Learning Dynamics of Particle Systems

Mikaela Finlay
Mentor: Harry Walden

Summer 2024

1 Introduction

This project aims to use partial differential equations to describe particle motion, both independently and between particles. While the eventual goal is to use real data, the results discussed in this paper are from simulated data only. In order to write equations to model particle motion, we first need to write equations to describe the positions of particles in space and time. Then, we take derivatives of these equations and use sparse linear regression to write the time derivative as a composition of spatial derivatives. In order to manipulate the data to a point where this comparison is meaningful, we need to reduce the noise built into the data and smooth out variations over both time and space. This process is known as coarse graining, and involves writing each equation as an infinite sum of other functions and changing the weights of sum of these functions to remove the highest frequency variations. Once the data is prepared, linear regression with $L1$-regularization is used to generate sparse solutions. This paper considers coarse graining the spatial and time dimensions of particle density and the spatial dimension of particle angular velocity, discusses different ways to express equations on a disk, and explores different approximations of derivatives and ways to use coefficient spaces instead of real values to derive equations.

Information about exact particle locations is often noisy, and becomes less useful as the number of particles increase. In order to retain only the useful information on particle locations, they must be coarse grained. Then, the coarse grained locations need to be translated to a basis where real values are easily recoverable from coefficients. These transitions are shown in figure 1.

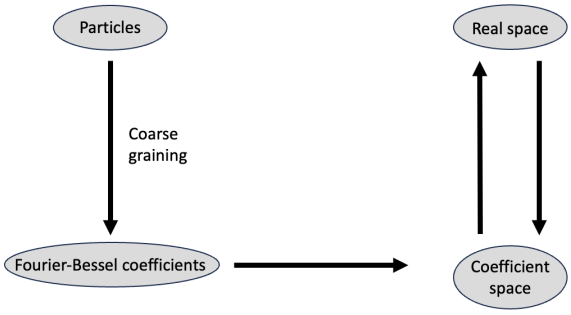

Figure 1: Transition from Particles to Coefficient Space

Any sufficiently nice function on the disk can be expressed as the sum of appropriately chosen functions:

$$\rho(r,\theta,t) = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty}\sum_{l=0}^{\infty} C_{nkl} f_{nkl}(\theta) g_{nkl}(r) h_{nkl}(t) \tag{1}$$

Here, the density over the disk is a function which is the sum of other functions multiplied by coefficients. Because these terms are consistent regardless of the particles the function describes, it is only necessary to store the coefficients as a way to describe the function. In the coarse graining step, we fix f, g, and h to be a

Fourier-Bessel series, as explained in section 2.1. When transitioning to coefficient space, the choice of f, g, and h is more flexible, and we explore different choices for g in section 3. Then, using information about the three sets of functions, we can use a set of coefficients C to determine density at any point and information about density at specific points to generate the coefficients C.

2 Coarse Graining

In order to learn about the density field of particles in general instead of exact particle locations, we use coarse graining to transition from particle locations to a Fourier-Bessel series as shown in figure 1.

2.1 Coarse Graining in Space

In order to create smooth data, it needs to be modified in both spatial and time directions. First, we addressed the spatial dimension. Our goal was to describe density as a probability field instead of exact locations. The density profile should represent how many particles are *likely* to be in a specific region, even though in the original density profile, the probability that a particle is in a region is 1 if the region is the particle's location and 0 otherwise. To turn particles from points to regions, it is common to use a kernel. This kernel could be a Gaussian distribution centered at the particle's location. The area under the Gaussian is 1, so the sum over the probability distribution is equal to the number of particles. In this project, we are interested in what happens to particles at the boundary of the disk, and using a Gaussian at the boundary would mean that some of the distribution falls outside the disk, which does not preserve the mass *on* the disk. Instead, we need a kernel that ensures the total mass on the disk stays the same. To implement this, we can imagine the particles as points of heat, and let the "heat" diffuse over time. We use the heat equation with an initial condition representing our point particles and a zero-flux boundary condition, so the total "heat" on the disk is constant.

