Washington Experimental Math Lab
University of Washington
Fall 2017
Randomly Mixing Fluids

Keith Fife
Yuqi Huang

Abstract:

In this research, two different models for fluid mixing are explored, implemented and statistical measurements are taken. The first is the Gather and Spread model proposed by Diaconis and Pal in 2017, the second is the Toroidal Lattice Mixing model proposed by Pierrehumbert in 2000. We calculate Kendall's τ, and show evidence for a linear relationship between the number of points in the Gather and Spread model, and the number of trials. We show evidence against Gather and Spread distributing its points in Poisson manner. We calculate the standard deviation and relative entropy in the Toroidal Lattice Mixing model. We find strong evidence for the TLM having an exponential mixing rate, and strong evidence suggesting that advection diffusion mixing methods are always faster than diffusion alone.

1 Contents

2 Mixing

The physical mixing of fluids is the the combination of advection usually caused by a stirring apparatus, and diffusion caused by the random molecular bombardment from one fluid onto another.

2.1 Advection

Advection is the movement of a fluid by bulk motion which preserves local properties of the fluid, e.g. energy and entropy. In physical systems, advection satisfies the advection equation:

$$\frac{\partial \psi}{\partial t} + \nabla \cdot (\psi \vec{v}) = 0$$

Where $\psi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is a differentiable scalar field representing the fluids concentration at a point in $\mathbb{R}^n$, $\vec{v} : \mathbb{R}^n \to \mathbb{R}^n$ is a differentiable velocity vector field, and t is the time variable.

> For a simple example of advection, let $f : \mathbb{R} \to \mathbb{R}$ differentiable. Then let $g(x,t) = f(x-t)$, $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, then $\frac{\partial g}{\partial t} = -f'(x-t)$, $\frac{\partial g}{\partial x} = f'(x-t)$, so g satisfies the advection equation.
> So we may just think of advection as a function sliding constantly in time.

2.2 Diffusion

Advection is the movement of fluid from a higher potential to lower potential by way of random molecular bombardment. A molecule in a fluid undergoing diffusion moves in a random walk, i.e. a Brownian motion. In physical systems diffusion satisfies the diffusion equation:

$$\frac{\partial \psi}{\partial t} = D \Delta \psi$$

Where ψ is the same $\psi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ differentiable as advection, D is the fluid's constant of diffusion, and t is the time variable.

> For a simple example of diffusion, simply note that the diffusion equation and the heat equation are the same. So a useful way think of diffusion is to think of heat moving through a fluid or through and object.

2.3 Random Mixing

Mixing has traditionally been done on a large scale in a deterministic manner. Moving the same stirring rod, in the same pattern, to produce roughly the same results. While the results from deterministic mixing are fairly predictable, in many cases they leave certain areas better mixed than others. The idea then is to use random advection processes to, in most cases, try to make a faster or better mixing process. Better in the sense, that the fluid being mixed is more evenly distributed throughout the other fluid.

3 Gather and Spread

The discrete model of mixing fluids contains a boundary. Say we have a table with fixed length and width x and y. Imagine there are m ranked particles that are on the table. In each experiment, we will randomly choose a position as the center of a circle with a fixed radius r. All particles that lie in this circle will be gathered to the center if the center is on the table. The position will be chosen from the range $(x + r) * (y + r)$, which means there are cases that the center is off the table. In those cases, we will test if the distance between the center and the closest location on the table is less than or equal to r. If it is, we will regard that closest location as the destination of gathered particles, otherwise we will skip this experiment. Once particles are gather to the center, we will randomly choose a direction from 0 to 2π. For each particle, we toss an independent coin with probability p of turning up heads. If the coin turns up heads, we will move that particle in the direction we choose for a fixed distance, otherwise it will not move. We will separately calculate the distance it will move on x-axis and y-axis (if the dimension is higher, we will calculate the distance to move on all axis separately) to make sure the particle will move along other axis even if it reaches the boundary on some axis.

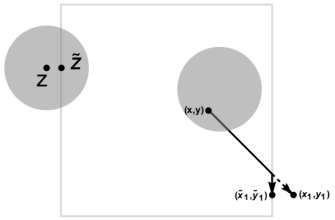


FIGURE 1. The effect of the boundary on the gather and spread operations

In this model, gather is a way of local mixing since we collect nearby particles in a small range. Advection happens when particles move in the direction we pick, and diffusion also happens in this process since particles are moving from high density to low density.

3.1 G&S Statistics

In order to test the relation between the number of particles and the number of experiments we need to do to achieve enough randomness, we will introduce Kendall's Tau. Suppose we have m ranked particles on the table. After randomly mixing, we will calculate the number of pairs (i, j) where point with rank i was to the left of point with rank j before mixing and was to the right of point with rank j after mixing. In the case that there is the tie, we will get a random permutation to break those ties.

