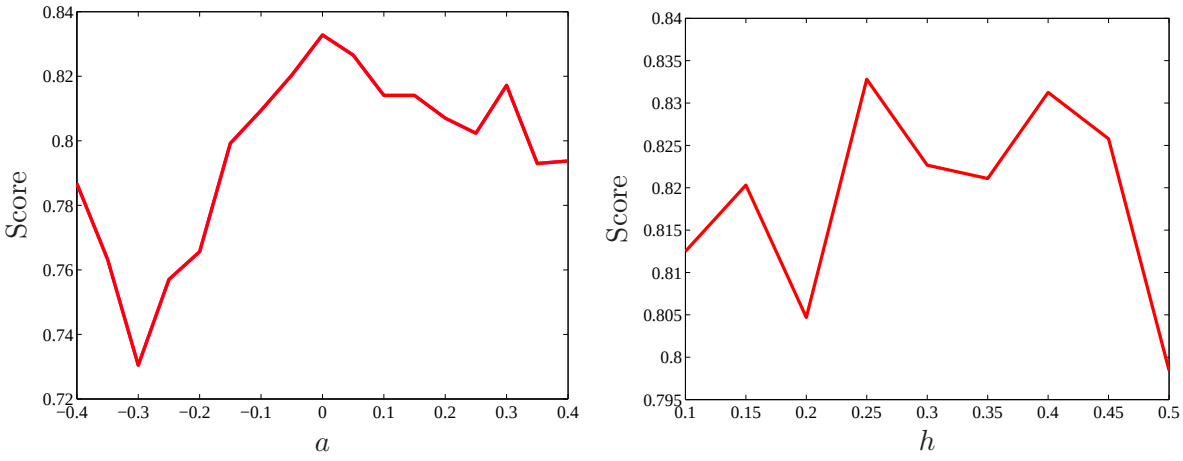
Table 4.1: Tables of the top 8 mixing scores for 3 different grid sizes(y by z).

to the mixing. So the two best configurations are ($\alpha = 3$, $\beta = 5$) and ($\alpha = 4$, $\beta = 5$). As $\alpha = 3$ or $\alpha = 4$ with $\beta = 5$ are the two best configurations we could continue here to see what happens if we vary α between 3 and 4, but the scoring system is not accurate enough to go into that much detail. From the scores we have, we can't really say whether $\alpha = 3$ or $\alpha = 4$ is the best, so it is pointless trying to look at the scores between these values. As ($\alpha = 3$, $\beta = 5$) gets the actual top score on two of the three tests, we will continue to study this configuration and see what happens if we change some other parameters.

As a slight aside, the ($\alpha = 3$, $\beta = 5$) configuration takes into account both forwards and backwards flow. This suggests that there is some recirculation occurring which could be good for mixing. The flow that mixes best for forward flow only is ($\alpha = 1$, $\beta = 0$) (or reflections and rotations of this) for all three grid sizes tested. The overall mixing score is about 0.5 less than the score of ($\alpha = 3$, $\beta = 5$) on each of the 3 tests for this configuration.

We will now look at what happens if we vary the height h of the channel or the position of a. We used exactly the same method as for the varying α and β to obtain mix scores. We use $\alpha = 3$ and $\beta = 5$ in this test and vary a from -0.4 to 0.4 keeping $h = 0.25$ and then vary h from 0.1 to 0.5 keeping $a = 0$.

Figure 4.9 shows plots of mix scores for a and h varying. Both plots don't give particularly conclusive results, however we'll start with 4.9(a). It appears that $a = 0$ gives the best score and in general $a > 0$ seems to give better scores than $a < 0$. In previous studies, the staggered herringbone was used, however we are looking at just the normal herringbone. For the staggered herringbone



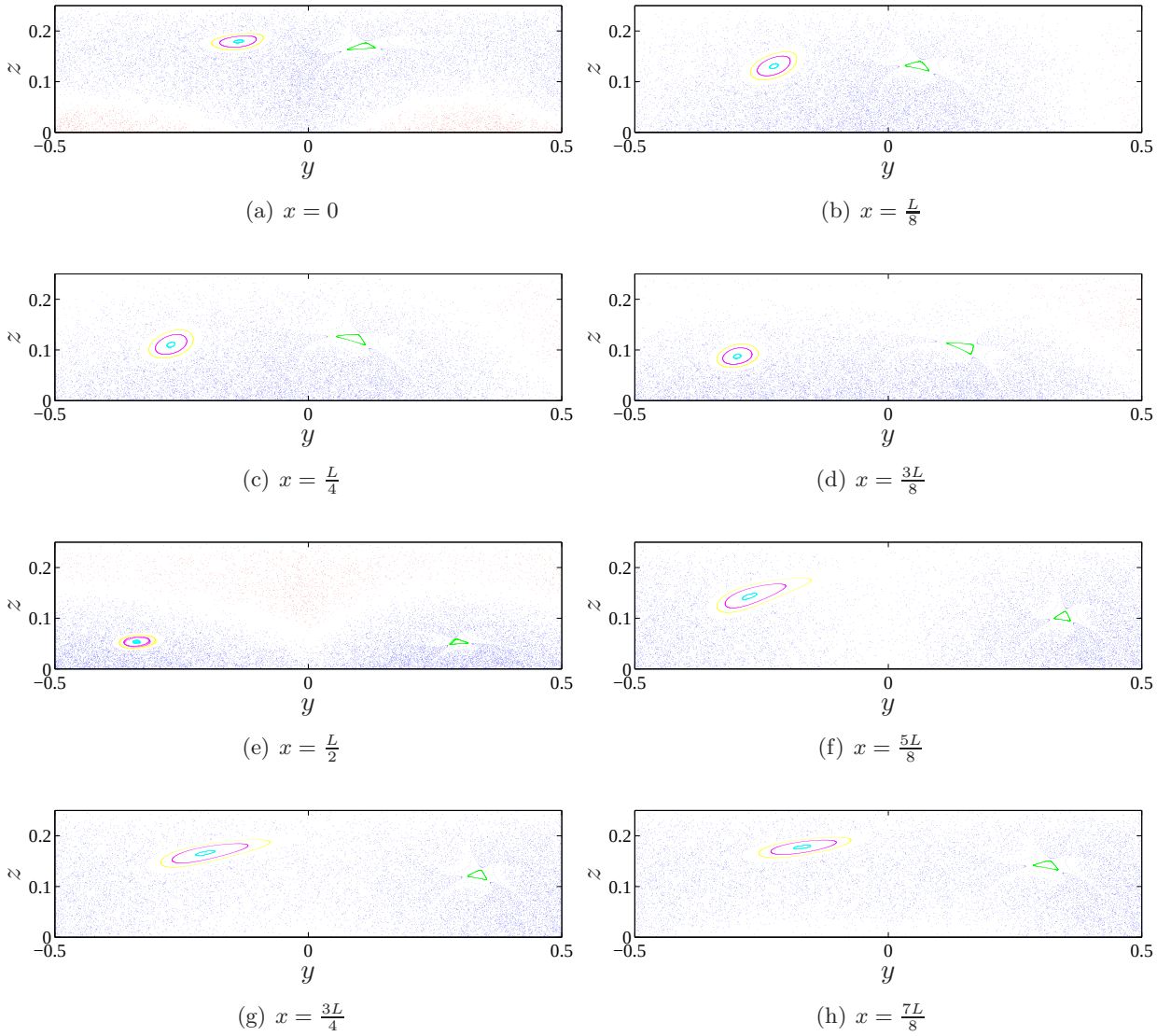
(a) Varying a (b) Varying h

Figure 4.9: Plots of mix scores for a varying and h varying.

$a > 0$ for a few periods and then $a < 0$ for a few periods etc gave the best results [9]. The problem with just the normal herringbone is that if a is not centered then one side will always dominate the flow more than the other. However for the staggered system the herringbone is reversed so overall one side doesn't dominate over the other. Therefore from now on we will look at the centered ($a = 0$) herringbone.

Figure 4.9(b) doesn't give us very much information at all. The mixing scores seem to be quite random as h varies. The best score seems to be when $h = 0.25$, however the next best is at $h = 0.4$. The score is less at 0.3 and 0.35. The problem is that the flows are chaotic, so we cannot predict what will happen for a slight change of parameters. However for the moment we cannot find a better system than $h = 0.25$, $a = 0$, $\alpha = 3$ and $\beta = 5$ for $U_0 = 1$ using this method of measuring mixing.

Figure 4.10 shows Poincaré section of the ($\alpha = 3$, $\beta = 5$) channel at eight positions through the period of the channel. These have been created from 10,000 periods of flow to create a high resolution picture. There are very small regions of poor mixing and we can see how they move through the channel. However these regions are very small and the majority of the particles in the cross-section are in the same chaotic region. At $x = 0$ there is forwards flow for most of the cross-section, however at the bottom of the channel there is some backwards flow. As x increases the forwards flow swirls to the left of the channel (although as the flow is coming towards us that would actually be the right of the channel) and the backwards flow moves round to the right of



(a) $x = 0$ (b) $x = \frac{L}{8}$

(c) $x = \frac{L}{4}$ (d) $x = \frac{3L}{8}$

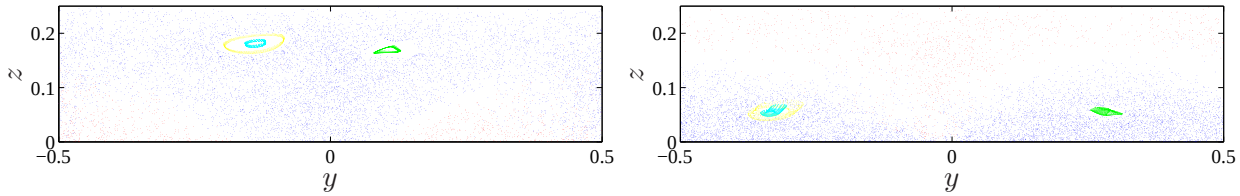
(e) $x = \frac{L}{2}$ (f) $x = \frac{5L}{8}$

(g) $x = \frac{3L}{4}$ (h) $x = \frac{7L}{8}$

Figure 4.10: Poincaré sections for channel: $U_0 = 1$, $a = 0$, $h = 0.25$, $\alpha = 3$, $\beta = 5$ at different positions in the channel. Blue is forwards flow and red is backwards flow for the same trajectory. The yellow, magenta, cyan and green loops represent different trajectories and are forwards flow only. Forwards flow is coming towards the reader and backwards flow is in to the paper.

the channel until $x = L/2$. Here there is the most backwards flow during the period L. All the forwards flow is now at the bottom of the channel and the backwards flow is at the top of the channel. However for $x > L/2$ there appears to be no backwards flow until $x = L(= 0)$. So the motion in the first half of the period is quite different to the second half. In the first half there is a swirling motion with some backwards flow, but in the second there is complete forwards flow. It is apparent from the diagrams, just as the mixing score suggests that there are only small regions of no mixing and this configuration is probably a reasonable mixer. However we do not know how fast the channel will actually mix.

Earlier we stated that for $U_0 = 5$ and provided $S_F = 1$ that the mixer is very similar as when $U_0 = 1$ with $S_F = 1$. We will now check to see if the Poincaré section for the ($\alpha = 3$, $\beta = 5$) configuration with $U_0 = 5$ looks the same.

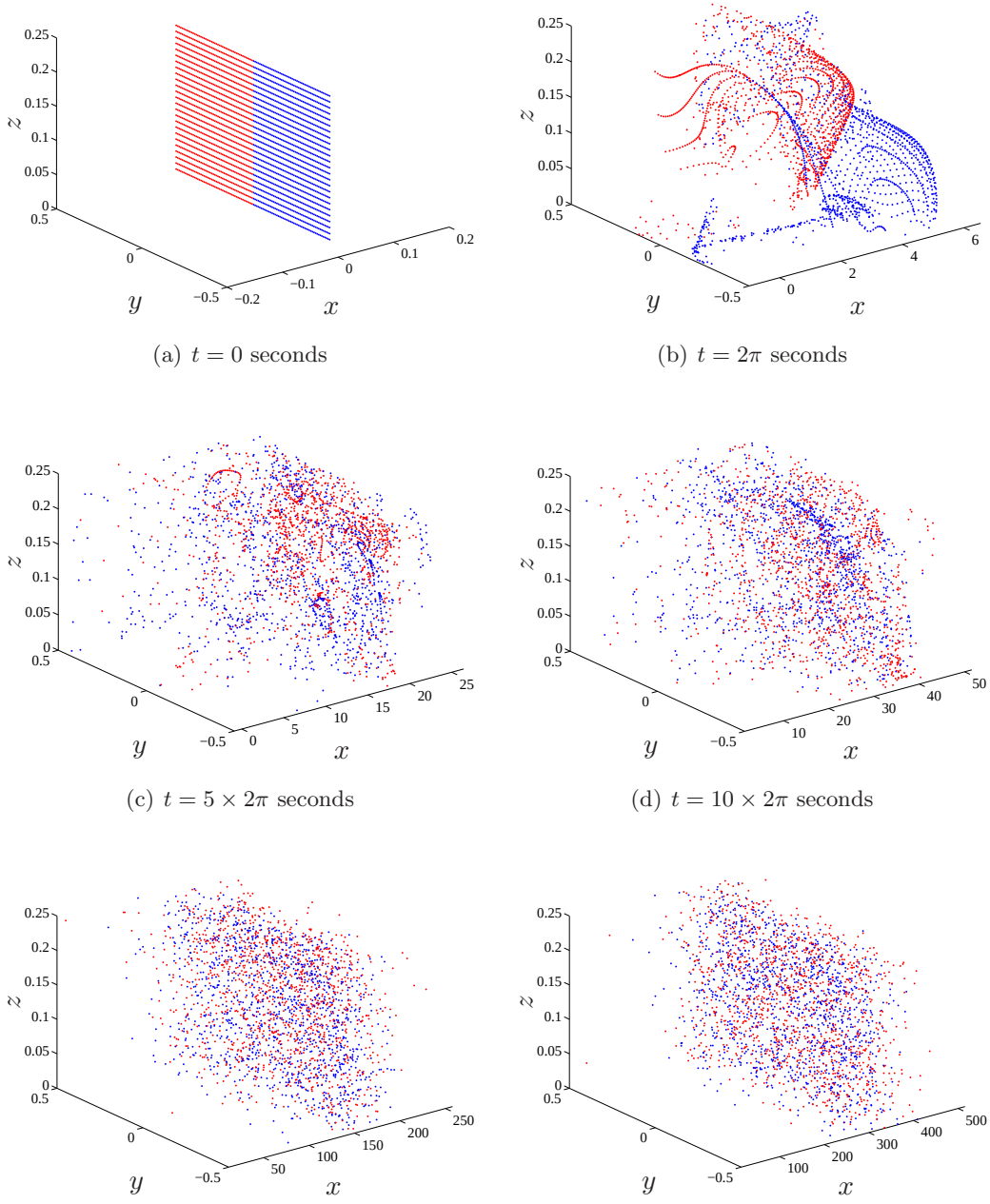


(a) $x = 0$ (b) $x = \frac{L}{2}$

Figure 4.11: Poincaré sections for channel: $U_0 = 5$, $a = 0$, $h = 0.25$, $\alpha = 3$, $\beta = 5$. Blue is forwards flow and red is backwards flow for the same trajectory. The yellow, cyan and green loops represent different trajectories and are forwards flow only.