To represent point particles, we use the delta function, $\delta(x)$. Given a set of P particles with particle p at position (r_p, θ_p), the initial density profile of the disk (without coarse graining or diffusing) is

$$\rho(r,\theta) = \sum_p \frac{1}{r}\delta(r-r_p)\delta(\theta-\theta_p). \tag{2}$$

The additional $\frac{1}{r}$ is due to the Jacobian of the transformation from Cartesian to polar coordinates, as integrals in polar coordinates are evaluated against r in addition to dr and $d\theta$. Next, we consider the heat equation on the disk. Our equation must satisfy two conditions:

1. The initial condition must be $\rho(r,\theta)$.
2. No heat can flow through the boundary of the disk. To satisfy this, the *derivative* of the equation must be 0 at the boundary. This is a Neumann boundary condition, as the value of the derivative at the boundary is fixed.

Because we are using the heat equation, our equation must satisfy the condition,

$$\frac{1}{\alpha^2}\frac{\partial u}{\partial t} = \nabla^2 u. \tag{3}$$

Written in polar coordinates, the equation must satisfy the condition,

$$\frac{1}{\alpha^2}\frac{\partial u}{\partial t} = \frac{\partial^2 u(r,\theta,t)}{\partial r^2} + \frac{1}{r}\frac{\partial u(r,\theta,t)}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u(r,\theta,t)}{\partial \theta^2}. \tag{4}$$

To ensure condition 1,

$$u(r,\theta,0) = \rho(r,\theta), \tag{5}$$

and to ensure condition 2, for a disk with radius a,

$$\frac{\partial u(a,\theta,t)}{\partial r} = 0. \tag{6}$$

Solutions to this equation come in the form [1],

$$u(r,\theta,t) = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty} C_{nk} e^{-\frac{m_k^n \alpha}{a}^2 t} J_n(\frac{m_k^n}{a} r) e^{-in\theta}, \tag{7}$$

where J_n is the nth Bessel function of the first kind and m_k^n is the kth extrema of the nth Bessel function. The 0th extremum of $J_0(r)$ occurs at $r = 0$, while the 0th extremum of all other Bessel functions is its first local maximum. Scaling Bessel functions to a maximum or minimum guarantees that $\frac{\partial u}{\partial r}$ will be 0 when $r = a$.

Bessel functions are, in general, used to describe the radial components of vibrations on a disk [2]. They are most often used when scaled to zeros instead of extrema, which established a boundary condition of no heat *at* the boundary, a Dirichlet boundary condition, instead of no heat *passing through* the boundary. There are two kinds of Bessel functions: Bessel functions of the first kind, $J_n(r)$, are either 0 or 1 when $r = 0$, while Bessel functions of the second kind, denoted $Y_n(r)$, diverge at the origin. Because we know that the total heat on the disk is equal to the number of particles, the equation to describe it cannot diverge in the center of the disk, so Bessel functions of the second kind are not a valid solution.

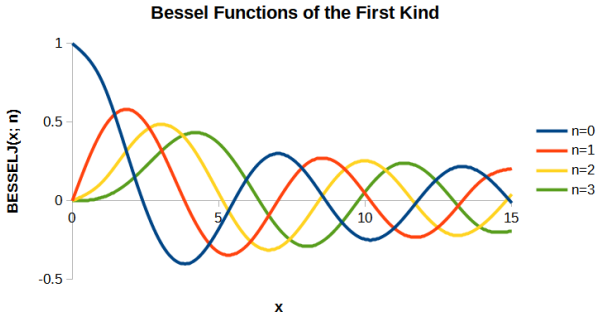

Figure 2: Bessel Functions [3]

The angular term represents a Fourier series, as the angular values are 2π-periodic on the disk. This also allows us to use algorithms like the Fast Fourier Transform to make calculations accurate and efficient.

The only value not provided by the setup of our problem is α. This value can be freely chosen, and determines how much the "heat" diffuses, or how large the kernel is. It is also necessary to derive the coefficients C_{nk}, which are used to enforce condition 1. The coefficients are defined as

$$C_{nk} = \frac{\langle \rho(r,\theta), J_n(\frac{m_k^n}{a} r) e^{-in\theta} \rangle_2}{\langle J_n(\frac{m_k^n}{a} r) e^{-in\theta}, J_n(\frac{m_k^n}{a} r) e^{-in\theta} \rangle_2}. \tag{8}$$

Here, $\langle f(r,\theta), g(r,\theta) \rangle_2$ is the inner product, defined as $\int_0^a \int_0^{2\pi} f(r,\theta) g(r,\theta) r d\theta dr$. The numerator and denominator of each coefficient can be considered independently, with $C_{nk} = \frac{N_{nk}}{D_{nk}}$. The numerator, expanded, is

$$N_{nk} = \sum_p \int_0^a \int_0^{2\pi} \delta(r - r_p)\delta(\theta - \theta_p) J_n(\frac{m_k^n}{a} r) e^{-in\theta} d\theta dr. \tag{9}$$

This simplifies to

$$N_{nk} = \sum_p J_n(\frac{m_k^n}{a} r_p) e^{in\theta_p}. \tag{10}$$

Note that when $n = 0$, the angular term becomes 1, and when both n and k are 0, both the angular and radial components evaluate to 1 for all particles, yielding a constant term.