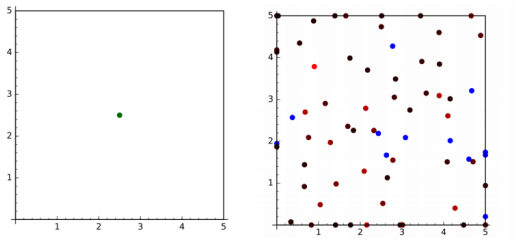
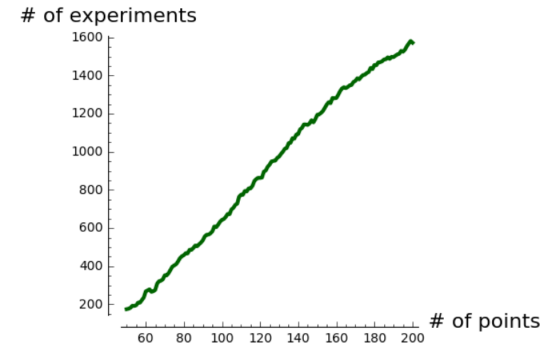


FIGURE 2. Configuration of 250 cards after 3000 steps of gather-and-spread. Table is $[0, 5]^2$, $\delta = 0.5$, $s_0 = 1$, $p = 0.5$. Initially all cards are at the center shown on the left. At the terminal step, shown on the right, there are 68 clusters of cards which are colored by the number of points in the cluster relative to the maximum. Singleton clusters are colored blue while the largest clusters is in bright red. There are 23 clusters on the boundary. (Figure produced by Yuqi Huang).

We tested the cases where the number of particles is from 50 to 200, and we averaged the result of experiments in 100 trials in order to reduce the effect of outliers. We can clearly see a linear trend from the plot. Also, we tested the distribution of particles based on the number of points at each cluster, and we got a strong evidence against Poisson distribution.



4 Toroidal Lattice Mixing

The Toroidal Lattice Mixing model (TLM) was originally proposed by Professor Raymond Pierrehumbert in 2000 in the journal Chaos. The model originally went, and was proposed as a model designed to be computationally efficient and used for understanding the PDEs at play for advection diffusion.
The TLM is a boundaryless model for simulating advection diffusion mixing on a discrete grid. The advection diffusion can either be programmed to be random or nonrandom, however for the purposes of this paper, it will be random unless otherwise stated. The TLM is bounaryless in the sense that if the fluid progresses to far to one side of the grid, it will be wrapped around to the other side. Hence as is trivial topology, the TLM allows us to mix on the surface of a torus, hence it is toroidal. As the TLM is composed of a grid of points to simulate a continuum, it is a lattice model. As the TLM roughly simulates fluid mixing, it is a mixing model. Hence the name Toroidal Lattice Mixing.

4.1 Details of the TLM

The TLM consists of points on an n by m grid. In this sense the at each step TLM may be thought of a matrix X, whose entries add up to 1 and are nonnegative, hence X may also be thought of as a probability distribution over $\mathbb{N}_n \times \mathbb{N}_m$[1]. So $X \in P$ for $P = \{x_{i,j} | \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} = 1, x_{i,j} \in [0,1]\} \subseteq [0,1]^{nm}$. At each step of the TLM X undergoes an advection and a diffusion process.
The advection process is where random mixing takes place. Advection is a function $A : P \times \Omega \to P$, where Ω is a probability space, which permutes the elements of X in a very specific way. Given functions $f : [0,m] \to \mathbb{R}$ and $g : [0,n] \to \mathbb{R}$ continuous, stirring functions for advection are functions $s_x : \mathbb{N}_m \times \Omega \to \mathbb{Z}$, $s_y : \mathbb{N}_n \times \Omega \to \mathbb{Z}$, given by $s_x(y) = \lfloor f((y-d) \mod m) \rceil$, $s_y(x) = \lfloor g((x-d) \mod n) \rceil$[2]. For a point z in X with coordinates (x,y), advection maps the tuple:

$$A : (x,y,z) \mapsto (x \pm s_x(y), y, z) \text{ or } A : (x,y,z) \mapsto (x, y \pm s_y(x), z)$$

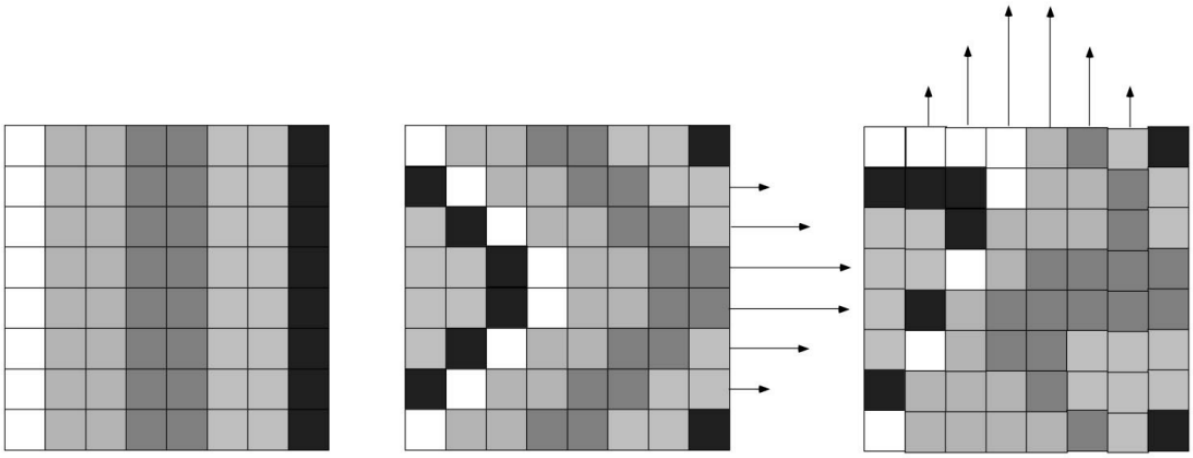
With the direction[3], and orientation[4] and phase shift[5] randomly selected from Ω. The advection step may be considered as a group action by a random element of a subset of S_{nm}[6].