Figure 4.11 shows the Poincaré sections of the (3,5) configuration with $U_0 = 5$. Notice that they look very similar to the corresponding sections in figure 4.10; if anything the flow looks more chaotic. There seems to be a little bit of mixing between the backwards and forwards flow near to where they meet. We cannot tell from Poincaré sections if the $U_0 = 5$ flow is a better mixer than $U_0 = 1$ flow. However later in the project we study the time that each configuration takes to mix.

Through the analysis using Poincaré sections we have found the $\alpha = 3$, $\beta = 5$, $a = 0$, $h = 0.25$ configuration to be the best mixer from the configurations studied when looking at mixing in the cross-section. From now on we will call this the 3-5 mixer.



(a) $t = 0$ seconds

(b) $t = 2\pi$ seconds

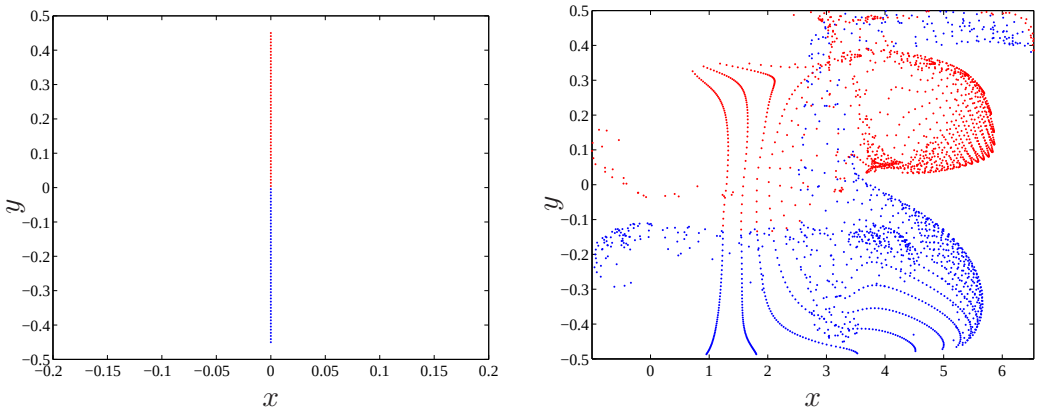
(c) $t = 5 \times 2\pi$ seconds

(d) $t = 10 \times 2\pi$ seconds

(e) $t = 50 \times 2\pi$ seconds

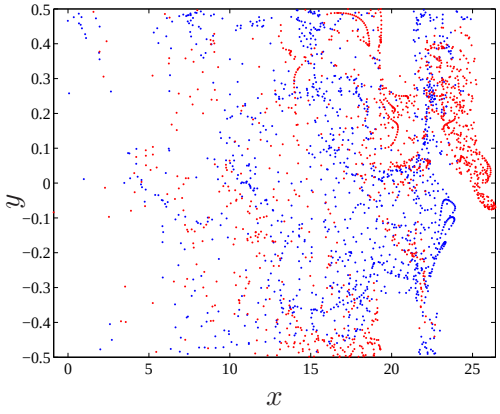
(f) $t = 100 \times 2\pi$ seconds

Figure 4.12: 3D views of the dispersion of particles in the channel at different times.

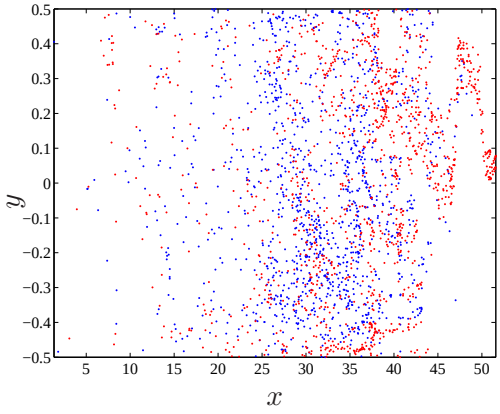


(a) $t = 0$ seconds

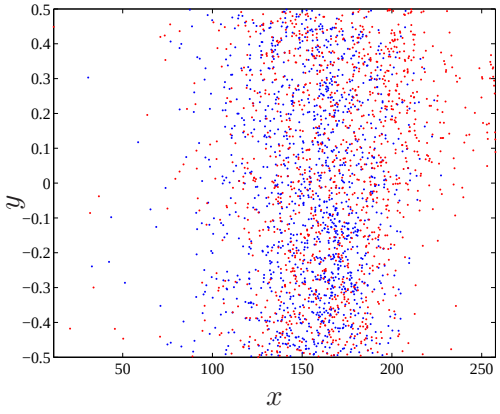
(b) $t = 2\pi$ seconds



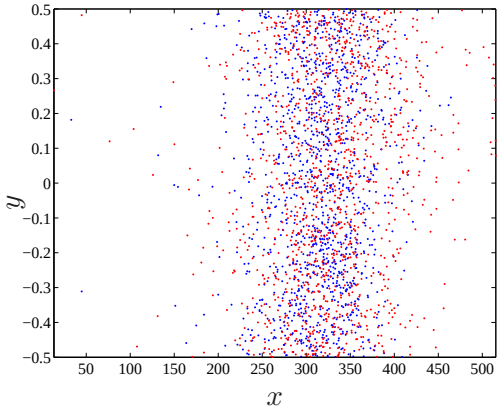
(c) $t = 5 \times 2\pi$ seconds



(d) $t = 10 \times 2\pi$ seconds

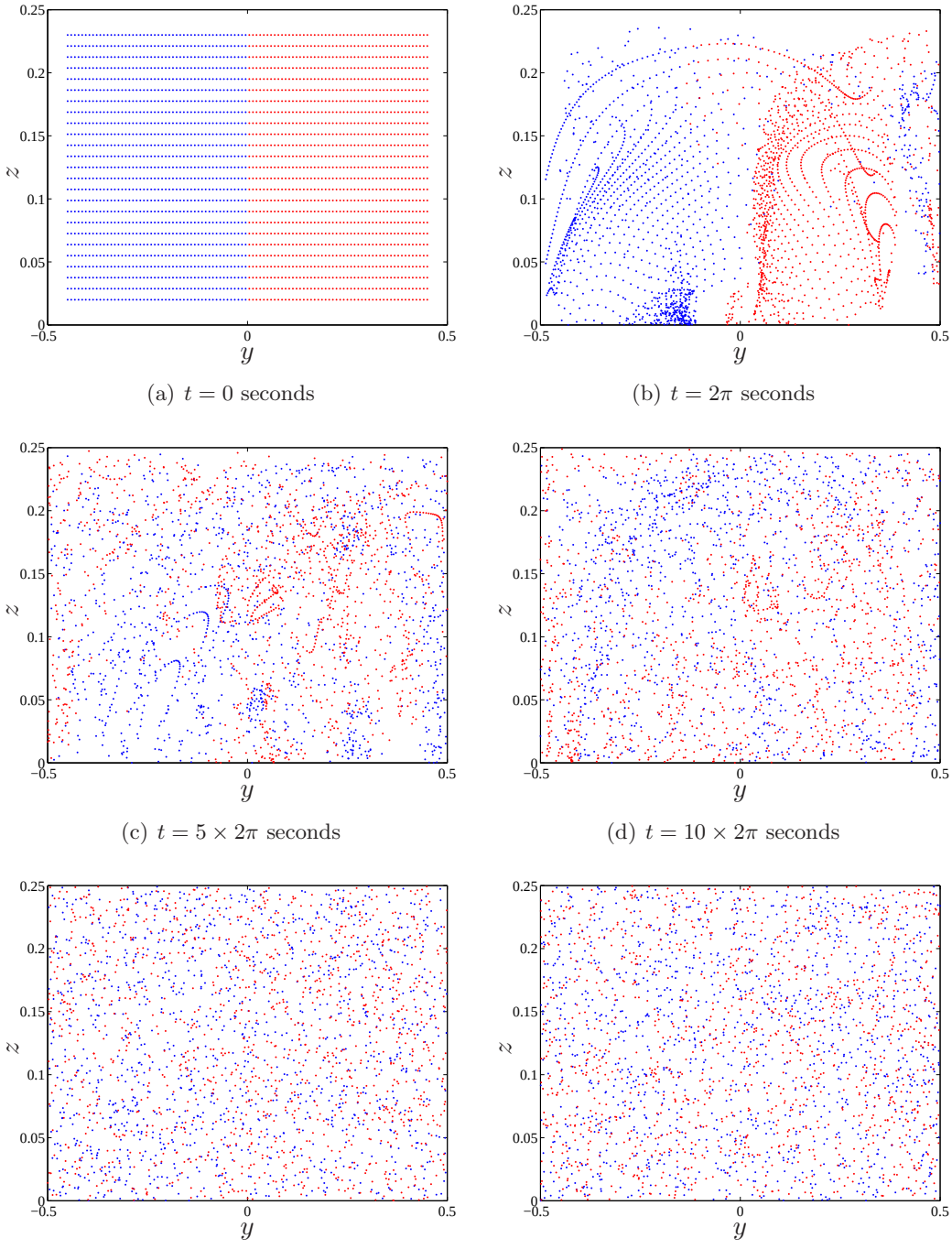


(e) $t = 50 \times 2\pi$ seconds



(f) $t = 100 \times 2\pi$ seconds

Figure 4.13: Top down views of the dispersion within the channel.



(a) $t = 0$ seconds

(b) $t = 2\pi$ seconds

(c) $t = 5 \times 2\pi$ seconds

(d) $t = 10 \times 2\pi$ seconds

(e) $t = 50 \times 2\pi$ seconds

(f) $t = 100 \times 2\pi$ seconds

Figure 4.14: Down channel views of the dispersion within the channel.

4.4 Dispersion

It is important to study the longitudinal spreading in the channel. This will happen through diffusion itself, however the mixer is likely to increase the spreading. We now analyse the longitudinal spreading to make sure the fluid is not being spread too much along the length of the channel. For example, if we were to analyse a drop of DNA then we might want it to react with a fluid, therefore we would like the DNA to be mixed with the fluid by the end of the channel. However we don't want there to still be bits of the DNA at the beginning of channel as we will only be looking at the fluid at the end of the channel.

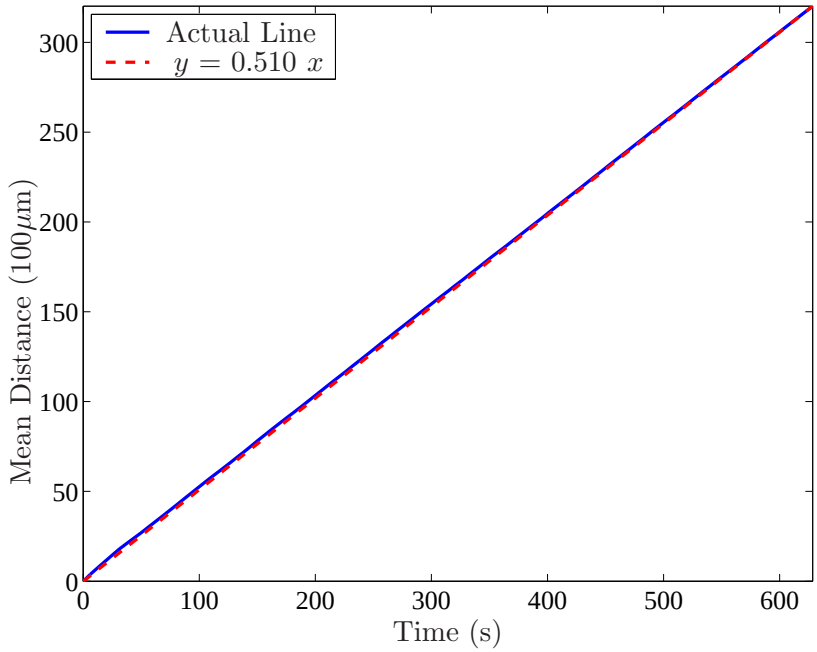
We created a program which can be seen in appendix B.5, called `velmmix_dispersion.cpp`. This program takes a grid of particles spread equally in the $y-z$ plane at $x = 0$ and integrates the particles, outputting the time, position and velocity at times a multiple of $L/U_0 = 2\pi$ seconds for $U_0 = 1$. We then look at the spreading of these particles in x. We would like the mean x-distance to increase linearly in time, i.e. there is a constant net flow. However more importantly we would like the variance of the x-distance to be linear with respect to time (i.e. the spreading to be linear over time) as this means that the spreading is diffusive.

We executed the program for the 3-5 mixer with $U_0 = 1$ for a 100 by 25 grid for y between -0.45 and 0.45 and z between 0.02 and 0.23 for 100 multiples of $t = 2\pi$ seconds. Figures 4.12, 4.13 and 4.14 show some plots of the positions of the particles after 0, 1, 5, 10, 50 and 100 multiples of $t = 2\pi$.

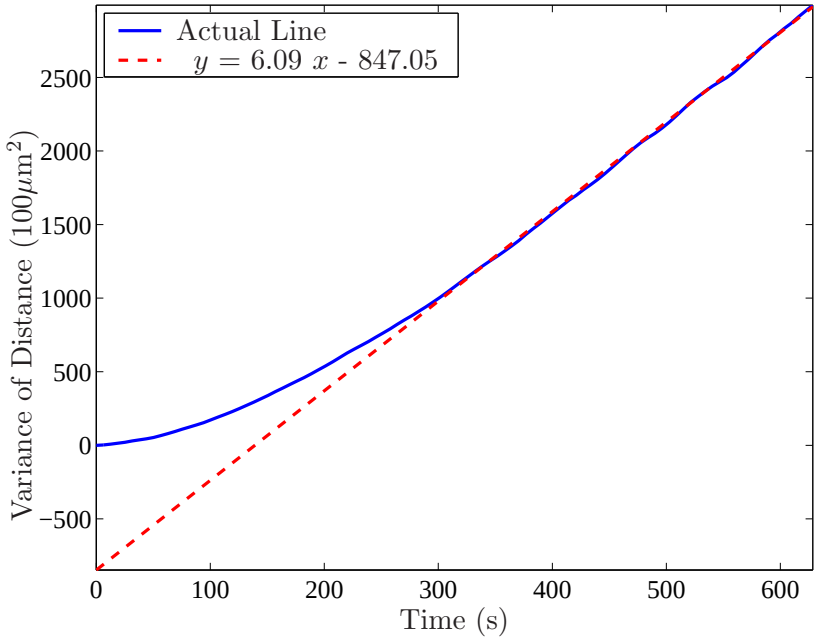
Figure 4.12 shows the 3D view of the particles. In figure 4.12(a) there are the initial positions of particles. Each half of the grid is coloured a different colour. At $t = 2\pi$ in 4.12(b) the blue and red particles are still quite separate. However after 5 or 10 multiples of 2π the blue and red particles are beginning to mix. Here the longitudinal spreading spans most of the length of the channel. By 50 or 100 multiples of 2π though, most of the particles are near the middle of the channel with respect to x which is a good sign that the spreading isn't too large.