The denominator expands to

$$D_{nk} = \int_0^a \int_0^{2\pi} (J_n(\frac{m_k^n}{a} r))^2 (e^{-in\theta})^2 r d\theta dr. \tag{11}$$

The angular term resolves to $\epsilon_n \pi$, where $\epsilon_n = 2$ when $n = 0$ and 1 otherwise. The radial term, while slightly more complicated, is simplified using identities found in the Digital Library of Mathematical Functions [4]. Overall, the denominator simplifies to

$$D_{nk} = \epsilon_n \frac{\pi a^2}{2}(J_n(m_k^n)^2 - J_{n-1}(m_k^n)^2). \tag{12}$$

Combining these, the simplified coefficient is

$$C_{nk} = \sum_p \frac{(3-\epsilon_n) J_n(\frac{m_k^n}{a} r_p) e^{in\theta_p}}{\pi a^2 (J_n(m_k^n)^2 - J_{n-1}(m_k^n)^2)}. \tag{13}$$

Here, the $3 - \epsilon_n$ reverses the condition, assuming the value 1 when $n = 0$ and 2 otherwise.

2.2 Spectral Entropy

Now that the function is fully defined, it is important to remember the purpose of using the heat equation. By incrementing the time that the kernel is diffused over, the coefficients are decreased proportional to their mode so that higher frequency terms (those with higher values of n) are penalized more and the function u becomes smoother. The degree to which the coefficients are smoothed depends on both the chosen parameter α and the time t. Because both of these are fixed, they can be considered as one coefficient $\beta = \alpha^2 t$. The original density profile $\rho(r, \theta)$ can be replaced by a new density profile,

$$\rho'(r, \theta) = \sum_{n=0}^{\infty} \sum_{k=0}^{\infty} C'_{nk} J_n \frac{m_k^n}{a} r e^{-in\theta}, \tag{14}$$

where $C'_{nk} = C_{nk} e^{-\frac{m_k^n}{a}^2 \beta}$. The choice of β is what determines how smooth ρ' is. To decide what parameter to choose, we look at the spectral entropy of the coefficients. Spectral entropy is the amount of information stored in the coefficients [5]. The less information coefficients store, the smoother the disk is. To show the spectral entropy of different β values, we looked at data involving twenty particles [6], and reported kernel widths in terms of real measurements on the disk their data is recorded on. Here, our disk radius is 8.75 cm and our particle radius is 0.6 cm. A kernel that is 8.75 cm covers the entire disk and no information is retained, while a kernel that is 0.6 cm matches the original function exactly, and no smoothing occurs, as shown in figure 3.

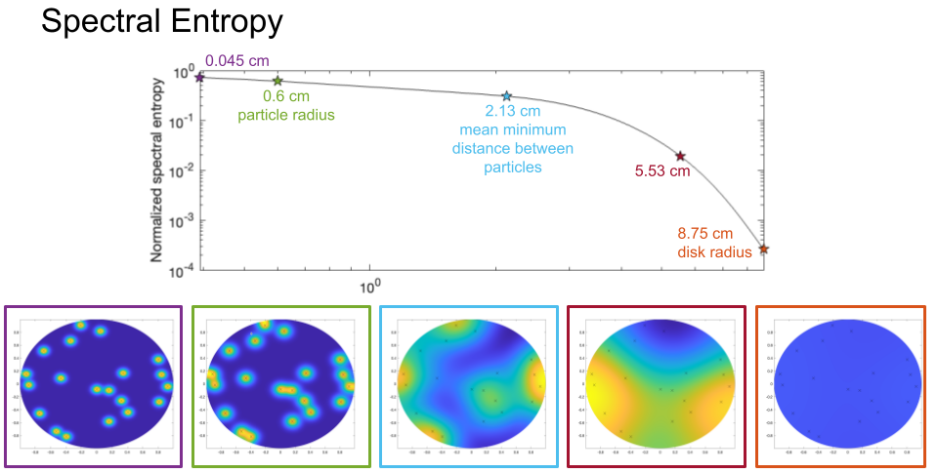

Figure 3: Spectral entropy and coarse grained density profiles of various kernel sizes

Using the mean minimum distance between particles (the kernel size shown in light blue) is a useful metric when determining a kernel size because using a kernel that is larger than the smallest distance between particles means that individual particles are no longer identifiable, which is the goal of coarse graining.