> A useful way to think about advection is to think about a combination lock. An advection step on a combination lock rotates each cylinder by a certain amount varying by some function.

The advection step is visually captured by this image from Pierrehumbert's paper:
The diffusion step in the TLM averages out a point with its neighbors. Diffusion is a function $D : P \to P$ where for a point z in the lattice X, with coordinates (x,y), diffusion maps:

$$D : (x,y,z) \mapsto \tfrac{1}{4}((x-1,y,z_1) + (x+1,y,z_2) + (x,y-1,z_3) + (z,y+1,z_4))$$

[1] $\mathbb{N}_n = \{i | , i \leq n, i \in \mathbb{N}\}$.
[2] $\lfloor x \rceil =$ nearest integer to x.
[3] Advecting in the x or y direction.
[4] Advecting in the positive or negative orientation.
[5] Shifting the stirring function along to center the stir at one point or another, given by the d in the stirring functions.
[6] S_n is the symmetric group over n tuples.



4.2 Properties of the TLM

Theorem 1 *Diffusion can be represented as a the sum of matrix multiplications on X.*

As X is:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix}$$

From the definition of our diffusion function from above we find $D(X)$ to be:

$$D(X) = \tfrac{1}{4}\Big(\begin{bmatrix} x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ x_{3,1} & x_{3,2} & \cdots & x_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \\ x_{1,1} & x_{1,2} & \cdots & x_{1,n} \end{bmatrix} + \begin{bmatrix} x_{m,1} & x_{m,2} & \cdots & x_{m,n} \\ x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m-1,1} & x_{m-1,2} & \cdots & x_{m-1,n} \end{bmatrix} + \begin{bmatrix} x_{1,2} & x_{1,3} & \cdots & x_{1,n} & x_{1,1} \\ x_{2,2} & x_{2,3} & \cdots & x_{2,n} & x_{2,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m,2} & x_{m,3} & \cdots & x_{m,n} & x_{m,1} \end{bmatrix} + \begin{bmatrix} x_{1,n} & x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,n} & x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m,n} & x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix} \Big)$$

As matrix row and column swapping can be represented a a matrix multiplication from elements of the permutation group on $n \times m$ matrices, diffusion may be considered:

$$D(X) = \tfrac{1}{4}(P_1 X + P_2 X + X P_1' + X P_2')$$

Where P_1, and P_2 permute the rows, and P_1' and P_2' permute the columns in the way specified above. This means that P_1 is given by the permutation $\sigma_1 = (1, 2, ..., n-1, n)$, and P_2 is given by $\sigma_2 = (n, n-1, ..., 2, 1)$, and P_1' and P_2' are given by $\sigma_1' = (1, 2, ..., m-1, m)$ and $\sigma_2' = (m, m-1, ..., 2, 1)$ respectfully. Hence as $\sigma_1 \sigma_2 = \sigma_1' \sigma_2' = \mathrm{id}$[7], it follows that $P_1 P_2 = I_m$, and $P_1' P_2' = I_n$. Hence we find:

$$D(X) = \tfrac{1}{4}(P_1 X + P_2 X + X P_1' + X P_2') = D_1 X + X D_2$$

So diffusion may be considered to be a sum of matrix multiplications. □

Unfortunately, we are not so lucky with advection.

Theorem 2 *Advection is not representable as a matrix multiplication on X.*

Note the possible matrix X, with possible advection $A(X)$:

[7] Where id is the identity permutation.

$$X = \begin{bmatrix} .5 & .5 \\ 0 & 0 \end{bmatrix},\ A(X) = \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix}$$

Then it trivially follows that A can not be represented as a matrix multiplication as $\det X = 0$, but $\det A(X) \neq 0$, hence if A could be represented as a matrix multiplication, then $A(X) = AX$, so $\det AX = \det A \det X = 0$. $\square$

5 Implementation of the TLM

I implemented the TLM in Mathematica using:
A grid size of $n = m = 100$.
The stirring functions $s_x(y) = \lfloor 10 \sin^2(y\frac{\pi}{m} - d) \rceil$, and $s_y(x) = \lfloor 10 \sin^2(x\frac{\pi}{n} - d)^2 \rceil$.[8]
The initial distribution $x_{i,j} = \begin{cases} \frac{.95}{4} & (i = \frac{n}{2} - 1 \text{ or } i = \frac{n}{2}) \text{ and } (j = \frac{n}{2} - 1 \text{ or } j = \frac{n}{2}) \\ \frac{.05}{nm-4} & \text{otherwise} \end{cases}$
I chose n and m to be large enough to have a decent grid size, but small enough that it wouldn't take forever to compute one mixing simulation.[9]
I chose the stirring functions to initially be sin rather than $\sin^2$, based on Pierrehumbert's paper. However, this caused a bug which I will call boundary sheering, and will discuss next. Sinusoidal functions are desirable to choose for stirring functions as they are periodic, this allows us to not have to deal with takig mod n or mod n.
I chose the initial conditions for the lattice to simulate a discrete Dirac spike. However for statistical testing, a well defined logarithm is needed, so I had to include some of the lattice's weight on each point, arbitrarily choosing 95% for the center four points, and 5% elsewhere.