Figures 4.13 and 4.14 show the top down view and down channel view for the same plots as the 3D views in figure 4.12. These 2D plots make the visualisation a little easier as one dimension has been removed. Both confirm the observations made about the 3D plots. Although the particles are not initially mixed, by 50 or 100 multiples of $t = 2\pi$, the particles are well mixed. Figure 4.13 also confirms that the majority of particles are clustered around the middle of the channel in the x-direction for large enough t.

Figure 4.15 shows plots of the mean x-distance vs time and the variance of the x-distance vs



(a) Mean Distance vs Time



(b) Variance of Distance vs Time

Figure 4.15: Mean and variance plots of the x-distance vs time

time. In figure 4.15(a) the mean distance of the particles is growing linearly which shows that the residence time [4] of the particles remains constant as time increases, so the mean flow is remaining constant. In figure 4.15(b) is the variance of the distance vs time. The variance is the dispersion or spreading of the particles. Initially the dispersion is not linear, however as time increases it appears to tend towards the line $y = 6.09x - 847.05$. We observed earlier that the particles spread a bit randomly at first, but after a number of periods the particles grouped together in the x-direction. Although the dispersion isn't initially linear, for large enough times the dispersion is linear so the spreading is like a random walk which is diffusive.

The diffusion coefficient is the slope of the line, which is 6.09 (i.e. $\kappa = 6.09 \times 100\mu\text{m}^2/\text{s} = 6.09 \times 10^{-8}\text{m}^2/\text{s}$). For DNA $\kappa = 10^{-10}$, so the channel increases diffusion in the x-direction but this is within acceptable limits, as the channel is a lot longer in the x-direction than y or z.

In this section we have analysed the 3-5 mixer for $U_0 = 1$ to see the dispersion of particles. This has shown that the particles appear to mix well, the residence time is constant and the dispersion of the particles is diffusive. When combining this information with the Poincaré section analysis it is apparent that the 3-5 mixer doesn't get too much longitudinal spreading and it mixes well in the cross-section. This implies that the 3-5 mixer will mix significantly faster than mixing due to diffusion only. However we still haven't looked at any time scales for the mixing.

Chapter 5

Lyapunov Exponents

So far we have considered classical ways of visualising the flows which only give an idea in a global sense how the mixer behaves [8]. Another method used to measure mixing is the study of Lyapunov exponents of the flow [2].

5.1 Theory

The idea behind Lyapunov exponents is related to the stretching of an infinitely small blob (ball of initial conditions) in the fluid. For a chaotic flow, this blob will stretch exponentially over time into an ellipsoid with the volume remaining constant. Lyapunov exponents are a measure of this divergence of initial conditions or stretching of the ellipsoid. For a flow there will be three Lyapunov exponents; one positive, one negative and one approximately 0, with the sum exactly 0 due to incompressibility. Hence the positive exponent should be approximately equal in magnitude to the negative exponent. In this project we only look at the largest exponent, as this gives an indication of how quickly the initial conditions diverge. We would like the Lyapunov exponent to be as large as possible as for a higher exponent, the system is more chaotic and hence the mixing is better.

To calculate the Lyapunov exponents we take two particles

$$\begin{aligned} &\boldsymbol{x}(\boldsymbol{a}, t) \qquad , \\ &\boldsymbol{x}(\boldsymbol{a} + \delta\boldsymbol{a}, t)\,. \end{aligned}$$

i.e. two particles, one starting at $\boldsymbol{a}$ and one starting at $\boldsymbol{a}+\delta\boldsymbol{a}$. Therefore the initial conditions are

$$\begin{aligned}\boldsymbol{x}(\boldsymbol{a},0) &= \boldsymbol{a}\,,\\ \boldsymbol{x}(\boldsymbol{a}+\delta\boldsymbol{a},0) &= \boldsymbol{a}+\delta\boldsymbol{a}\,,\end{aligned}$$

therefore

$$\begin{aligned}\delta\boldsymbol{x}(t) &= \boldsymbol{x}(\boldsymbol{a}+\delta\boldsymbol{a},t)-\boldsymbol{x}(\boldsymbol{a},t)\,,\\ \delta\dot{\boldsymbol{x}}(t) &= \dot{\boldsymbol{x}}(\boldsymbol{a}+\delta\boldsymbol{a},t)-\dot{\boldsymbol{x}}(\boldsymbol{a},t)\,.\end{aligned}$$

Now

$$\dot{\boldsymbol{x}}(\boldsymbol{a},t)=\boldsymbol{v}(\boldsymbol{x}(\boldsymbol{a},t),t)\,, \tag{5.1}$$

hence

$$\begin{aligned}\delta\dot{\boldsymbol{x}}(t) &= \boldsymbol{v}(\boldsymbol{x}(\boldsymbol{a}+\delta\boldsymbol{a},t),t)-\boldsymbol{v}(\boldsymbol{x}(\boldsymbol{a},t),t)\,,\\ &= \boldsymbol{v}(\boldsymbol{x}+\delta\boldsymbol{x},t)-\boldsymbol{v}(\boldsymbol{x},t)\,.\end{aligned}$$

We now assume that $||\delta\boldsymbol{x}|| \ll 1$ and perform a Taylor expansion to get

$$\frac{\mathrm{d}}{\mathrm{d}t}\delta x_i=\frac{\delta v_i}{\delta x_j}\delta x_j+O(\delta x^2)\,.$$

We now divide through by δa_k to get

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\delta x_i}{\delta a_k}=\frac{\delta v_i}{\delta x_j}\frac{\delta x_j}{\delta a_k}\,. \tag{5.2}$$

From equations (5.1) and (5.2) there are a total of 12 equations to solve for a given $\boldsymbol{a}$. Equation (5.1) has $\boldsymbol{x}(\boldsymbol{a},0)=\boldsymbol{a}$ for its three initial conditions. For (5.2), $\delta\boldsymbol{x}/\delta\boldsymbol{a}$ is a 3×3 matrix, and there are 9 initial conditions

$$\frac{\delta x_i}{\delta a_k}(\boldsymbol{a},0)=\delta_{ik}\,,$$

where δ_{ik} is the identity matrix. It is the matrix $\delta\boldsymbol{x}/\delta\boldsymbol{a}$ that represents the stretching of the ellipsoid. The determinant of this matrix relates directly to the volume of the fluid ellipsoid. Due to incompressibility, the volume of the fluid ellipsoid is constant over time. We therefore look at $\mathrm{d}/\mathrm{d}t$ of the determinant

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left[\det\left(\frac{\delta\boldsymbol{x}}{\delta\boldsymbol{a}}\right)\right] &= (\nabla\cdot\boldsymbol{v})\left[\det\left(\frac{\delta\boldsymbol{x}}{\delta\boldsymbol{a}}\right)\right]\\ &= 0 \qquad (\text{because } \nabla\cdot\boldsymbol{v}=0)\,.\end{aligned}$$

Therefore the volume remains constant and from the initial conditions we get

$$\det\left(\frac{\delta \boldsymbol{x}}{\delta \boldsymbol{a}}\right) = 1\,.$$

Finally to obtain the Lyapunov exponents we find the eigenvalues of $\delta\boldsymbol{x}/\delta\boldsymbol{a}$ and label the largest eigenvalue Λ_1. If the system is chaotic then Λ_1 will increase exponentially over time; i.e.

$$\Lambda_1 = \mathrm{e}^{\lambda_1 t}\,,$$

where λ_1 is the finite time Lyapunov exponent (FTLE). Hence

$$\lambda_1 = \tfrac{1}{t}\ln(\Lambda_1)\,.$$

Therefore for initial position $\boldsymbol{a}$ we have the FTLE $\lambda_1(\boldsymbol{a}, t)$ for a given time t. We also get the following

$$\lambda_1^\infty = \lim_{t\to\infty}\lambda_1(\boldsymbol{a}, t)\,. \tag{5.3}$$

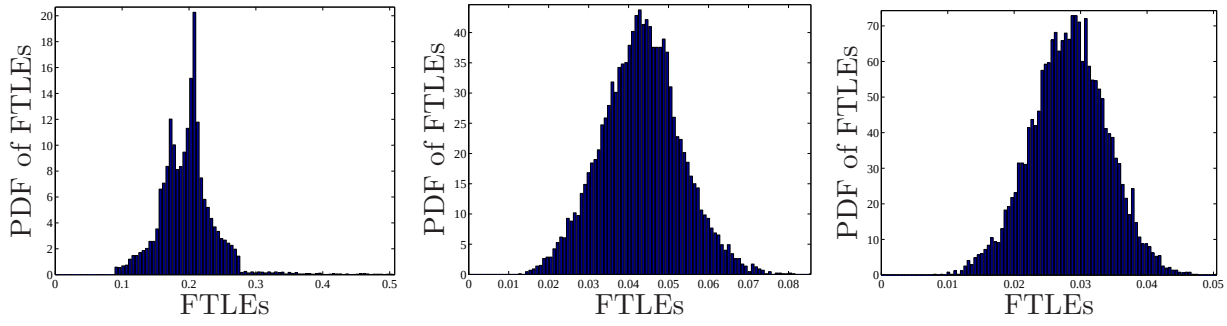
Note that unlike the FTLE, this does not depend on the initial position $\boldsymbol{a}$. This is called the Lyapunov exponent of the flow. However, we cannot integrate to infinity, so to get around this we integrate many different initial positions (within the same region) to a large enough time t and take the mean $\bar{\lambda}$ of the λ_1 values. This $\bar{\lambda}$ tends towards the Lyapunov exponent λ_1^∞ for a large enough time.

5.2 Histograms, Mean and Standard Deviation

We will now analyse the Lyapunov exponents for the 3-5 mixer for $U_0 = 1$ and 5. The `velmmix.hpp` program calculates the matrix $\delta\boldsymbol{x}/\delta\boldsymbol{a}$ and then the program `velmmix_lyapsquare.cpp` (appendix B.6) uses this to find the eigenvalues, calculate the FTLEs and output them to a file. The program does this for a square of initial conditions and outputs the 3 FTLEs for each trajectory at set times. The program is set to output for the 3-5 mixer. We chose a square from $y = -0.1$ to -0.05 and $z = 0.05$ to 0.1 with $x = 0$ for the starting positions. Refer to figure 4.10 to see that all particles within this square are in the same region at $x = nL = 0$. We used a 101×101 grid which gives a total of 10201 sets of FTLEs and outputted the Lyapunov values for each trajectory at times of multiples of 6.3 (i.e. roughly the time to travel 1 period at $U_0 = 1$) up to t=504 which is approximately 80 periods L.

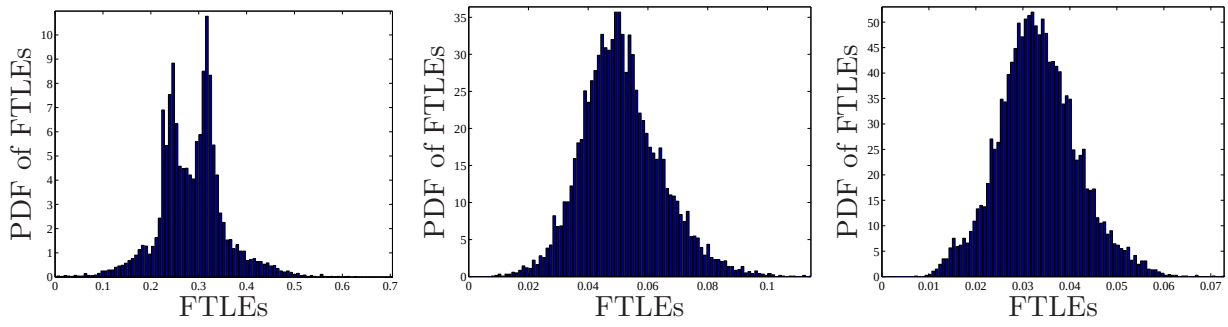
Calculating the Lyapunov values is quite time consuming and for 10201 particles it takes a long time to calculate all the values up to $t = 504\,\mathrm{s}$. Therefore initially we used 1 Fourier mode and

then later 20 modes and then finally tried 50 modes. For $U_0 = 1$, 1 mode only took 40 minutes, but 20 modes took 12 hours, and 50 modes took 3 days. For $U_0 = 5$ the 20 mode system took 4 days. This is because the particle is travelling a lot further down the channel for $U_0 = 5$. For this reason we have not found the FTLEs using 50 modes for $U_0 = 5$. Although we have not looked at the 1 mode solutions so far, we will now see whether the Lyapunov exponent is larger for 1 mode or 20 or 50 modes. For 1 mode there is a sine function induced on the floor of the channel, whereas for 20 or 50 modes we have the step function.



(a) $t = 18.9\,\mathrm{s}$ (b) $t = 201.6\,\mathrm{s}$ (c) $t = 504\,\mathrm{s}$

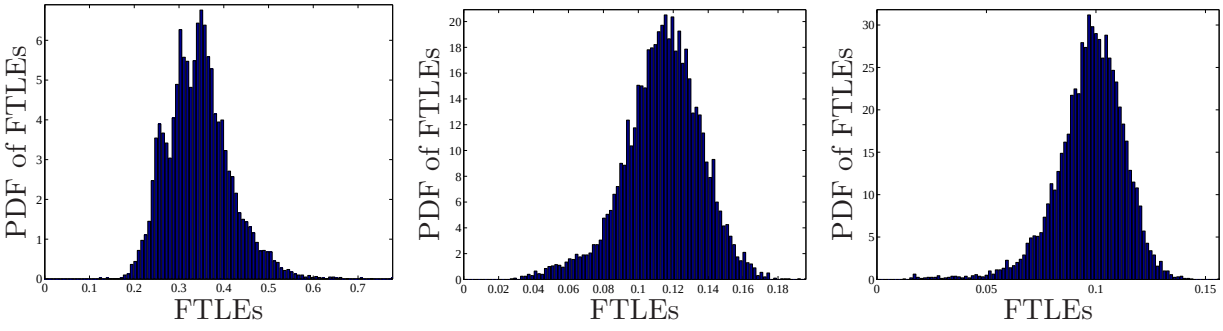
Figure 5.1: PDF of Finite time Lyapunov exponents for flow with 1 Fourier mode and $U_0 = 1$.