2.3 Coarse Graining in Time

In addition to smoothing the function spatially, we must also smooth it in time. We need to take the time derivative of our data, but because the data is collected at discrete intervals and is noisy, the approximation of the time derivative is almost entirely noise. In order to create a smooth time derivative, we use a Fourier series. The noise comes from high frequency oscillations, so we want to decrease the affects of these high frequency Fourier terms. Instead of simply discarding modes after a certain threshold, we set a smoothing parameter γ and scale the nth term by $e^{-\gamma n}$, so that the first term, with $n = 0$, has no scaling at all and the higher frequency terms are damped more extremely. This method works well because it easily reduces the least smooth parts of the function. However, a Fourier series assumes that a function is periodic. Our data is not, so the Fourier series that we have generated corresponds to a function where the time window is repeated infinitely. This means that there is a jump at the end of the time frame where it must immediately assume the initial value again. Using all of the modes, this is achieved exactly. When the higher frequency modes are damped, the jump at the end of the time frame cannot be exact because the terms that account for very short-range changes in the function have been suppressed. Thus, the transition from the final value back to the initial value begins earlier, and some information about the last time frames is lost. Just as β had to be chosen to balance smoothness and information retained, γ must also generate a meaningful time derivative without losing too much information. While we did not explore this option during this project, a Chebyshev expansion, discussed in section 6.1, may be a preferable basis for the time direction and would help mitigate the obstacles presented by assuming periodicity and using a Fourier series.

2.4 Coarse Graining Angular Velocity

In addition to information on the location of particles, the data obtained by Petroff, et al. [6] contains information on each particle's angular velocity. This is more difficult to represent because while a value of zero for density means that there are no particles present, a value of zero for angular velocity means that the particles are not spinning, and there must be a different choice for locations with no particles. If we consider the density profile to be a probability field representing how many particles are likely to be present in a given region, the angular velocity coarse graining can be thought of as the probability of an angular velocity value conditioned on the density at that point. This means that a disk with one particle will have uniform angular velocity equal to the angular velocity of the single particle. In order to achieve this, we use density coarse graining weighted by the angular velocity of each particle. The density at a location $\mathbf{x}$ is the sum of the value of each particle's kernel K at $\mathbf{x}$ as defined in section 2.1,

$$\rho(\mathbf{x}) = \sum_{j \in particles} K_j(\mathbf{x_j} - \mathbf{x}). \tag{15}$$

Then, the coarse grained angular velocity at $\mathbf{x}$ is

$$\omega(\mathbf{x}) = \frac{\sum_j \omega_j K_j(\mathbf{x_j} - \mathbf{x})}{\sum_j K_j(\mathbf{x_j} - \mathbf{x})}. \tag{16}$$

Figure 4 shows the results when this is implemented on one time frame of the data in [6] with 20 particles. While this paper does not cover learning the dynamics of particles with non-zero angular velocity, this technique will be helpful in future projects that aim to learn the dynamics of this data.

3 Bases

After coarse graining the particles into a Fourier-Bessel series, we transition to a new basis to describe the density. In section 2, we chose functions for f, g, and h in equation 1 with f and h as complex exponentials and g as Bessel functions. We maintain this choice for f and h as explained in section 3.1, and explore different functions for g in the rest of this section.