Visualizations of the TLM in action can be found:

An unscaled demonstration:	https://youtu.be/Qs10ytClazM
A scaled demonstration:	https://youtu.be/LYe9XI0gInE

5.1 Boundary Sheering

Boundary sheering is a glitch in the TLM which causes the TLM to not have its desired boundaryless property. Boundary sheering occurs when the functions f and g for which he stirring functions s_y and s_x are based on are not nonnegative.
Boundary sheering is the phenomena given by the following conditions:
A portion of the distribution approaches and crosses boundary[10] from a direction α.
Upon crossing the boundary from α, the system advects in a codirection[11] β.
Given the correct circumstances, as the stirring functions have negative values on some part of their domain, the portion which crossed the boundary is advected in opposite directions, one side to positive β and the other to negative β.
This phenomena has the effect of tearing or shearing the distribution near the boundary. This exposes high density areas to low density areas very quickly, causing a great boost to diffusion and rapidly accelerating the mixing process. However, this is a glitch or bug, and not actually a part of the model.
The boundary sheering glitch is easily fixed by simply taking our stirring functions to be entirely nonnegative. Unfortunately this bug was not caught until very late, so I have not had time to run new simulations with much longer mixes.

Visualizations of mixing with the boundary shear glitch may be found at:

An unscaled demonstration:	https://youtu.be/hEmn3mY4MXU
A scaled demonstration:	https://youtu.be/7LPf6ikzRms

[8] I originally user sin rather than $\sin^2$, however I noticed a bug which I will discuss later which went unnoticed until very recently, but caused a large problem.
[9] My code is unfortunately not very optimized at this point, and could use an overhaul, specifically with regards to diffusion.
[10] What the boundary would be when thought of as a simple grid $\mathbb{N}_n \times \mathbb{N}_m \subseteq [0, n] \times [0, m]$.
[11] Given a direction α, a codirection β is a direction orthogonal to α. For example x positive has the codirections y positive and negative.

5.2 The Code

The Mathematica code I wrote is presented here. I will go line by line explaining each step.

n = 100;
This specifies our n, one dimension of our lattice.

m = 100;
This specifies our n, one dimension of our lattice.

total = n*m;
This defines a shorthand for nm.

z[x_, y_]:=N[If[(x==n/2-1||x==n/2)&&(y==m/2-1||y==m/2),95/400,5/(100*total-400)]]
This is our initial conditions for our lattice.

columnset=Table[{Mod[i-1,n],Mod[-j,m],z[i,j]},{j,m},{i,n}];
This forms our lattice as the set of triples (x, y, z). This is necessary for our advection step, as it will require operating on the coordinate of a point, then sorting.

pointset=Flatten[columnset,1];
This is our actual set of triples. When forming the previous set, Mathematica partitions entire rows of our matrix X into separate lists. However for advection to operate properly, we need to only have the set of triples.

repointset:=pointvectorset[Table[{Mod[i-1,n],Mod[-j,m],z[i,j]},{j,m},{i,n}]]
This is a constant set which is defined to reset our initial conditions for running multiple simulations via a loop.

latticesort[set_]:=Sort[set,#1[[2]]>#2[[2]]||(#1[[2]]==#2[[2]]&&#1[[1]]>#2[[1]])&]
This sorts our lattice based on a lexicographical ordering for the purposes of advection. Since in advection we operate on the incidences, we have to sort by the indecies to place the elements in the proper order after an advection, so the following diffusions and advections will operate properly.

latticematrixsetn[set_]:=Partition[set,n]
This takes our lattice as a set of triples, then places it into a $n \times m$ matrix form. This will be useful late for diffusion, as my initial code did not use matrix multiplication.

latticematrixsetm[set_]:=Partition[set,m]
This take our set of triples and places it into a $m \times n$ matrix form. Again this is useful later in diffusion.

latticematrix[set_]:=MatrixForm[Partition[set,n]]
This put the set of triples into a literal matrix in Mathematica. This is useful in debugging, but not otherwise.

pointvectorset[set_]:=Flatten[set,1]
This takes our set of triples in a matrix form, and forces it back down into its set of triples form.

modn[x_]:=Mod[x,n]
This is a mod by n function defined because of the quirks of the Mathematica MapAt function.

modm[x_]:=Mod[x,m]
This is a mod by m function defined for the same reason. It allows us to take the mod of an entire list or part of a list at once.

modsetn=Table[{i,1},{i,n*m}];
This allows us to apply our previous mod n function to each triple's first coordinate.

modsetm=Table[{i,2},{i,n*m}];

This allows us to apply our previous mod m function to each triple's second coordinate.

sx[p_,h_]:=Round[10Sin[p*Pi/m+h]^2]
This is our s_x stirring function.

sy[p_,h_]:=Round[10Sin[p*Pi/n+h]^2]
This is our s_y stirring function.