(a) $t = 18.9\,\mathrm{s}$ (b) $t = 201.6\,\mathrm{s}$ (c) $t = 504\,\mathrm{s}$

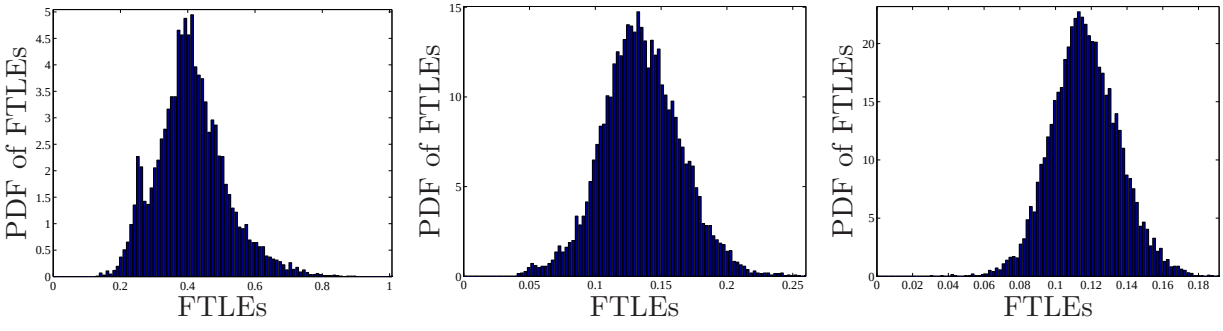
Figure 5.2: PDF of Finite time Lyapunov exponents for flow with 50 Fourier modes and $U_0 = 1$.

We simulated the flow for 1 and 50 Fourier modes for $U_0 = 1$ and 1 and 20 modes for $U_0 = 5$. In figures 5.1 to 5.4 there are PDF's of the finite time Lyapunov exponents as time increases. The histograms give an idea of how the FTLEs change for increasing values of t. For all 4 tests the distribution is quite random for $t = 18.9$ seconds, but as the time increases the distribution appears to become more Gaussian and the range of the exponents is narrowing. This implies that



(a) $t = 18.9\,\mathrm{s}$ (b) $t = 201.6\,\mathrm{s}$ (c) $t = 504\,\mathrm{s}$

Figure 5.3: PDF of Finite time Lyapunov exponents for flow with 1 Fourier mode and $U_0 = 5$.



(a) $t = 18.9\,\mathrm{s}$ (b) $t = 201.6\,\mathrm{s}$ (c) $t = 504\,\mathrm{s}$

Figure 5.4: PDF of Finite time Lyapunov exponents for flow with 20 Fourier modes and $U_0 = 5$.

the exponents are tending towards a limit as implied by (5.3).

Another observation is that for $t = 18.9$, both of the $U_0 = 5$ flows appear to have a more Gaussian distribution than the $U_0 = 1$ flow. This is a sign that the faster flow could be mixing faster. This is good as 500 μm/s is a more realistic and common speed for microchannels compared to 100 μm/s [6]. Also the faster velocity flow needs to mix faster, so that the channel is not required to be too long for a good mix to be obtained.

Figures 5.5 and 5.6 show plots of the mean and standard deviation of the exponents. Figure 5.5 shows that all 4 means tend towards a limit. This limit is the Lyapunov exponent ($\bar{\lambda}$) for the flow. For both of the $U_0 = 5$ flows, $\bar{\lambda}$ is much larger than for $U_0 = 1$ which implies that $U_0 = 5$ may be the better mixer. In addition, the 20 and 50 mode flows appears to have larger $\bar{\lambda}$ than the 1 mode flow with the same U_0 value. Figure 5.6 shows that the standard deviation (σ) is proportional to $1/\sqrt{t}$. Figure 5.6 also shows that the coefficient of σ is larger for $U_0 = 5$ compared to $U_0 = 1$ and

when there are 20 or 50 modes as opposed to 1 mode. We define σ as

$$\sigma = \sqrt{\frac{\nu}{t}}.$$

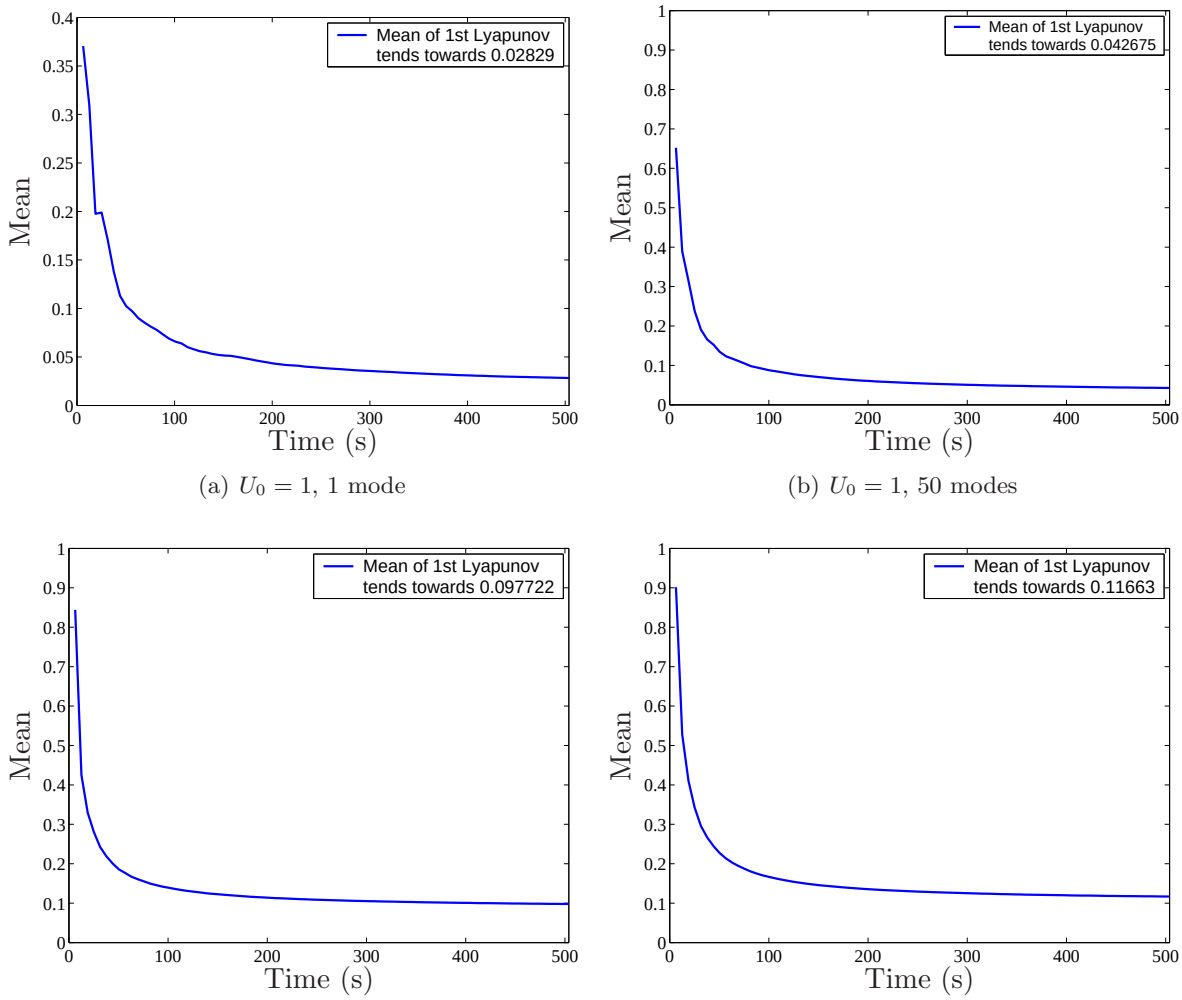
and this ν value has a negative effect on mixing (see section 5.3) . The values of $\bar{\lambda}$ and ν which have been obtained from figures 5.5 and 5.6 are in table 5.1.

From the $\bar{\lambda}$ and ν values, it is apparent that the Lyapunov exponents are larger for a step function rather than a sine function and for $U_0 = 5$ rather than $U_0 = 1$. However the standard deviation is larger too which affects the mixing.

U_0	Modes	$\bar{\lambda}$	Slope ($\nu^{\frac{1}{2}}$)	ν
1	1	0.028	0.127	0.016
1	50	0.042	0.219	0.048
5	1	0.097	0.268	0.072
5	20	0.116	0.410	0.168

Table 5.1: Table of $\hat{\lambda}$ and ν values.

Earlier we chose the square of initial positions to be in one region for each of the 4 configurations. However if a mistake is made and the initial positions cross into 2 regions, the standard deviation plot provides a large warning. Figure 5.7 is a plot of the standard deviation when the initial position square crosses into 2 regions. There appears to be a constant value of 0.5 which suggests that σ may not go to 0 at $t = \infty$. This violates the theory and if the standard deviation plot has a constant value, there has probably been a mistake made with the choice of the initial position square.



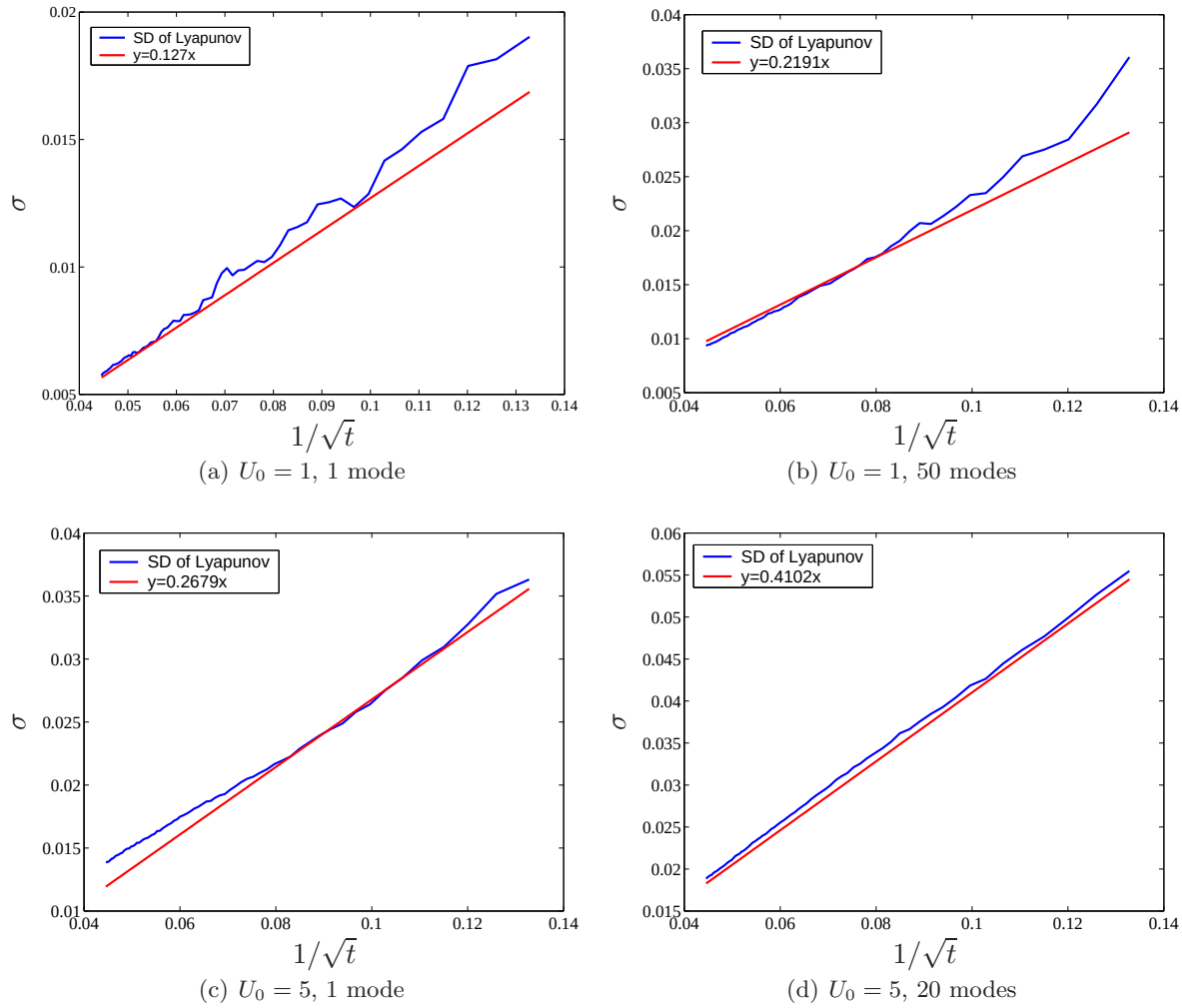
(a) $U_0 = 1$, 1 mode

(b) $U_0 = 1$, 50 modes

(c) $U_0 = 5$, 1 mode

(d) $U_0 = 5$, 20 modes

Figure 5.5: Plots of the mean value of the finite time Lyapunov exponents.



(a) $U_0 = 1$, 1 mode

(b) $U_0 = 1$, 50 modes

(c) $U_0 = 5$, 1 mode

(d) $U_0 = 5$, 20 modes

Figure 5.6: Plots of the standard deviation of the finite time Lyapunov exponents with a line with the same gradient.

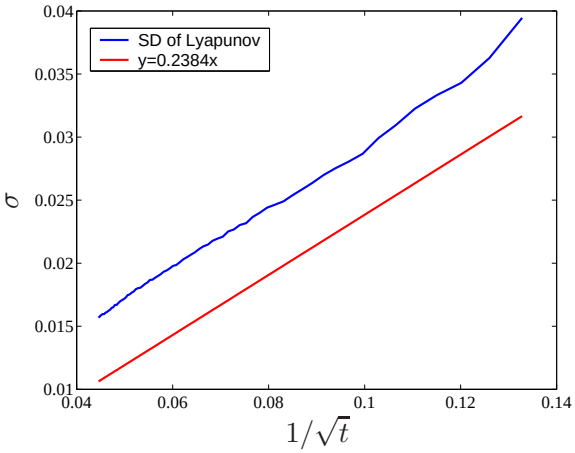


Figure 5.7: Plot of the standard deviation of the finite time Lyapunov exponent for $U_0 = 5$, with 1 mode when the square of initial conditions crosses into 2 regions. The red line is a line with the same gradient.