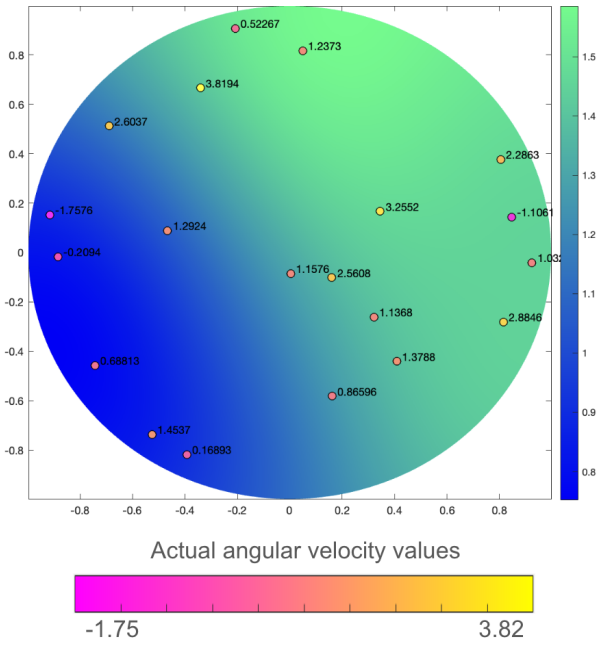

Figure 4: Coarse grained angular velocity with 20 particles

3.1 Fourier Series

We choose Fourier series for the angular and time directions because it allows us to use the Fast Fourier Transform. The FFT takes an input of N complex numbers x_0 to x_{N-1} and produces N outputs X_0 to X_{N-1} such that

$$X_j = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi jn}{N}}. \tag{17}$$

For the angular Fourier series, $f_{nkl}(\theta_j) = e^{-in\theta_j}$ for the jth input angular value. Thus, if we can write $-in\theta_j$ in the form $\frac{-i2\pi jn}{N}$, we can use the FFT to compute the Fourier series coefficients. Setting these two equal yields

$$\theta_j = \frac{2\pi j}{N}. \tag{18}$$

This means that picking N equally spaced input angles between 0 and 2π makes the FFT an effective tool. The T time terms can be similarly selected,

$$t_l = \frac{2\pi l}{T}. \tag{19}$$

Note that the time terms have been evenly spaced over the range 0 to 2π. Rescaling the time axis does not affect the Fourier series used to describe it, and the actual time value in the range 0 to t_{max} can be defined by $t_l' = \frac{t_{max} t_l}{2\pi}$. Like with angles, this technique is useful because all the data we are using is sampled at evenly spaced times. If this were not the case, a different basis may be more effective in the time direction. It is unhelpful to sample the radial terms at evenly distributed points because the area of a section of the disk depends on both the radial width, but also the radial distance from the center. Thus, to avoid oversampling the center of the disk and undersampling the edges, the radial points should not be evenly spaced, causing us to consider bases besides a Fourier series.

3.2 Bessel Functions

Our first choice was to maintain the basis of Bessel functions. This was useful because they are already designed to describe disk functions. However, the derivative of a Bessel function is not another Bessel

function in our basis [4],

$$\frac{d}{dr}J_n(\frac{m_k^n r}{a}) = \frac{m_k^n}{2a}(J_{n-1}(\frac{m_k^n r}{a}) - J_{n+1}(\frac{m_k^n r}{a})). \tag{20}$$

This means that when the derivative is taken, for example when taking the Laplacian of the density, the resulting function is no longer a linear combination of basis functions, meaning that the derivatives must be projected back onto the original basis. This is difficult to resolve analytically, and Matlab's integration is ineffective in calculating these projections.

3.3 Zernike Polynomials

Zernike polynomials are used in optics to describe functions on the disk [7]. They have a radial polynomial component and a trigonometric angular component. Because we have already defined our angular component, we examine only the radial component, using the radial Zernike polynomials,

$$R_{n+2k}^n(r) = \sum_{t=0}^{k} \frac{(-1)^t(n+2k-t)!}{t!(n+k-t)!(k-t)!} r^{n+2k-2t}. \tag{21}$$

We have chosen to index the polynomials using n and $n+2k$ instead of n and k because if the angular index (n) does not have the same parity as the radial index $n+2k$, the polynomial is 0 everywhere. The radial index must also be greater than or equal to the angular index n. These are a useful basis function because they are orthogonal on the unit disk, defined as

$$\int_0^1 R_{n+2k}^n(r) R_{n+2k'}^n(r) r dr = \frac{\delta(k-k')}{2(n+2k)+2}. \tag{22}$$

This means that when taking the inner product of functions defined as sums of Zernike polynomials, there are no cross terms to evaluate. Another helpful property of Zernike polynomials is their connection to Bessel functions [8],

$$\int_0^1 R_{n+2k}^n(r) J_{n+2k}(mr) = (-1)^k \frac{J_{n+2k+1}(m)}{m}. \tag{23}$$

This makes the transition from Bessel functions to Zernike polynomials simple, but quickly and accurately calculating the derivatives of Zernike polynomials proved difficult just as it did for Bessel functions. Although derivatives could be directly calculated because these are polynomials, summing large coefficients over hundreds of terms was time consuming and rounding errors accumulated into large inaccuracies. The final basis we considered was Chebyshev polynomials.