stirxplus[vector_,b0_]:=latticesort[MapAt[modn,vector+Flatten[Table[{sx[Mod[-j,m],b0],0,0},{j,m},{i,n}],1],modsetn]]
This is the application of our stirring function s_x in the positive direction. The Table command in the center forms a set of triples $(s_x(y), 0, 0)$ for each $(x, y) \in \mathbb{N}_n \times \mathbb{N}_m$. This is flattened down so it is compatible with out set of triples. Then it is added to our set of triples. To make sure that each element in the lattice has coordinates still in $\mathbb{N}_n \times \mathbb{N}_m$, mod n is taken on the first coordinate. Then the set of triples is sorted so it may be operate on again.

stiryplus[vector_,c0_]:=latticesort[MapAt[modm,vector+Flatten[Table[{0,sy[Mod[-i,n],c0],0},{j,m},{i,n}],1],modsetm]]
This does the same as the above, but for the positive y direction.

stirxminus[vector_,b0_]:=latticesort[MapAt[modn,vector-Flatten[Table[{sx[Mod[-j,m],b0],0,0},{j,m},i,n],1],modsetn]]
This does the same as the above, but for the negative x direction.

stiryminus[vector_,c0_]:=latticesort[MapAt[modm,vector-Flatten[Table[{0,sy[Mod[-i,n],c0],0},j,m,i,n],1],modsetm]]
This does the same as the above, but for the negative y direction.

stir[vector_,displacement1_,displacement2_,direction_]:=Piecewise[{{stirxplus[vector,displacement1],direction==0},
{stirxminus[vector,displacement1],direction==1},{stiryplus[vector,displacement2],direction==2},
{stirxminus[vector,displacement2],direction==3}}]
This specifies a function which applies the stirring functions, and depends on a randomly chosen direction x or y, and orientation positive or negative.

deletesetnm=Table[{i},{i,n*m-n}];
This is a table useful for deleting the first $nm - n$ entries in our set of triples. This will be useful in our row swapping.[12]

deletesetn=Table[{i},{i,n}];
This does the same as the above, but only deletes the first n points.

deletesetm=Table[{i},{i,m}];
This does the same as above, but only deletes the first m points.

deletesetmn=Table[{i},{i,n*m-m}];
This does the same as above, but deletes the first $nm - m$ points.

subdiffusionsortxplus[vector_]:=Join[Delete[vector,deletesetm],vector[[1;;m]]]
This takes the first m elements of our set of triples and puts them at the end of our set of triples.

subdiffusionsortxminus[vector_]:=Join[Delete[vector,deletesetmn],vector[[1;;n*m-m]]]
This takes the first $nm - m$ elements of our set of triples and puts them at the end of our set of triples

diffusionsortxplus[vector_]:=pointvectorset[Transpose[latticematrixsetm[subdiffusionsortxplus[pointvectorset[
Transpose[latticematrixsetn[vector]]]]]]]
This has the effect of moving the first column vector to the end of our matrix of points, the same effect as P_1'. We first put our set into a matrix form. Then take the transpose. Then compress back into a set of triples. We

[12] I didn't know about the power of permutation matrices until after I had already written this code, so our row swapping will come from deleting and joining parts of our set of triples.

take the first m elements and send them to the back of our set of triples. We then reform the set into a matrix. Retake then transpose. Then compress back to a set of triples.

diffusionsortxminus[vector_]:=pointvectorset[Transpose[latticematrixsetm[subdiffusionsortxminus[pointvectorset[Transpose[latticematrixsetn[vector]]]]]]]
This has the effect of moving the last column vector to the front of our matrix, the same effect as P_2'. This operates similar to the above description.

diffusionsortyplus[vector_]:=Join[Delete[vector,deletesetnm],vector[[1;;n*m-n]]]
This has the effect of moving the last column vector to the top of our matrix, the same effect as P_2. This operates similar to the above description.

diffusionsortyminus[vector_]:=Join[Delete[vector,deletesetn],vector[[1;;n]]]
This has the effect of moving the first column vector to the bottom of our matrix, the same effect as P_1. This operates similar to the above description.

diffusion[vector_]:=latticesort[1/4*(diffusionsortxplus[vector]*Table[{0,0,1},{i,n*m}]+diffusionsortxminus[vector] *Table[{0,0,1},{i,n*m}]+diffusionsortyplus[vector]*Table[{0,0,1},{i,n*m}]+diffusionsortyminus[vector]* Table[{0,0,1},{i,n*m}])+vector*Table[{1,1,0},{i,n*m}]]
This has the net effect of our diffusion function. The multiplication by the tables has the effect of deleting the x and y coordinates of the triples, so they won't be messed with in diffusion. They are re-added in the last step by adding the set of triples with their z coordinate[13] and adding back the (x, y) coordinates.

sumdeleteset=Join[Table[{i,1},{i,n*m}],Table[{i,2},{i,n*m}]];
This is a set formed for deleting off the (x, y) coordinates of our triples for statistical evaluation.

euclideandeviation[x_]:=(x-1/total)^2
This takes the Euclidean distance to the fully mixed solution for a given point.

standardeuclideandeviation[vector_]:=Sqrt[First[Total[euclideandeviation[Delete[vector,sumdeleteset]]]]/(total)]
This takes the standard deviation for th entire lattice.

partialentropy[x_]:=x*Log[x]
This takes $x \ln x$ for use in calculating the relative entropy of the system.

totalentropy[vector_]:=Log[total]+First[Total[partialentropy[Delete[vector,sumdeleteset]]]]
This calculates the total entropy for the lattice.

k=300;
This specifies the number of steps desired in an experiment.