5.3 Mixing Time

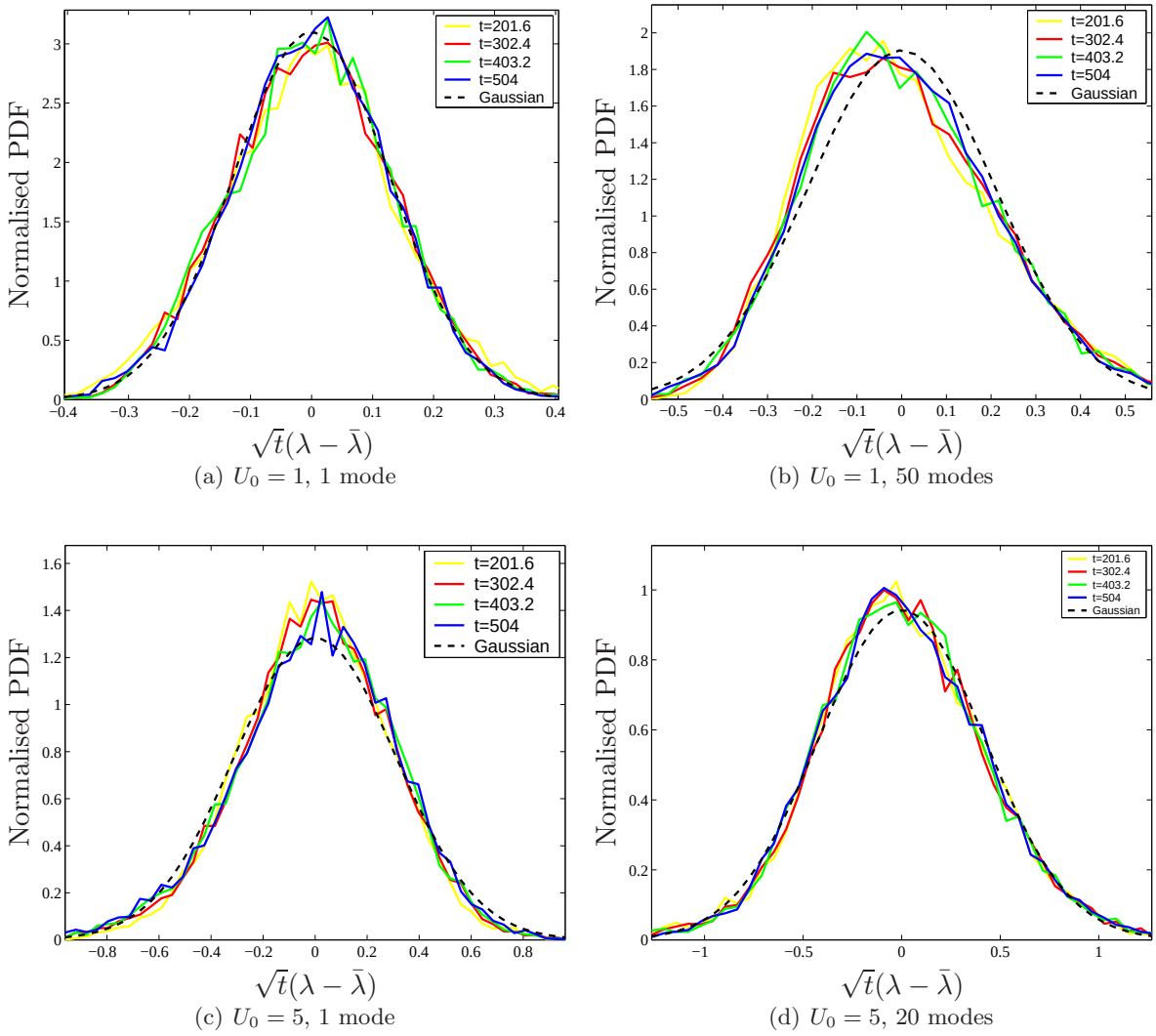
We now look more closely at how the $\bar{\lambda}$ and ν values affect the mixing to get an idea of how long these flows will take to mix. The PDF of the FTLEs appear to be forming a Gaussian shape. The standard deviation changes proportionally to $1/\sqrt{t}$ so we will now look to see if the distributions are staying the same for large enough times. We do this by normalising the PDFs at times $t = 201.6\,\mathrm{s}$, $t = 302.4\,\mathrm{s}$, $t = 403.2\,\mathrm{s}$, $t = 504\,\mathrm{s}$, shifting the FTLE λ so that the mean is 0 and multiplying by $\sqrt{t}$. i.e we plot

$$\frac{\text{PDF}}{\text{Area}} \quad \text{vs} \quad \sqrt{t}(\lambda - \bar{\lambda})\,.$$

The Matlab code for doing this is in appendix C.2 with an example using some data files on the CD. In figure 5.8 there are 4 plots for 1 and 20 or 50 modes for $U_0 = 1$ and $U_0 = 5$. The distributions do look Gaussian. Particularly for the 1 mode configurations, the distributions are very Gaussian. The plot for $U_0 = 1$ with 50 modes is slightly skewed. For 20 modes with $U_0 = 5$, the plot is very similar but not quite as skewed. This is not a problem as they are still very close to being Gaussian.

We have obtained acceptable Gaussian distributions for the finite time Lyapunov exponents. This now enables us to get an idea of mixing time. As the PDF is Gaussian, it is of the form

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}\,,$$



(a) $U_0 = 1$, 1 mode

(b) $U_0 = 1$, 50 modes

(c) $U_0 = 5$, 1 mode

(d) $U_0 = 5$, 20 modes

Figure 5.8: Plots of the Normalised PDF of the finite time Lyapunov exponents. Each colour represents a different time (in seconds).

where $x = \lambda$, $\mu = \bar{\lambda}$ and $\sigma = \sqrt{\nu/t}$. Therefore the equation becomes

$$P(\lambda, t) = \frac{\sqrt{t}}{\sqrt{2\pi\nu}} \mathrm{e}^{-\frac{t(\lambda-\bar{\lambda})^2}{2\nu}} . \tag{5.4}$$

Antonsen et al.[2] derived an expression for the asymptotic form of the variance:

$$\langle \theta^2 \rangle \sim \int_0^\infty \mathrm{e}^{-\lambda t} P(\lambda, t) \mathrm{d}\lambda . \tag{5.5}$$

$\langle \cdot \rangle$ is the average over a space, θ is the consentration of a passive scalar of mean 0 and $\langle \theta^2 \rangle$ is the variance which measures fluctuations. We now substitute (5.4) into the (5.5) to get

$$\langle \theta^2 \rangle \sim \int_0^\infty \frac{\sqrt{t}}{\sqrt{2\pi\nu}} \mathrm{e}^{-\lambda t} \mathrm{e}^{-\frac{t(\lambda-\bar{\lambda})^2}{2\nu}} \mathrm{d}\lambda .$$

We integrate this using the saddle point method. This is done by taking the exponential term and factorising the $-t$ to get

$$\mathrm{e}^{-t(\lambda + \frac{(\lambda-\bar{\lambda})^2}{2\nu})} . \tag{5.6}$$

The e^{-t} term is very small as the time t is arbitrary and the integral is dominated by the smallest λ (λ_s). Therefore we need to minimise

$$\lambda + \frac{(\lambda - \bar{\lambda})^2}{2\nu} \tag{5.7}$$

with respect to λ to find λ_s. We then put λ_s into equation (5.6) which is approximately $\langle \theta^2 \rangle$. Therefore to minimise equation (5.7) we differentiate w.r.t λ and set the result equal to 0 to find λ_s. We get

$$1 + \frac{\lambda_s - \bar{\lambda}}{\nu} = 0 ,$$

which rearranges to get

$$\lambda_s = \bar{\lambda} - \nu .$$

If $\bar{\lambda} > \nu$ then λ_s is positive. However if $\bar{\lambda} < \nu$ then λ_s is negative, but λ cannot be negative as it is the positive Lyapunov exponent. If $\bar{\lambda} < \nu$ we have to use $\lambda_s = 0$ as the minimum value and hence there are two cases. If $\bar{\lambda} > \nu$ we substitute $\lambda = \bar{\lambda} - \nu$ into equation (5.6) to get

$$\langle \theta^2 \rangle \sim \mathrm{e}^{-t(\bar{\lambda} - \nu + \frac{(\bar{\lambda}-\nu-\bar{\lambda})^2}{2\nu})} = \mathrm{e}^{-(\bar{\lambda} - \frac{1}{2}\nu)t} = \mathrm{e}^{-\gamma_2 t} ,$$

where $\gamma_2 = \bar{\lambda} - \frac{1}{2}\nu$ is the decay rate of the scalar variance. However if $\bar{\lambda} < \nu$ we substitute $\lambda = 0$ into (5.6) to get

$$\langle \theta^2 \rangle \sim \mathrm{e}^{-t(0 + \frac{(0-\bar{\lambda})^2}{2\nu})} = \mathrm{e}^{-(\frac{\bar{\lambda}^2}{2\nu})t} = \mathrm{e}^{-\gamma_2 t} ,$$

U_0	Modes	$\bar{\lambda}$ vs ν	$\bar{\lambda}^{-1}$	Mixing Time
1	1	$\bar{\lambda} > \nu$	36 s	50 s
1	50	$\bar{\lambda} < \nu$	24 s	54 s
5	1	$\bar{\lambda} > \nu$	10 s	16 s
5	20	$\bar{\lambda} < \nu$	9 s	25 s

Table 5.2: Table of mixing times.

where $\gamma_2 = \frac{\bar{\lambda}^2}{2\nu}$ is the decay rate of the scalar variance. To summarise, we have the following two cases:

$$\gamma_2 = \begin{cases} \bar{\lambda} - \frac{1}{2}\nu\,, & \bar{\lambda} > \nu\,; \\ \frac{\bar{\lambda}^2}{2\nu}\,, & \bar{\lambda} < \nu\,. \end{cases} \tag{5.8}$$

For both of the cases, γ_2 has dimension s^{-1} and the "mixing time" is approximately γ_2^{-1} seconds. Thus ν has a negative affect on mixing and hence it is desired that ν is as small as possible. Therefore, although we want $\bar{\lambda}$ to be as large as possible, we need ν not to be too large. We now use the values from table 5.1 to calculate the mixing times, which are in table 5.2.

Table 5.2 shows that for $U_0 = 5$, the mixing time is a lot less than for $U_0 = 1$. In addition the 1 mode system (sine wave) seems to mix better than the high mode system (step function). We are more interested in the higher mode system than the 1 mode system as a step function is more likely to be used in a real micromixer. Interestingly $\bar{\lambda}^{-1}$ is less for the higher mode systems. It is the larger ν values that increase the mixing time for higher modes. For both $U_0 = 1$ and $U_0 = 5$, the ν value adds a significant amount of time to the mixing. This is a concern as ν would need to be much lower for a really efficient mixer.

For the higher mode system the mixing time is less than half for $U_0 = 5$ than it is for $U_0 = 1$. This is promising, as a velocity of $U_0 = 5$ is a more common velocity to be used in microchannels. For $U_0 = 5$, the time of 25 s is a vast improvement on the time to mix due to diffusion only which is 100 s. Thus the 3-5 mixer with $U_0 = 5$ takes 25% of the time to mix. That means at a speed of 500μm/s, the 3-5 mixer will mix in a distance of 1.25 cm which is a good improvement on 5 cm. In addition the $U_0 = 1$ mixer will mix in a distance of 0.54 cm rather than 1 cm when relying on diffusion only. Furthermore the distances are calculated by using the mean velocity on the floor of the channel. The mean velocity of the whole fluid will always be less than this, so the mixing distance will be less than the values found.

Chapter 6

Conclusion

6.1 Accomplishments

Throughout this project the aim has been to model flow in a microchannel analytically. We used electro-osmotically induced flow from a herringbone pattern on the floor of the channel. We then continued to find some of the mixing properties of these flows. We have only concentrated on simple configurations and have found some good results. We successfully manipulated the governing equations of the flow and imposed the boundary conditions to get an analytical solution for the velocity field. We then created programs using C++ which can trace trajectories of particles, output data files for Poincaré sections and data files for the dispersion of a square of particles as time evolves. We analysed this data, including using Matlab to create a mix scoring program for the Poincaré section.

The results of the analysis found that the 3-5 mixer (i.e. a Herringbone with $\alpha = 3$, $\beta = 5$, $a = 0$, $h = 0.25$) was the best overall mixer from the 150 mixers we tested with $U_0 = 1$. The visualisation of the Poincaré sections agree with the mixing scoring and the 3-5 mixer does look good. In addition for a faster velocity of $U_0 = 5$ the 3-5 mixer looks very similar. We could have studied some more configurations however we would have struggled with the accuracy level of the mixing scoring program to find any significantly better configurations. In addition we got the fairly conclusive result that the 8 best mixers were variations of 3-5 or 4-5 configurations and hence the direction of the herringbone doesn't matter.

Further analysis of the dispersion of particles showed that the 3-5 mixer has a constant mean residence time and the x-direction dispersion of the particles is diffusive only. This suggests that the 3-5 mixer potentially is a good mixer. It provides good mixing in the cross-section without too

much spreading along the channel.

Finally we studied the finite time Lyapunov exponents of the flow. We picked a small square of 101×101 particles and modelled the evolution of the FTLEs up to $t = 500\,\mathrm{s}$. As well as the step function herringbone pattern we studied the flow due to a sine wave in the herringbone pattern. The normalised PDF of the Lyapunov exponents for each configuration is Gaussian (or very similar), which enables us to look at the decay rate of variance to find the mixing time. We found that for $U_0 = 5$ the mixing time is a lot less than for the $U_0 = 1$. For the faster velocity, the 3-5 mixer reduces the mixing time to 25% of the mixing time due to diffusion alone. In addition we found that although the $\bar{\lambda}$ value was higher for the faster velocity, so was ν which has a negative effect on mixing. For velocities which are faster than $500\,\mu\mathrm{m/s}$ it is possible that ν could become too large and good mixing could become harder to create.

Overall this project has shown that it is possible to successfully find an analytical solution for the velocity field of flow in a channel, rather than using a numerical method like in previous studies. We have found a mixer which makes mixing 75% more efficient than diffusion only. The 3-5 mixer could be manufactured with today's technology, especially as the coating required to enable electro-osmosis to drive the flow would only need to be placed on "forwards" herringbones. In past studies better mixers have been found using staggered herringbones as well as other methods but this was not using an analytical velocity field. From this project we could now develop the analytical model further for more complicated systems that may produce better mixers.

6.2 Further Work

We have only modelled some simple configurations of microchannels here. There are many studies that could follow from this project. The next step would be to keep the herringbone configuration, but to impose the no-slip boundary conditions on the side walls of the channel. To do this we would need to modify the equations for the herringbone on floor of the channel so that they induce no flow at the walls. One method of doing this may be to use a cosh function.

A large improvement would come from extending the model to use staggered herringbones. We found that we need to have $a = 0$ because otherwise one side of the flow dominates and mixing reduces. However, with a staggered herringbone we would not get this problem. In previous studies the optimal a was found to be 1/3 and -1/3 for the staggered system [9]. An analytical solution for the velocity field of the staggered herringbone system would enable us to accurately model the flow for some of the best mixers that have been currently found. This would enable us to try many

different configurations and would probably enable us to find some very good new mixers.