3.4 Chebyshev Polynomials

Chebyshev polynomials are often used because they have a construction that allows the FFT to be used to evaluate them given a specific set of points [9]. Just as the FFT can be used on Fourier series with equidistant points, the FFT can be used on Chebyshev polynomials where points are distributed equidistantly around a semicircle, as shown in figure 5. This means that Chebyshev points are oversampled both near the center and edge of the disk. The Chebyshev polynomials T_n are defined as

$$T_0(x) = 1, \tag{24}$$

$$T_1(x) = x, \tag{25}$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{26}$$

Unlike Zernike polynomials, Chebyshev polynomials are not orthogonal on the unit disk; they are orthogonal with respect to a different weight function. Because we are only taking the inner product of functions over the disk, there will be cross terms of functions defined by Chebyshev polynomials. However, Chebyshev polynomials proved most useful because there was existing infrastructure on Matlab for them as a part of the chebfun package [10]. chebfun is a Matlab package that implements functions in Chebyshev space. A

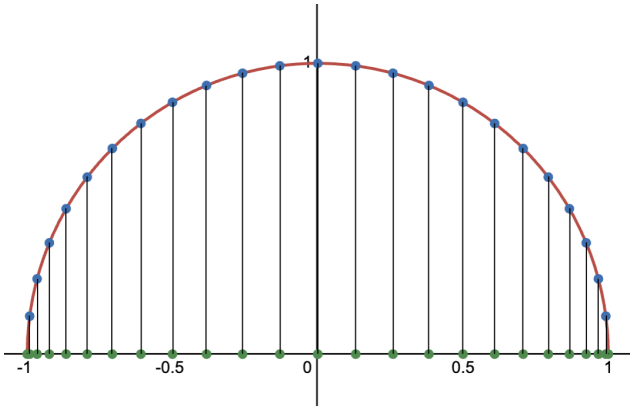

Figure 5: 25 Chebyshev points and corresponding points on the unit circle

subsection of this package is diskfun, which specifically handles functions on the disk. It uses a shifted Chebyshev basis $T'_n(x) = T_n(2x-1)$, which expands to

$$T_0(x) = 1, \tag{27}$$

$$T_1(x) = 2x - 1, \tag{28}$$

$$T_{n+1}(x) = 4xT_n - 2T_n - T_{n-1}. \tag{29}$$

This adjustment projects the range of radial values from $[0, 1]$ to $[-1, 1]$, the range of values on which Chebyshev polynomials are defined.

4 Transitioning Between Real Space and Coefficient Space

Once a basis is chosen, it is important to be able to transition between real values and coefficient space because some calculations cannot occur just with the coefficients. For example, the easiest way to calculate ρ^2 is to turn the coefficients in to real values, square them, and turn the squared values back to coefficients. We implemented this transition when our g basis was Bessel functions, so the process below is based on Bessel functions. However, this process is adaptable to any basis with two indices (each Bessel function has an index for the function number and the extrema number that it is scaled to), and a basis with one index or three indices would simply change the dimensions of some matrices, while the overall process would be consistent.

The transition between coefficient space and real space considers each component of the basis individually, as shown in figure 6.

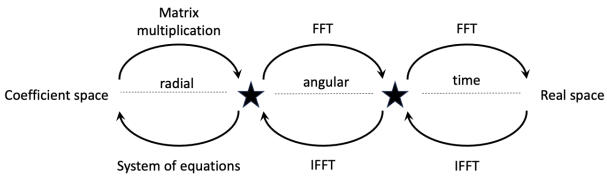

Figure 6: Transition Between Real Space and Coefficient Space

By design, the time and angular components are quickly and accurately computed using the FFT. In order to compute the radial component, we tabulated the values of each Bessel function at every radial input in an array A such that $A_{nkp} = J_n(\frac{m_k^n r_p}{a})$. This is the step that is dependent on the choice of g, and for a