For[l=0,l¡k,l++,{Clear[b0,c0],a:=RandomInteger[3],b:=RandomReal[2Pi],c:=RandomReal[2Pi],a0=a,b0=b,c0=c, pointset=stir[N[pointset],b0,c0,a0],pointset=diffusion[pointset]}]
This is an example of a simple implementation of the loop to do a TLM simulation. At each step we generate random numbers a, b, and c. These are used to determine our direction and orientation for our advection, and our phase shifts for our advection. Then advection and diffusion are applied to our lattice.

5.3 Improvements to be Made

The main improvement I would suggest for this code, would be to generate for a given n and m, the P_1, P_2, P_1' and P_2' so that diffusion could be ran as a simple matrix multiplication rather than a complicated list reorganization method. I intend to implement this in the future, but have not had the time to do so thus far.

[13]The actual part of the triple that we care about.

6 TLM Statistics

Given the model parameters explained prior, I ran 500 simulations each with 250 steps with my debugged code. At each step of each simulation I computed the standard deviation of the lattice and the relative entropy of the lattice.
The results were not as spectacular as that of the bugged code, but that is to be expected.

6.1 Standard Deviation

The standard deviation of the lattice is the usual standard deviation. However we are instead taking the deviation from the long run solution of $\frac{1}{nm}$ of a fully mixed fluid. For the matrix of points X, the standard deviation of X is:

$$\sigma(X) = \sqrt{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(x_{i,j} - \frac{1}{nm})^2}$$

The standard deviation is a scaled average denoting the distance from the fully mixed solution. The better mixed X is, the lower $\sigma(X)$.

6.2 Relative Entropy

The entropy of the lattice is the same as the same as the thermodynamic entropy of the fluid that the lattice represents. The entropy of the matrix X is:

$$s(X) = -\sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j} \ln x_{i,j}$$

The maximum entropy of the lattice is the entropy of a fully mixed fluid. This occurs when $x_{i,j} = \frac{1}{nm}$, so the maximum entropy is:

$$s_{max} = -\sum_{i=1}^{n}\sum_{j=1}^{m} \frac{1}{nm} \ln \frac{1}{nm} = \ln(nm)$$

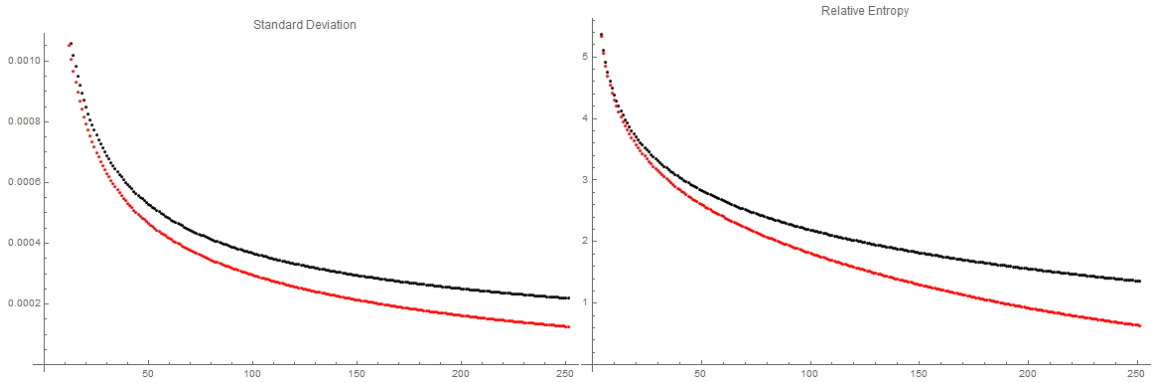
The relative entropy is then taken to be the difference between the maximum entropy and the relative entropy. The relative entropy for X is then:

$$\Delta s(X) = \ln(nm) + \sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j} \ln x_{i,j}$$

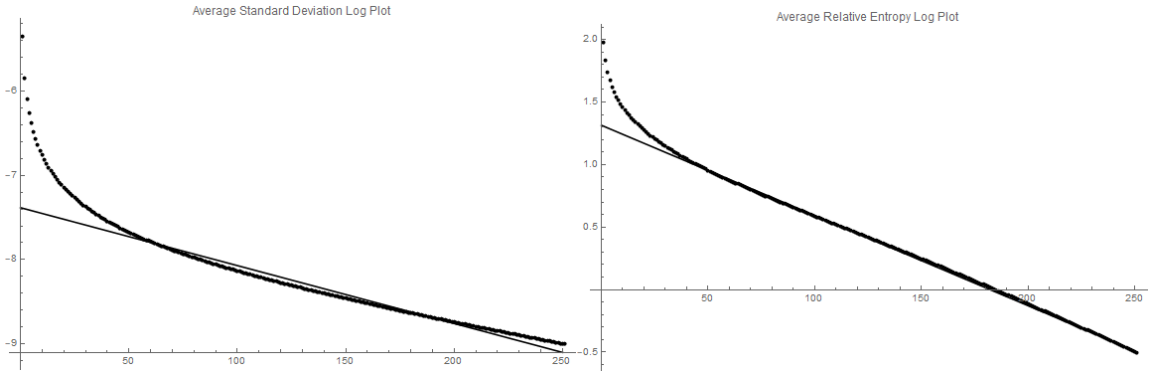
As each $x_{i,j} \in (0, 1)$ the entropy and relative entropy is well defined and positive. The entropy of the lattice is a better measure of smaller differences in points as the logarithm has a pole at 0.