Another method that could be furthered, would be to develop a better mix scoring system. The method we used was reasonably crude and although it gave us fairly conclusive results, it wasn't accurate enough to find better mixers as it couldn't be relied upon beyond 2 decimal places for the mix score. In addition the Lyapunov study could be furthered. For distributions that are not quite Gaussian we could try to fit better curves and this may enable us to learn more about the mixing time.

Finally, we didn't study the sine wave herringbone (i.e. 1 Fourier mode) configuration very much. We found that the distribution of the Lyapunov exponents was more Gaussian than for the step function. Also the ν values were much lower for the sine wave system and hence the mixing time was less. Although it might be difficult to manufacture compared to the step function herringbone system, we may find that using a sine wave produces better mixing and this would definitely be a worth while study.

Bibliography

[1] A. Ajdari, "Electro-Osmosis on Inhomogeneously Charged Surfaces", Physical Review Letters, 75, 755-758 (1995)

[2] T.M. Antonsen, Jr., Z. Fan, E. Ott, E. Garcia-Lopez. "The Role of Chaotic Orbits in the Determination of Power Spectra of Passive Scalars", Physics of Fluids, 8, 3094-3104 (1996)

[3] H. Aref, "Stirring by Chaotic Advection", Journal of Fluid Mechanics, 143, 1-21 (1984)

[4] K. G. Denbigh, J. C. R. Turner "Chemical Reactor Theory", 3rd edition, CUP (1984)

[5] S. Hong, J-L. Thiffeault, L. Fréchette, V. Modi "Numerical study of mixing in microchannels with patterned zeta potential surfaces", IMECE2003-41912, (2003)

[6] T. J. Johnson, D. Ross and L. E. Locascio "Rapid Microfluidic Mixing", Anal. Chem.2002, 74, 45-51, (2002)

[7] G. E. Karniadakis and A. Beskok, "Micro flows: Fundamentals and Simulation", Springer-Verlag (2002)

[8] J. M. Ottino, "The Kinematics of mixing: Stretching, chaos, and transport", (1997)

[9] A. D. Stroock, S. K. W. Dertinger, A. Ajdari, I. Mezic, H. A. Stone, G. M. Whitesides, "Chaotic Mixer for Microchannels", Science, 295, 647-651 (2002)

[10] A. D. Stroock, S. K. Dertinger, G. M. Whitesides, and A. Ajdari, "Patterning Flows Using Grooved Surfaces", Anal. Chem.2002, 74, 5306-5312, (2002)

[11] A. D. Stroock, J. M. K. Ng, I. Gitlin, G. M. Whitesides, "Components for integrated poly(dimethylsiloxane) microfluidic systems", Electrophoresis, 23, 34613473, (2002)

[12] H. Wang, P. Iovenitti, E. Harvey, S. Masood "Optimizing layout of obstacles for enhanced mixing in microchannels", Smart Materials and Structures, 11, 662-667, (2002)

Appendix A

Maple Code

A.1 Constants Calculator

```
restart;
#Throughout the worksheet the 1 subscript refers to y<a and the 2 subscript
#refers to y>a.
#Particular integral:
Q:=-6*k*Uk/((h^2)*(gamma^2+k^2))*cos(gamma*(y-a)-chi):
#Defining P(hat):
P[1]:=A[1]*sinh(k*y)+B[1]*cosh(k*y)+Q:
P[2]:=A[2]*sinh(k*y)+B[2]*cosh(k*y)+Q:
P[1]:=eval(P[1],{gamma=alpha}):
P[2]:=eval(P[2],{gamma=-beta}):
#Boundary Conditions:
BC:=eval(diff(P[1],y),y=-l/2)=0,
    eval(P[1],y=a)=eval(P[2],y=a),
    eval(diff(P[2],y),y=l/2)=0,
    eval(diff(P[1],y),y=a)=eval(diff(P[2],y),y=a):
#Solving for Constants:
assign(solve({BC},{A[1],A[2],B[1],B[2]}));
#Constants can be see here by printing them out.
#Defining P0:
P1:=cos(k*x)*eval(P[1],chi=0)+cos(k*x+Pi/2)*eval(P[1],chi=Pi/2):
P2:=cos(k*x)*eval(P[2],chi=0)+cos(k*x+Pi/2)*eval(P[2],chi=Pi/2):
#Defining U:
U[gen]:=Uk*sin(k*x+phi)*cos(gamma*(y-a)+chi):
U[1]:=eval(U[gen],{gamma=alpha,chi=0,phi=0})+eval(U[gen],{gamma=alpha,chi=Pi/2,phi=Pi/2}):
U[2]:=eval(U[gen],{gamma=-beta,chi=0,phi=0})+eval(U[gen],{gamma=-beta,chi=Pi/2,phi=Pi/2}):
#Solving for u,v,w
v1:=(1/2)*simplify(diff(P1,y)*z*(z-h)):
v2:=(1/2)*simplify(diff(P2,y)*z*(z-h)):
u1:=(h-z)*(U[1]/h-diff(P1,x)*z/2):
u2:=(h-z)*(U[2]/h-diff(P2,x)*z/2):
w1:=-(z/(h^2))*((h-z)^2)*diff(U[1],x):
w2:=-(z/(h^2))*((h-z)^2)*diff(U[2],x):
#Setting parameters:
l:=1:
```

```
alpha:=1:
beta:=2:
k:=1:
h:=0.25:
a:=1/4:
Uk:=4/Pi:
with(plots):
#Plotting results. Set F1,F2 to be (P1,P2), (u1,u2), (v1,v2) or (w1,w2) to
#plot graphs. In d1,d2 the x and z positions can be changed.
F1:=P1:
F2:=P2:
d1:=plot(eval(F1,{x=Pi,z=h/2}),y=-l/2..a):
d2:=plot(eval(F2,{x=Pi,z=h/2}),y=a..l/2,color=blue):
display(d1,d2, labels=["y","P0"],title="P0_vs_y_x=0_z=h",thickness=3,
       font=[times, roman, 20], labelfont=[times, roman, 20],
       titlefont=[times, roman, 20], tickmarks=[4, 4]);
```

Appendix B

C++ Code

In this section all of the programs are adapted versions of Dr Jean-Luc Thiffeault's code, some of which was used for the paper "Numerical study of mixing in microchannels with patterned zeta potential surfaces" [5]. However now our analytical solution for the velocity field is used.

B.1 velmmix.hpp

```
#include <math.hpp>
#include <matrix.hpp>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdexcept>

using namespace std;
using namespace jlt;

template<class Real>
class Velmmix {
private:
  // Problem parameters.
  const Real a, al, be;
  vector<Real> Uks, Ukc;
  const Real h, L, l;
  const int n, nkmax;
  vector<Real> Ac1, Bc1, Ac2, Bc2;
  vector<Real> As1, Bs1, As2, Bs2;

private:
  Real Sech(const Real x) const { return 1/Cosh(x); }
  Real Csch(const Real x) const { return 1/Sinh(x); }

  // These were cut&pasted from soln in herringbone.nb.

  Real A1f(const Real k, const Real chi) const
  {
```

```
  return
    ((-3*Csch((k*l)/2)*((al + be)*(k*k + al*be)*
                        Cosh(a*k - (k*l)/2)*Sin(chi) -
                        k*k*al*Sin((-a - l/2)*al + chi) -
                        al*be*be*Sin((-a - l/2)*al + chi) -
                        k*k*be*Sin(a*be - (l*be)/2 + chi) -
                        al*al*be*Sin(a*be - (l*be)/2 + chi) -
                        k*(al*al
                           - be*be)*Cos(chi)*Sinh(a*k - (k*l)/2)))
     / (h*h*(k*k + al*al)*(k*k + be*be)));
}

Real B1f(const Real k, const Real chi) const
{
  return
    ((-3*Sech((k*l)/2)*((al + be)*(k*k + al*be)*
                        Cosh(a*k - (k*l)/2)*Sin(chi) +
                        k*k*al*Sin((-a - l/2)*al + chi) +
                        al*be*be*Sin((-a - l/2)*al + chi) -
                        k*k*be*Sin(a*be - (l*be)/2 + chi) -
                        al*al*be*Sin(a*be - (l*be)/2 + chi)
                        - k*(al*al
                             - be*be)*Cos(chi)*Sinh(a*k - (k*l)/2)))
     / (h*h*(k*k + al*al)*(k*k + be*be)));
}

Real A2f(const Real k, const Real chi) const
{
  return
    ((3*Csch((k*l)/2)*(-((al + be)*(k*k + al*be)*
                         Cosh((k*(2*a + l))/2)*Sin(chi)) +
                        k*k*al*Sin((-a - l/2)*al + chi) +
                        al*be*be*Sin((-a - l/2)*al + chi) +
                        k*k*be*Sin(a*be - (l*be)/2 + chi) +
                        al*al*be*Sin(a*be - (l*be)/2 + chi) +
                        k*(al*al
                           - be*be)*Cos(chi)*Sinh((k*(2*a + l))/2)))
     / (h*h*(k*k + al*al)*(k*k + be*be)));
}

Real B2f(const Real k, const Real chi) const
{
  return
  ((3*Sech((k*l)/2)*((al + be)*(k*k + al*be)*
                     Cosh((k*(2*a + l))/2)*Sin(chi) -
                     k*k*al*Sin((-a - l/2)*al + chi) -
                     al*be*be*Sin((-a - l/2)*al + chi) +
                     k*k*be*Sin(a*be - (l*be)/2 + chi) +
                     al*al*be*Sin(a*be - (l*be)/2 + chi) -
                     k*(al*al
                        - be*be)*Cos(chi)*Sinh((k*(2*a + l))/2)))
   / (h*h*(k*k + al*al)*(k*k + be*be)));
}
```

```
  void fillAB() {
    // Fill A, B vectors using the corresponding functions.
    // Avoids repeated function calls, since the A,B constants don't
    // change for different values of x and y.
    // (A,B)c and (A,B)s are the cosine and sine versions, respectively,
    // and 1 and 2 refer to the two regions (left and right).

    // The k=0 mode requires special treatment.
    // The A,B[0] variables are unused.

    for (int nk = 1; nk < nkmax; ++nk) {
      Real k = 2*M_PI*nk/L;
      Real cs = -M_PI_2;

      Ac1[nk] = A1f(k,0);
      Bc1[nk] = B1f(k,0);
      Ac2[nk] = A2f(k,0);
      Bc2[nk] = B2f(k,0);

      As1[nk] = A1f(k,cs);
      Bs1[nk] = B1f(k,cs);
      As2[nk] = A2f(k,cs);
      Bs2[nk] = B2f(k,cs);
    }
  }

public:
  Velmmix(Real a_, Real al_, Real be_,
          vector<Real> Uks_, vector<Real> Ukc_,
          Real h_ = 0.25, Real L_ = 2*M_PI, Real l_ = 1) :
    a(a_), al(al_), be(be_), Uks(Uks_), Ukc(Ukc_),
    h(h_), L(L_), l(l_), n(3), nkmax(Uks.size()),
    Ac1(nkmax), Bc1(nkmax), Ac2(nkmax), Bc2(nkmax),
    As1(nkmax), Bs1(nkmax), As2(nkmax), Bs2(nkmax)
  {
    assert(Uks.size() == Ukc.size());
    assert(Uks[0] == 0);        // The k=0 mode is in Ukc[0].

    fillAB();
  }

  void uvwk(const int nk, Real, const vector<Real>& X, vector<Real>& V)
  {
    // X = (x,y,z), V = (u,v,w).

    // Returns the solution for k-th mode.

    Real x, y, z, u = 0, v = 0, w = 0;

    Real k = 2*M_PI*nk/L;

    Real xi, Ac, Bc, As, Bs;
```

```
Real h2 = h*h, k2 = k*k;

x = X[0];
y = X[1];
z = X[2];

if (nk == 0) {
  V[0] = Ukc[0]*(1 - z/h);
  V[1] = 0;
  V[2] = 0;
  return;
}

if (y < a) {
  // Use the Region 1 (left) values.
  xi = al;
  Ac = Ac1[nk];
  Bc = Bc1[nk];
  As = As1[nk];
  Bs = Bs1[nk];
} else {
  // Use the Region 2 (right) values.
  xi = -be;
  Ac = Ac2[nk];
  Bc = Bc2[nk];
  As = As2[nk];
  Bs = Bs2[nk];
}
Real xi2 = xi*xi;
Real Coshky = Cosh(k*y), Sinhky = Sinh(k*y);
Real Cosy = Cos(xi*(y-a)), Siny = Sin(xi*(y-a));
Real Cosx = Cos(k*x), Sinx = Sin(k*x);

// The sin part.
if (Uks[nk] != 0) {
  u += Uks[nk]*0.5*(h-z)*
    ((k*z*Ac*Coshky + k*z*Bc*Sinhky
      + 2*(k2*(h - 3*z) + h*xi2)*Cosy/(h2*(k2 + xi2))) * Sinx
    +
     (k*z*As*Coshky + k*z*Bs*Sinhky
      + 2*(k2*(h - 3*z) + h*xi2)*Siny/(h2*(k2 + xi2))) * Cosx);
  v += Uks[nk]*0.5*k*z*(h-z)*
    ((As*Sinhky + Bs*Coshky - 6*xi*Cosy/(h2*(k2 + xi2))) * Sinx
    -
     (Ac*Sinhky + Bc*Coshky + 6*xi*Siny/(h2*(k2 + xi2))) * Cosx);
  w += Uks[nk]*k*z*(h-z)*(h-z)*(Siny*Sinx - Cosy*Cosx)/h2;
}

// The cos part.
if (Ukc[nk] != 0 ) {
  u += Ukc[nk]*0.5*(h-z)*
    (-(k*z*As*Coshky + k*z*Bs*Sinhky
       + 2*(k2*(h - 3*z) + h*xi2)*Siny/(h2*(k2 + xi2))) * Sinx
```

```
      +
       (k*z*Ac*Coshky + k*z*Bc*Sinhky
        + 2*(k2*(h - 3*z) + h*xi2)*Cosy/(h2*(k2 + xi2))) * Cosx);
    v += Ukc[nk]*0.5*k*z*(h-z)*
      ((Ac*Sinhky + Bc*Coshky + 6*xi*Siny/(h2*(k2 + xi2))) * Sinx
      +
       (As*Sinhky + Bs*Coshky - 6*xi*Cosy/(h2*(k2 + xi2))) * Cosx);
    w += Ukc[nk]*k*z*(h-z)*(h-z)*(Siny*Cosx + Cosy*Sinx)/h2;
  }

  V[0] = u;
  V[1] = v;
  V[2] = w;
}

void operator()(Real t, const vector<Real>& X, vector<Real>& V)
{
  // X = (x,y,z), V = (u,v,w).

  Real x, y, z;
  vector<Real> Vk(n);

  x = X[0];
  y = X[1];
  z = X[2];
  V[0] = 0;
  V[1] = 0;
  V[2] = 0;

  // Check if within bounds in y and z.
  if (y < -l/2 || y > l/2 || z < 0 || z > h) {
    throw(range_error("Out of domain range error in Velmmix::operator()."));
    return;
  }

  for (int nk = 0; nk < nkmax; ++nk) {
    uvwk(nk,t,X,Vk);
    V[0] += Vk[0];
    V[1] += Vk[1];
    V[2] += Vk[2];
  }
}

// Jacobian matrix of the equation.
void Jacobian(Real t, const vector<Real>& V, const vector<Real>& Vp,
              const Real scale, jlt::matrix<Real>& Jac)
{
  // Returns the forward-differenced Jacobian, scaled by a
  // factor "scale", which is usually the stepsize.
  vector<Real> V1(V), Vp1(n);
  Real eps = 1.e-8;
  Real dh = eps*scale;
```

```
    for (int j = 0; j < n; ++j)
      {
        // Forward-difference.
        V1[j] = V[j] + dh;
        // When very near the edge, it is possible that operator() may
        // return a range_error exception.  Catch that and return the
        // backward difference instead.
        try {
          this->operator()(t, V1, Vp1);
          for (int i = 0; i < n; ++i) {
            Jac(i,j) = (Vp1[i] - Vp[i])/dh;
          }
        } catch(range_error& oor) {
          // We've gone past the end of the domain.
          // Use backward-difference.
          V1[j] = V[j] - dh;
          this->operator()(t, V1, Vp1);
          for (int i = 0; i < n; ++i) {
            Jac(i,j) = (Vp1[i] - Vp[i])/(-dh);
          }
        }
        V1[j] = V[j];
      }
    /*
    // Enforce incompressibility by adjusting last element.
    Real div = 0;
    for (int i = 0; i < n-1; ++i) div += Jac(i,i);
    Jac(n-1,n-1) = -div;
    */
  }

  int size() const { return n; }
};
```