different basis, only the construction of A would be changed; all other steps remain the same. Although the basis contains infinitely many terms, we chose to truncate the angular modes at $N = 60$ and the radial modes at $K = 30$. We also must choose a number of radial sample points, R, though this varies depending on the disk radius and the number of theta samples, Θ. Then, A has dimensions (N, K, R), which is then composed with the coefficient array C with dimensions (N, K, L), where L is the number of time modes, to yield an array B such that

$$B_{npl} = \sum_{k=0}^{K} C_{nkl} A_{nkp}. \tag{30}$$

Then, using the FFT in the first and third dimensions yields an array Q with dimensions (Θ, R, T) such that $Q_{nkl} = \rho(\theta_n, r_k, t_l)$. To recover C from Q, first the inverse FFT is used in the first and third dimensions to reverse the FFT from the opposite transition. These two steps return B. Because B is the composition of two arrays, C can be recovered using A and B. Equation 30 Can be rewritten as

$$B^{(n)} = (A^{(n)})^T C^{(n)}. \tag{31}$$

where $X^{(n)}$ is the matrix of values of X_{n**} . With this notation, each slice of B can be recovered individually. Although $(A^{(n)})^T$ and $C^{(n)}$ are not square, we know that the second dimension of $(A^{(n)})^T$, K, is smaller than its first, R, because we take significantly more radial samples than radial modes. Then, we can consider $(A^{(n)})^T$ in two parts: $(A_1^{(n)})^T$ is the upper half with dimensions (K, K) and $(A_2^{(n)})^T$ is the lower half with dimensions $(R - K, K)$. We divide $B^{(n)}$ similarly to yield $B_1^{(n)}$ with dimensions (K, L) and $B_2^{(n)}$ with dimensions $(R - K, L)$. Then, equation 31 can be rewritten as two different matrix equations,

$$B_1^{(n)} = (A_1^{(n)})^T C^{(n)}, \tag{32}$$

$$B_2^{(n)} = (A_2^{(n)})^T C^{(n)}. \tag{33}$$

Solving these to minimize squared error yields

$$(A_1^{(n)}(A_1^{(n)})^T + A_2^{(n)}(A_2^{(n)})^T)C^{(n)} = A_1^{(n)} B_1^{(n)} + A_2^{(n)} B_2^{(n)}. \tag{34}$$

The left hand side is square and invertible with dimensions (K, K) so this equation can be solved directly for $C^{(n)}$ and by repeating N times, each slice of C can be recovered.

5 Calculating Derivatives

While we ultimately chose to use diskfun to implement these derivatives, we worked through much of the theory behind calculating derivatives before changing directions due to Matlab rounding errors. The main derivative that we wanted to calculate is the Laplacian of the density. Defined in polar coordinates, the Laplacian of the density is

$$\nabla^2 \rho(r, \theta) = \frac{\partial^2 \rho}{\partial r^2} + \frac{1}{r}\frac{\partial \rho}{\partial r} + \frac{1}{r^2}\frac{\partial^2 \rho}{\partial \theta^2}. \tag{35}$$

Therefore, to implement this, we need to be able to implement $\frac{\partial \rho}{\partial r}$, $\frac{\partial \rho}{\partial \theta}$, and $\frac{\rho}{r}$. To do so, we use an array that projects each equation onto the basis we have chosen. Each array is six dimensions and maps a coefficient of a derivative to a coefficient of the original basis. For example, if the derivative with respect to r is defined as

$$\frac{\partial \rho(r, \theta, t)}{\partial r} = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty}\sum_{l=0}^{\infty} C_{nkl} f_{nkl}(\theta)\frac{\partial g_{nkl}(r)}{\partial r} h_{nkl}(t), \tag{36}$$

then $f_{nkl}(\theta)\frac{\partial g_{nkl}(r)}{\partial r} h_{nkl}(t)$ can be projected on to $f_{quv}(\theta)\partial g_{quv}(r) h_{quv}(t)$ to yield a projection coefficient b_{nklquv}. The derivative can then be expressed as

$$\frac{\partial \rho(r, \theta, t)}{\partial r} = \sum_{q=0}^{\infty}\sum_{u=0}^{\infty}\sum_{v=0}^{\infty} C'_{quv} f_{nkl}(\theta) g_{quv}(r) h_{nkl}(t), \tag{37}$$

where

$$C'_{quv} = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty}\sum_{l=0}^{\infty} b_{nklquv} C_{nkl}. \tag{38}$$