6.3 Statistical Results

The following are the averaged results from the 500 simulations, and the results from a diffusion only simulation: Plotting the standard deviation and relative entropy at each step of the mixing process we generate the plots, where black is the diffusion only system, and red is the diffusion advection system:

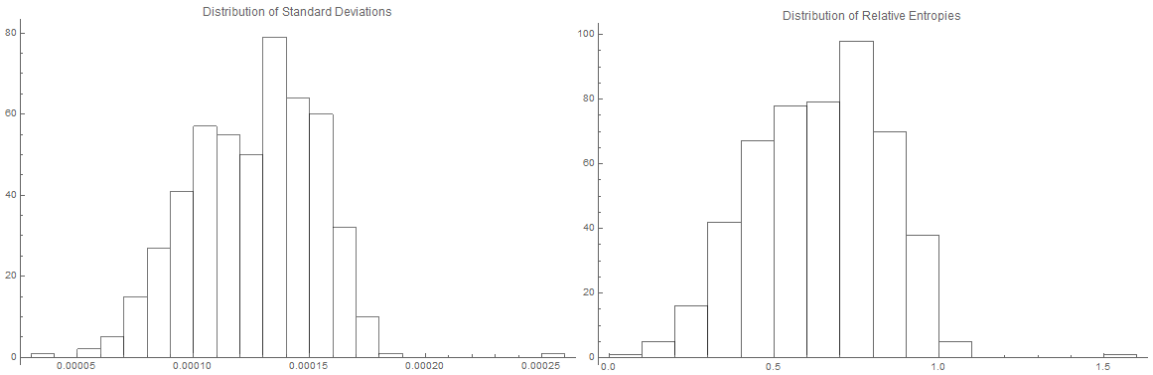


Given that the plots look roughly exponential, the log plots are taken, resulting in the following plots[14], with r^2 values of 97.6% and 99.9% respectfully, and regression slopes $-6.88 \cdot 10^{-3}$ and $-7.22 \cdot 10^{-3}$ respectfully[15]:



While the relative entropy looks roughly linear, the standard deviation does not look linear. That being said, they both have strong r^2 values, so perhaps they are linear, or perhaps after more steps in the simulations the standard deviation will break from its linearity. Regardless, for the data that was calculated, the standard deviation and relative entropy were found to decay roughly exponentially.
That being said, as the standard deviation appears to be concave up, and visually nonlinear, leads to the conjecture that the standard deviation decays sub-exponentially. This conjecture leads immediately to the next, that if the standard deviation decays sub-exponentially, then the relative entropy should decay sub-exponentially.

Comparing the initial and final standard deviations and relative entropies shows that the standard deviation drops by a factor of[16] 37.6, and the relative entropy drops by a factor of 36.3, and the final distributions look roughly normal:



Videos showcasing the evolution of the distributions of the standard deviation and relative entropy per step may be found:

Standard Deviation:	https://youtu.be/RVKMjuP6Ihg
Relative Entropy:	https://youtu.be/D1Q61LS5Zdo

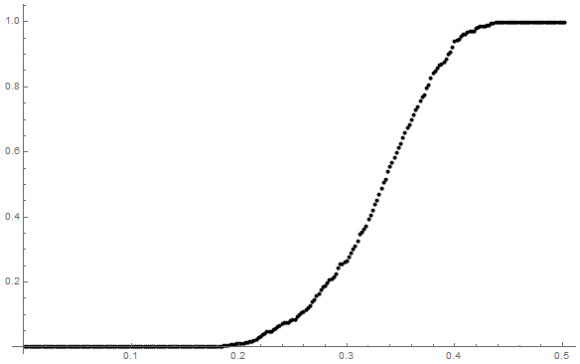
When considering what proportion of the standard deviations after t steps are below some $\varepsilon > 0$, taking $\varepsilon = \frac{1}{nm} = \frac{1}{\text{LatticeArea}(X)}$ we find a pitiful 18.2% after our full 250 steps. However doubling this radius, we find that this proportion rises to 99.8%. So depending on how well one considers a fluid to be well mixed, the model

[14]The first 10% of the data was not taken into account for these regressions as it is nonlinear, and we would not expect linearity from it.

[15]For reference the regression slopes for the original data with the boundary shear bug were $-1.19 \cdot 10^{-2}$ and $-1.68 \cdot 10^{-2}$, that is roughly 1.73 and 2.33 times larger. So if some company can make a machine which mixes on a torus randomly, then occasionally shears large areas, they would be able to mix extremely quickly.