B.2 velmmix_test.cpp

```
#include <fstream>
#include <sstream>
#include <string>
#include <iomanip>
#include <vector>
#include <rodent/explicitrk.hpp>
#include <stlio.hpp>
#include <math.hpp>

#include "velmmix.hpp"
#include <iostream>

int main()
{
  const int n = 3;
```

```
  vector<double> x(n);
  vector<double> v(n);

  double a = 0;
  double alpha = 3;
  double beta = 5;
  double L = 2*M_PI;
  const int nmodes = 50;
  vector<double> Uks(nmodes+1), Ukc(nmodes+1);

  // Fourier sine series for step function.

  // Constant mode.
    Ukc[0] = 5;

    //Stepfunction variable
  double SF = 1;

  const double lambda = .001;   // Smoothing to avoid Gibb's phenomenon.
  for (int k = 0; k <= nmodes; ++k) {
    // Only include odd modes.
    if (k % 2 == 1) Uks[k] = SF * Ukc[0] * 4/(k*M_PI) * Exp(-lambda*(k-1)*(k-1));
  }


  Velmmix<double> vel(a, alpha, beta, Uks, Ukc);

  x[0] = 0;
  x[1] = a;
  x[2] = 0;

  const int N = 1000;
  for (int i = 0; i < N; ++i) {
    x[0] = i*L/N;
    vel(0,x,v);
    cout << x[0] << '\t';
    cout << x[1] << '\t';
    cout << x[2] << '\t' << '\t';
    cout << v[0] << '\t';
    cout << v[1] << '\t';
    cout << v[2] << endl;
  }

  return 0;
}
```

B.3 velmmix_particle.cpp

```
#include <vector>
#include <rodent/explicitrk.hpp>
#include "velmmix.hpp"
```

```
// Calculate a particle trajectory.

using namespace rodent;
//using namespace jlt;

int main()
{
  typedef double Real;

  const int n = 3;
  vector<Real> x(n);
  vector<Real> v(n);

  double a = 0.25;
  double alpha = 1;
  double beta = 2;
  const int nmodes = 50;
  vector<double> Uks(nmodes);
  vector<double> Ukc(nmodes);

  // Fourier sine series for step function.
  const double lambda = .005;   // Smoothing to avoid Gibb's phenomenon.
  for (int k = 0; k < nmodes; ++k) {
    // Only include odd modes.
    if (k % 2 == 1) Uks[k] = 4/(k*M_PI) * Exp(-lambda*k*k);
  }
  // Constant mode.
  Ukc[0] = 5;

  Velmmix<double> vel(a, alpha, beta, Uks, Ukc);

  ofstream fout("traj.dat");

  // Initial condition:
  x[0] = 0;
  x[1] = 0.15;
  x[2] = 0.1;

  AdaptiveRKCashKarp<Velmmix<Real>, vector<Real> >
    velint(vel, 0, x, .01, 0, 1e-10);

  const int N = 500;
  Real dt = .1;
  Real t = 0;

  fout << t << "\t" << x << endl;
  for (int i = 1; i <= N; ++i) {
    t = i*dt;
    try {
      velint(t,x);
    } catch(range_error& oor) {
      // We've gone past the end of the domain.
```

```
      break;
    }
    fout << t << "\t" << x << endl;
  }

  return 0;
}
```

B.4 velmmix_section.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <iomanip>
#include <vector>
#include <rodent/explicitrk.hpp>
#include <stlio.hpp>
#include <math.hpp>
#include "velmmix.hpp"

using namespace rodent;
using namespace jlt;


// An InitialConfiguration class provides some set of initial conditions
// and a method for looping through them.
template<class T>
class InitialConfiguration
{
public:
  // InitialConfiguration();

  // Next returns true until we run out of particles.
  virtual bool next(vector<T>& v) = 0;

  // The total number of particles.
  virtual int total() const = 0;
};


template<class T>
class PointInitialConfiguration : public InitialConfiguration<T>
{
  int N;
  const int Nmax;
  T x, y, z;
public:
  PointInitialConfiguration(T x_, T y_, T z_) :
    N(0), Nmax(1),
    x(x_), y(y_), z(z_)
```

```
  {
  }

  bool next(vector<T>& v)
  {
    if (N++ < Nmax) {
      v[0] = x;
      v[1] = y;
      v[2] = z;
      return true;
    } else {
      return false;
    }
  }

  int total() const { return Nmax; }
};


template<class T>
class LineInitialConfiguration : public InitialConfiguration<T>
{
  int N;
  const int Nmax;
  T y0, z0, y1, z1;
  T dy, dz;
public:
  LineInitialConfiguration(T y0_, T z0_, T y1_, T z1_, int N_) :
    N(0), Nmax(N_),
    y0(y0_), z0(z0_), y1(y1_), z1(z1_),
    dy((y1-y0)/(Nmax-1)), dz((z1-z0)/(Nmax-1))
  {
  }

  bool next(vector<T>& v)
  {
    if (N < Nmax) {
      v[0] = 0;
      v[1] = y0 + dy*N;
      v[2] = z0 + dz*N++;
      return true;
    } else {
      return false;
    }
  }

  int total() const { return Nmax; }
};


template<class T>
class SquareInitialConfiguration : public InitialConfiguration<T>
{
```

```
  int Ny, Nz;
  int Nym, Nzm;
  const int Nmax;
  T y0, z0, y1, z1;
  T dy, dz;
public:
  SquareInitialConfiguration(T y0_, T z0_, T y1_, T z1_, int Ny_, int Nz_) :
    Ny(0), Nz(0), Nym(Ny_), Nzm(Nz_), Nmax(Nym*Nzm),
    y0(y0_), z0(z0_), y1(y1_), z1(z1_),
    dy((y1-y0)/(Nym-1)), dz((z1-z0)/(Nzm-1))
  {
  }

  bool next(vector<T>& v)
  {
    if (Ny >= Nym) return false;

    T y = y0 + dy*Ny;
    T z = z0 + dz*Nz++;

    if (Nz >= Nzm) { Nz = 0; ++Ny; }

    v[0] = 0;
    v[1] = y;
    v[2] = z;

    return true;
  }

  int total() const { return Nmax; }
};


int main()
{
  typedef double Real;

  const int n = 3;
  double a = 0;
  double alpha = 3;
  double beta = 5;
  double L = 2*M_PI;
  const int nmodes = 50;
  vector<double> Uks(nmodes+1);
  vector<double> Ukc(nmodes+1);


 // Constant mode.
  Ukc[0] = 1;

  //Stepfunction variable
  double SF = 1;
```

```
// Fourier sine series for step function.
const double lambda = .005;   // Smoothing to avoid Gibb's phenomenon.
for (int k = 0; k <= nmodes; ++k) {
  // Only include odd modes.
  if (k % 2 == 1) Uks[k] = SF * Ukc[0] * 4/(k*M_PI) * Exp(-lambda*(k-1)*(k-1));
}


Velmmix<double> vel(a, alpha, beta, Uks, Ukc);

const Real x_max = L;

cerr << "Trajectories..." << endl;

const int Nsect = 2;          // Number of sections in x.
vector<Real> x_sect(Nsect);


// Rescale
for(int i = 0; i < Nsect; ++i) x_sect[i] = i*x_max/Nsect;

// Vector of output streams.
ofstream *sect_out[Nsect];

for (int sect = 0; sect < Nsect; ++sect) {
  // Form file name.
  string dataprefix = "section";
  ostringstream params;
  params << "a";
  params << alpha;
  params << "b";
  params << beta;
  params << "_a=";
  params << a;
  params << "_U0=";
  params << Ukc[0];
  dataprefix = dataprefix + "_" + params.str() + "_Lx";

      ostringstream ostr;
  ostr.precision(3);
  int t_width = 5;
  ostr.setf(ios::fixed);
  ostr.fill('0');             // Pad with zeros to the left.
  ostr << setw(t_width) << x_sect[sect]/x_max;
  string sect_file = dataprefix + ostr.str() + ".dat";

  // Open the file.
  sect_out[sect] = new ofstream(sect_file.c_str());

  sect_out[sect]->precision(10);
  sect_out[sect]->setf(ios::scientific);
}
```

```
//Intial configuration
PointInitialConfiguration<Real> init(0,-0.0892,0.1920);

vector<vector<Real> > x(init.total(),vector<Real>(n));

vector<Real> dummy(n,.1), v(n);
AdaptiveRKCashKarp<Velmmix<Real>, vector<Real> >
  velint(vel, 0, dummy, .01, 0, 1e-9);

int p = 0;

while (init.next(x[p])) {

  cerr << "Particle " << p+1 << endl;

  Real Nperiods = 1000;       // Number of periods to go through.
  unsigned long int period0 = 0, period;
  Real x_final = Nperiods*x_max;
  Real dt = .1;               // Step size.
  Real t = 0;                 // Initial time.
  Real t_max = 1e9;           // Max time to integrate.

  unsigned long int i = 0;

  vector<Real> dx0_sect(Nsect), dx_sect(Nsect);
  for (int sect = 0; sect < Nsect; ++sect) {
    dx0_sect[sect] = x[p][0] - x_sect[sect];
  }

  try {
    velint.Restart(t,x[p],.01);
  } catch(range_error& oor) {
    // We've started outside the domain.
    cerr << oor.what() << endl;
    exit(1);
  }

  while (x[p][0] <= x_final && t <= t_max) {

    t = (++i)*dt;
    try {
      velint(t,x[p],v);
    } catch(range_error& oor) {
      // We've somehow left the domain.
      cerr << oor.what() << endl;
      exit(1);
    }

    // Did we cross over to a new period?  Then all the dx0_sect change sign.
    period = (int)(x[p][0]/x_max);
    if (period != period0) {
      for (int sect = 0; sect < Nsect; ++sect) {
        dx0_sect[sect] = -dx0_sect[sect];
```

```
        }
        period0 = period;
      }

      for (int sect = 0; sect < Nsect; ++sect) {
        dx_sect[sect] = Mod(x[p][0],x_max) - x_sect[sect];

        // Crossed the next section?
        if (dx_sect[sect]*dx0_sect[sect] <= 0) { // && Abs(dx_sect) <= 1e-2) {
          // Integrate back to section first.

          // The distance to the section.
          Real dx = min(Abs(dx_sect[sect]),Abs(x_max - dx_sect[sect]));
          Real dt_sect = Abs(dx/v[0]);          // Approx. how much time?
          velint(t-dt_sect,x[p]);               // One Newton iteration.

          // Output position.
          *sect_out[sect] << t-dt_sect << "\t" << x[p] << "\t";
          *sect_out[sect] << v[0] << endl;

          dx0_sect[sect] = dx_sect[sect];

          velint(t,x[p]);                       // Back to where we were.
        }
      }
    }
    ++p;
  }