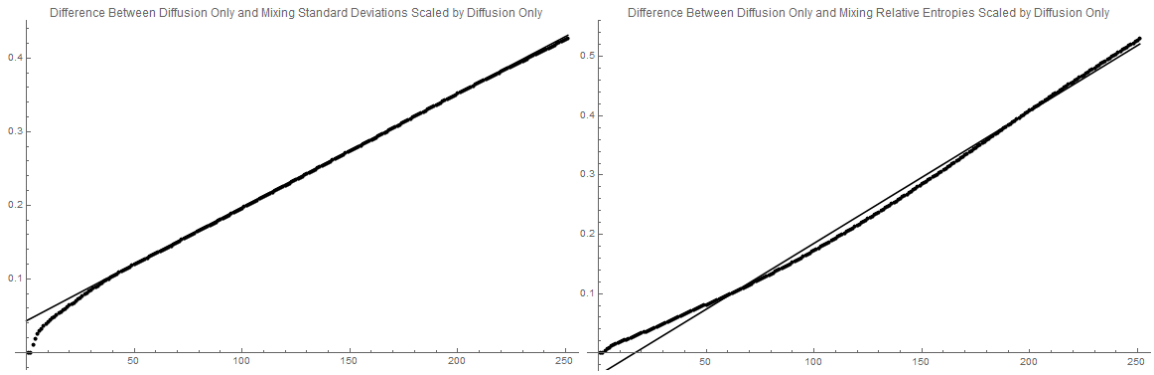
[16]By drops by a factor of, I mean $\frac{\text{Initial}}{\text{Final}}$ is some value.

is either very effective, or not so much[17]. Generating a plot for the proportion of the simulations who's tth step is below 2ε we find:



Note that the curve looks roughly logistic, indicating that as soon as some of the simulations get below ε for some $\varepsilon > 0$, the rest quickly follow suit. This leads to the conjecture that given a few[18] extra steps, the simulations would likely be largely[19] below ε for $\varepsilon = \frac{1}{nm}$.

From the initial plots we can see that the advection diffusion system is faster and mixing the lattice than diffusion alone, but two questions are or at least should be raised: How much faster is advection diffusion, and is advection diffusion faster than diffusion alone?
Taking the plot of the difference between diffusion and advection diffusion scaled by diffusion[20] gives:



Given that they appeared nearly linear after the initial 10% of the data, linear regressions were taken. Their resulting r^2 values were 99.99%, and 99.55%, with regression slopes $1.55 \cdot 10^{-3}$, and $2.23 \cdot 10^{-3}$, indicating that the scaled difference grows roughly linearly.
Answering the first question, we find that advection diffusion after 250 steps gives roughly a 40% boost to mixing speed when looking at the standard deviation, and roughly a 50% boost in mixing speed when looking at the relative entropy. The monotonically increasing nature of the scaled difference leads to the conjecture that, despite the fact that both advection diffusion and diffusion alone lead to the same steady state solution, advection diffusion should always be faster than diffusion alone.

6.4 Data Access

To access the data gathered from the TLM model and the Mathematica notebook containing the model and implementation simply follow this link:

[17] Again for reference for ε as before, after 250 steps was 98.2%. Again if a company could make this magical machine, they would develop an extremely effective mixing technique.
[18] Few meaning somewhere less than another full 250 steps.
[19] As in greater than 75%.
[20] $\frac{\text{Diffusion} - \text{Advection Diffusion}}{\text{Diffusion}}$.

https://drive.google.com/file/d/1DPCqy7Nea50iHQgV64BeyIRl0TItf7wv/view?usp=sharing

7 Conjectures, Questions, and Ways Forward

7.1 Conjectures

- There is exactly one fixed point solution to the equation $X_{t+1} = D(X_t)$.
- The solution is a stable solution.
- There is only one stable solution to the TLM advection diffusion system.
- The Lyapunov exponent for the TLM diffusion only system is negative.
- The Lyapunov exponent for the TLM advection diffusion system is less than the one for the diffusion only system.
- The diffusion advection system always mixes faster than the diffusion only system.
- The standard deviation and relative entropy of the lattice in the TLM decay sub-exponentially.
- The difference between diffusion and diffusion advection scaled by diffusion grows linearly.
- The images by diffusion $D^t(P)$, are nested.
- The set $D^\infty(P) = \lim_{t\to\infty} D^t(P)$ is compact and connected.
- The diffusion function is a contraction map in a metric compatible with the standard topology on $\mathbb{R}$.
- The set of advections on X with the identity permutation form a subgroup of S_{nm}.

7.2 Questions

- What are the effects of an increased grid size?
- What are the effects of non-square grids?
- What are the effects of different stirring functions?
- Do different initial conditions affect mixing speeds?
- How long does it take for two different initial conditions to be indistinguishable?
- What are the effects of increased run time?
- How would a deterministic fixed pattern advection process fair against the random process?

7.3 Ways Forward

- Change the diffusion code to run more efficiently.
- Consider advection as random group action from a subset of S_{nm}, possibly implement that to speed up the code.
- Look into the conjectures and questions.

8 References

Diaconis, P., & Pal, S. (2017). Shuffling cards by spatial motion, 131. doi:arXiv:1708.08147

Gouillart, E., & Dauchot, O. (2007). Etude de ladvection chaotique dans des melangeurs a tiges, en ecoulements ouverts et fermes (thesis).

Pierrehumbert, R. T. (2000). Lattice models of advection-diffusion. Chaos: An Interdisciplinary Journal of Nonlinear Science, 10(1), 6174. doi:10.1063/1.166476

Advection. (2017, December 15). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Advection. Accessed 16 December 2017

Diffusion. (2017, December 10). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Diffusion. Accessed 16 December 2017

Kendall rank correlation coefficient. (2017, December 7). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient. Accessed 16 December 2017

Permutation matrix. (2017, December 4). Wikipedia. Wikimedia Foundation. https://en.wikipedia.org/wiki/Permutation_matrix. Accessed 16 December 2017