  return 0;
}
```

B.5 velmmix_dispersion.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <iomanip>
#include <vector>
#include <rodent/explicitrk.hpp>
#include <stlio.hpp>
#include <math.hpp>
#include "velmmix.hpp"

using namespace rodent;
using namespace jlt;


// An InitialConfiguration class provides some set of initial conditions
// and a method for looping through them.
```

```
template<class T>
class InitialConfiguration
{
public:
  // InitialConfiguration();

  // Next returns true until we run out of particles.
  virtual bool next(vector<T>& v) = 0;

  // The total number of particles.
  virtual int total() const = 0;
};


template<class T>
class PointInitialConfiguration : public InitialConfiguration<T>
{
  int N;
  const int Nmax;
  T x, y, z;
public:
  PointInitialConfiguration(T x_, T y_, T z_) :
    N(0), Nmax(1),
    x(x_), y(y_), z(z_)
  {
  }

  bool next(vector<T>& v)
  {
    if (N++ < Nmax) {
      v[0] = x;
      v[1] = y;
      v[2] = z;
      return true;
    } else {
      return false;
    }
  }

  int total() const { return Nmax; }
};


template<class T>
class LineInitialConfiguration : public InitialConfiguration<T>
{
  int N;
  const int Nmax;
  T y0, z0, y1, z1;
  T dy, dz;
public:
  LineInitialConfiguration(T y0_, T z0_, T y1_, T z1_, int N_) :
    N(0), Nmax(N_),
```

```
    y0(y0_), z0(z0_), y1(y1_), z1(z1_),
    dy((y1-y0)/(Nmax-1)), dz((z1-z0)/(Nmax-1))
  {
  }

  bool next(vector<T>& v)
  {
    if (N < Nmax) {
      v[0] = 0;
      v[1] = y0 + dy*N;
      v[2] = z0 + dz*N++;
      return true;
    } else {
      return false;
    }
  }

  int total() const { return Nmax; }
};


template<class T>
class SquareInitialConfiguration : public InitialConfiguration<T>
{
  int Ny, Nz;
  int Nym, Nzm;
  const int Nmax;
  T y0, z0, y1, z1;
  T dy, dz;
public:
  SquareInitialConfiguration(T y0_, T z0_, T y1_, T z1_, int Ny_, int Nz_) :
    Ny(0), Nz(0), Nym(Ny_), Nzm(Nz_), Nmax(Nym*Nzm),
    y0(y0_), z0(z0_), y1(y1_), z1(z1_),
    dy((y1-y0)/(Nym-1)), dz((z1-z0)/(Nzm-1))
  {
  }

  bool next(vector<T>& v)
  {
    if (Ny >= Nym) return false;

    T y = y0 + dy*Ny;
    T z = z0 + dz*Nz++;

    if (Nz >= Nzm) { Nz = 0; ++Ny; }

    v[0] = 0;
    v[1] = y;
    v[2] = z;

    return true;
  }
```

```
  int total() const { return Nmax; }
};


int main()
{
  typedef double Real;

  const int n = 3;
  double a = 0;
  double alpha =3 ;
  double beta = 5 ;
  double L = 2*M_PI;
  const int nmodes = 50;
  vector<double> Uks(nmodes+1);
  vector<double> Ukc(nmodes+1);

  Real max_d = 100;
  for(Real d=0; d<=max_d; ++d)
  {
    //output file
    ofstream fname;
        string dataprefix = "Disp_a3b5_";
    ostringstream params;
        params << d;
        dataprefix = dataprefix + params.str() + ".dat";
        // Open the file.
        fname.open(dataprefix.c_str(),ios::out);
        fname.close();
  }


 // Constant mode.
  Ukc[0] = 1;

  //Stepfunction variable
  double SF = 1;

  // Fourier sine series for step function.
  const double lambda = .005;   // Smoothing to avoid Gibb's phenomenon.
  for (int k = 0; k <= nmodes; ++k) {
    // Only include odd modes.
    if (k % 2 == 1) Uks[k] = SF * Ukc[0] * 4/(k*M_PI) * Exp(-lambda*(k-1)*(k-1));
  }

  Velmmix<double> vel(a, alpha, beta, Uks, Ukc);

  const Real x_max = L;

  cerr << "Trajectories..." << endl;

  //Square initial configuration
  SquareInitialConfiguration<Real>
```

```
  init(-0.45,0.02,0.45,0.23,100,25);


vector<vector<Real> > x(init.total(),vector<Real>(n));

vector<Real> dummy(n,.1), v(n);
AdaptiveRKCashKarp<Velmmix<Real>, vector<Real> >
  velint(vel, 0, dummy, .01, 0, 1e-7);

int p = 0;

while (init.next(x[p]))
{

      cerr << "Particle " << p+1 << endl;

  Real dt = .1;               // Step size.
  Real t = 0;                 // Initial time.
  Real t_max = x_max/Ukc[0];  // Max time to integrate.

  unsigned long int i = 0;

  try
  {
    velint.Restart(t,x[p],.01);
  }
  catch(range_error& oor)
  {
    // We've started outside the domain.
    cerr << "At Restart: " << oor.what() << endl;
    exit(1);
  }

      Real d=0;
  while (d <= max_d)
  {
      if (d!=0) //Only do if d is not 0 as don't want to integrate if d=0.
      {
    while (t <= d*t_max)
    {
      t = (++i)*dt;
      try
      {
        velint(t,x[p],v);
      }
      catch(range_error& oor)
      {
            // We've somehow left the domain.
            cerr << oor.what() << endl;
            exit(1);
      }
    }
  }
```

```
    else //Just get velocity for printing to file
    {
                velint(t,x[p],v);
        }

        //output file
    ofstream fname;
    string dataprefix = "Disp_a3b5_";
    ostringstream params;
    params << d;
        dataprefix = dataprefix + params.str() + ".dat";
    // Open the file.
        fname.open(dataprefix.c_str(),ios::app);
    // Output position.
        fname << p << "\t" << t << "\t" << x[p] << "\t" << v << endl;
    //Close file
    fname.close();

        //Increase d by 1
        ++d;
    }
    ++p;
  }

  return 0;
}
```

B.6 velmmix_lyapsquare.cpp

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include <rodent/explicitrk.hpp>
#include <mathvector.hpp>
#include <mathmatrix.hpp>
#include "LyapunovFlowQR.hpp"
#include "velmmix.hpp"

using namespace rodent;
using namespace jlt;

int main()
{
  typedef double Real;

  const int fbtime = 1;
  Real t0 = 0, t = t0, t_max = 500, dt = 6.3, acc = 1e-8;

  Real a = 0;
  Real alpha = 3;
  Real beta = 5;
```

```
const int nmodes = 20;
vector<Real> Uks(nmodes+1), Ukc(nmodes+1);


// Fourier sine series for step function.

// Constant mode.
Ukc[0] = 1;

//Stepfunction Multiplier
double SF=1;

const double lambda = .01;    // Smoothing to avoid Gibb's phenomenon.
for (int k = 0; k <= nmodes; ++k) {
  // Only include odd modes.
  if (k % 2 == 1) Uks[k] = SF * Ukc[0] * 4/(k*M_PI) * Exp(-lambda*(k-1)*(k-1));;
}

//Square variables
double yval, zval, ystart, yend, zstart, zend, ystep, zstep;
int ynum, znum;

//square config
ystart=-0.1;
yend=-0.05;
zstart=0.05;
zend=0.1;

//number of points on grid (not including end point)
ynum=100;
znum=100;

//step sizes
ystep=(yend-ystart)/double(ynum);
zstep=(zend-zstart)/double(znum);

//storage array
int tnum, arysize, arypos;
tnum=int(ceil(t_max/dt));
arysize=3*tnum*(ynum+1)*(znum+1);

// double Lyapexps
double *Lyapexps=new double[arysize];
if(Lyapexps==NULL)
{
        cerr << "Memory Allocation Error" << endl;
}

//Loop for square
for(int yi=0; yi<=ynum; yi++)
{
        //defining start of y
        yval=ystart+(yi*ystep);
```

```
for(int zi=0; zi<=znum; zi++)
{
        //defining start of z
        zval=zstart+(zi*zstep);

        //setting time to zero
        t=0;

        Velmmix<Real> vel(a, alpha, beta, Uks, Ukc);

        // LyapunovFlowDirect<Velmmix<Real>,Real> vellyap(vel,fbtime);
        LyapunovFlowQR<Velmmix<Real>,Real> vellyap(vel,fbtime);

        const int nr = vellyap.base_size();

        mathvector<Real> r(vellyap.size());   // Base space state vector.
        mathvector<Real> lyap(nr);            // Lyapunov exponents.
        mathmatrix<Real> Wt(nr,nr);           // Characteristic eigenvectors.
        mathvector<Real> Lam(nr);             // Coefficients of expansion.

        r[0] = 0;
        r[1] = yval;
        r[2] = zval;

        vellyap.Initialize(r);

        cout.precision(15);
        cout.setf(ios::scientific);

        AdaptiveRKCashKarp<LyapunovFlowQR<Velmmix<Real>,Real>,mathvector<Real> >
              int_vellyap(vellyap, t0, r, .001, 0, acc);

        //Integrate to t max
        while (t < t_max)
        {

              t += dt;
              int_vellyap.IntegrateTo(t,r);

              vellyap.Eigensystem(r,Lam,Wt);

              // Find the finite-time Lyapunov exponents.
              //Real sumlyap = 0;
              for (int i = 0; i < nr; ++i)
              {
                lyap[i] = Log(Lam[i])/(t - t0);
                //sumlyap += lyap[i];
              }


              //putting in positions
              arypos=3*(ynum+1)*(znum+1)*(int(round(t/dt))-1)+3*(ynum+1)*zi+3*yi;
              for (int i = 0; i <nr; ++i)
```

```
                        {
                                Lyapexps[arypos+i]=lyap[i];
                        }

                }


        }
  }
  //end square loop


    for(int ti=1; ti<=tnum; ti++)
    {
     //output file
        ofstream fname;
        string dataprefix = "Lyap_t=";
        ostringstream params;
        params << ti*dt;
        dataprefix = dataprefix + params.str() + ".dat";
        // Open the file.
        fname.open(dataprefix.c_str(),ios::out);
        arypos=3*(ynum+1)*(znum+1)*(ti-1);
        for (int ii =0; ii<=(ynum+1)*(znum+1)-1; ii++)
        {
                fname << setw(8) << Lyapexps[arypos+3*ii] << "\t";
                fname << setw(8) << Lyapexps[arypos+3*ii+1] << "\t";
                fname << setw(8) << Lyapexps[arypos+3*ii+2] << "\t";
                fname << endl;
        }
        fname.close();
        }
        delete Lyapexps;

}
```

Appendix C

Matlab Code

C.1 Mix Score Code

```
function sectionplot(a_st,a_end,b_st,b_end)

%params
h=0.25;
l=1;

%steps
step=1;
zdiv=10;
ydiv=60;

fprintf(1,'alpha\tbeta\t\tRatio\t\t\tFwds Ratio\t\tBack Ratio\n');

for i=a_st:step:a_end
    for j=b_st:step:b_end
        fmix=0;
        fnomix=0;
        bmix=0;
        bnomix=0;
        mix=0;
        nomix=0;

        %start
        d=load(['section_' int2str(i) '_' int2str(j) '_Lx0.000.dat']);

        for z=h/zdiv:h/zdiv:h
            for y=-l/2+l/ydiv:l/ydiv:l/2
                yy=find(d(:,4)<z & d(:,4)>z-h/zdiv & d(:,3)<y & d(:,3)>y-l/ydiv);
                if length(yy)>0
                    mix=mix+1;
                else
                    nomix=nomix+1;
                end
            end
```

```
end

for z=h/zdiv:h/zdiv:h
    for y=-l/2+l/ydiv:l/ydiv:l/2
        yy=find(d(:,5)>0 & d(:,4)<z & d(:,4)>z-h/zdiv & d(:,3)<y & d(:,3)>y-l/ydiv);
        if length(yy)>0
            fmix=fmix+1;
        else
            fnomix=fnomix+1;
        end
    end
end

for z=h/zdiv:h/zdiv:h
    for y=-l/2+l/ydiv:l/ydiv:l/2
        yy=find(d(:,5)<=0 & d(:,4)<z & d(:,4)>z-h/zdiv & d(:,3)<y & d(:,3)>y-l/ydiv);
        if length(yy)>0
            bmix=bmix+1;
        else
            bnomix=bnomix+1;
        end
    end
end

%Half way
d=load(['section_' int2str(i) '_' int2str(j) '_Lx0.500.dat']);

for z=h/zdiv:h/zdiv:h
    for y=-l/2+l/ydiv:l/ydiv:l/2
        yy=find(d(:,4)<z & d(:,4)>z-h/zdiv & d(:,3)<y & d(:,3)>y-l/ydiv);
        if length(yy)>0
            mix=mix+1;
        else
            nomix=nomix+1;
        end
    end
end

for z=h/zdiv:h/zdiv:h
    for y=-l/2+l/ydiv:l/ydiv:l/2
        yy=find(d(:,5)>0 & d(:,4)<z & d(:,4)>z-h/zdiv & d(:,3)<y & d(:,3)>y-l/ydiv);
        if length(yy)>0
            fmix=fmix+1;
        else
            fnomix=fnomix+1;
        end
    end
end

for z=h/zdiv:h/zdiv:h
    for y=-l/2+l/ydiv:l/ydiv:l/2
        yy=find(d(:,5)<=0 & d(:,4)<z & d(:,4)>z-h/zdiv & d(:,3)<y & d(:,3)>y-l/ydiv);
        if length(yy)>0
```

```
                    bmix=bmix+1;
                else
                    bnomix=bnomix+1;
                end
            end
        end

        % Print to screen
        ratio=mix/(mix+nomix);
        fratio=fmix/(fmix+fnomix);
        bratio=bmix/(bmix+bnomix);
        fprintf(1,'%g\t\t%g\t\t\t%7.5g\t\t\t%7.5g\t\t\t%7.5g\n',i,j,ratio,fratio,bratio);
    end
end
```

C.2 Normalised PDF Code

```
%Normalised PDF code

%Number of Fourier modes
fn=1;

%U0 velocity
U0=5;

%Setting font size
fsize=16;
fcsize=14;
fonttype = 'Times';

%No of histogram bins
nb = 51;

%time=i
i=201.6;
d=load(['Lyap_a3b5_fn=' int2str(fn) '_U0=' int2str(U0) '_t=' num2str(i) '.dat']);
d=(d(:,1)-mean(d(:,1)))*sqrt(i);

%define bins
dmax=max(d);
dmin=min(d);
step=(dmax-dmin)/(nb-1);
bins = [dmin:step:dmax];

[P,ly]=hist(d(:,1),bins);
% Normalise
P = P/trapz(ly,P);
plot(ly,P,'y','linewidth',2);
hold on

Pmax=max(P)+max(P)/10;
```

```
i=302.4;
d=load(['Lyap_a3b5_fn=' int2str(fn) '_U0=' int2str(U0) '_t=' num2str(i) '.dat']);
d=(d(:,1)-mean(d(:,1)))*sqrt(i);
[P,ly]=hist(d(:,1),bins);
% Normalise
P = P/trapz(ly,P);
plot(ly,P,'r','linewidth',2);

i=403.2;
d=load(['Lyap_a3b5_fn=' int2str(fn) '_U0=' int2str(U0) '_t=' num2str(i) '.dat']);
d=(d(:,1)-mean(d(:,1)))*sqrt(i);
[P,ly]=hist(d(:,1),bins);
% Normalise
P = P/trapz(ly,P);
plot(ly,P,'g','linewidth',2);

i=504;
d=load(['Lyap_a3b5_fn=' int2str(fn) '_U0=' int2str(U0) '_t=' num2str(i) '.dat']);
d=(d(:,1)-mean(d(:,1)))*sqrt(i);
[P,ly]=hist(d(:,1),bins);
% Normalise
P = P/trapz(ly,P);
plot(ly,P,'b','linewidth',2);

%Guassian
mu=mean(d(:,1));
sd=std(d(:,1));
x=bins;
gauss=0;
for(i=1:1:nb)
    gauss(i)=(1/sqrt(2 * pi * sd^2))*exp(-0.5*((x(i)-mu)/sd)*((x(i)-mu)/sd));
end

plot(x,gauss,'k--','linewidth',2);
hold off

%Labelling the Graph
title(['Shifted Lyapunov Exponents with U0=' int2str(U0) ' and ' int2str(fn)
        Fourier Modes'],'FontName',fonttype,'FontSize',fsize)
ylabel('Normalised PDF','FontName',fonttype,'FontSize',fsize)
xlabel('(Lyapunov Exponent - Mean) * sqrt(Time)','FontName',fonttype,'FontSize',fsize)
text(.01,1700,['Time=' num2str(i)],'FontName',fonttype,'FontSize',fsize);
xmax=min(abs(dmin+step),abs(dmax-step));
axis([-xmax,xmax,0,Pmax])
LEGEND('t=201.6','t=302.4','t=403.2','t=504','Gaussian')
set(gca,'FontName',fonttype,'FontSize',fcsize,'FontWeight','normal');
```