By tabulating these coefficients b, we generate an array for each of $\frac{\partial\rho}{\partial r}$, $\frac{\partial\rho}{\partial\theta}$, and $\frac{\rho}{r}$ and by composing them, the input coefficients can be transformed to coefficients representing derivatives and their compositions such as the Laplacian. Because these arrays are independent of values of ρ, they can be stored and applied to multiple problems, and only need to be generated once for a given basis. Despite this convenience, implementing this method proved difficult because the projection arrays stored such large values that multiplying them with other arrays caused rounding errors, and we ultimately used diskfun's functions to generate these derivatives.

6 Sparse Linear Regression

Our goal is to use sparse linear regression to find the combination of spacial derivatives that produce the time derivative. We use linear regression, restricted so that the majority of coefficients are 0. In order to achieve this, we use $L1$-regularization. Linear regression with $L1$-regularization aims to minimize loss of the form

$$L = Zx - D + \lambda \sum_i |x_i|. \tag{39}$$

Here, Z is the library of spatial derivatives and D is the time derivative. λ determines how many terms in x are nonzero. In order to find the most accurate equation, x should be calculated for various values of λ, and the one that provides the most accurate results is chosen.

Because Z and D are stored in real space and we have generated a library in coefficient space, we need to factor in transitions from coefficient space to real space in the calculation of $Zx - D$ by considering both of their expansions,

$$Z(r,\theta,t) = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty}\sum_{l=0}^{\infty} C^z_{nkl} f_{nkl}(\theta) g_{nkl}(r) h_{nkl}(t), \tag{40}$$

$$D(r,\theta,t) = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty}\sum_{l=0}^{\infty} C^d_{nkl} f_{nkl}(\theta) g_{nkl}(r) h_{nkl}(t), \tag{41}$$

$$Z(r,\theta,t)x - D(r,\theta,t) = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty}\sum_{l=0}^{\infty} (C^z_{nkl}x - C^d_{nkl}) f_{nkl}(\theta) g_{nkl}(r) h_{nkl}(t), \tag{42}$$

$$Zx - D = \langle Z(r,\theta,t)x - D(r,\theta,t), Z(r,\theta,t)x - D(r,\theta,t)\rangle, \tag{43}$$

$$Zx - D = \sum_{n,k,l=0}^{\infty}\sum_{n',k',l'=0}^{\infty} (C^z_{nkl}x - C^d_{nkl})(C^z_{n'k'l'}x - C^d_{n'k'l'}) \langle f_{nkl}g_{nkl}h_{nkl}, f_{n'k'l'}g_{n'k'l'}h_{n'k'l'}\rangle_2. \tag{44}$$

Then, minimizing $Zx - D$ becomes a problem of minimizing its coefficients and its inner products. In order to achieve this, we can store the inner products in an array W, where $W_{nkln'k'l'} = \langle f_{nkl}g_{nkl}h_{nkl}, f_{n'k'l'}g_{n'k'l'}h_{n'k'l'}\rangle_2$.

6.1 Evaluating on the Chebyshev Basis

This formula assumes no knowledge of the basis functions f, g, and h, but becomes much simpler knowing that f and h are complex exponentials and choosing g to be a Chebyshev basis. Because f, g, and h are each defined on one variable and one index (f depends only on n, g depends only on k, and h depends only on l), their inner products are evaluated independently. Then, we can rewrite W as $W_{nkln'k'l'} = U_{nn'}V_{kk'}X_{ll'}$ where $U_{nn'} = \langle f_n, f_{n'}\rangle_2$, $V_{kk'} = \langle g_k, g_{k'}\rangle_2$, and $X_{ll'} = \langle h_l, h_{l'}\rangle_2$. Evaluating these yields delta functions,

$$U_{nn'} = \pi\epsilon_n\delta(n - n'), \tag{45}$$

$$X_{ll'} = \pi\epsilon_l\delta(l - l'). \tag{46}$$

The V matrix is the integral of two shifted Chebyshev polynomials. We can write $T'_n(r)$ as the sum of its terms, $T'_n(r) = \sum_{m=0}^{n} c_{mn} r^m$ with c_{nm} as the mth coefficient of the nth shifted Chebyshev polynomial. With this definition,

$$V_{kk'} = \sum_{m=0}^{k} \sum_{m'=0}^{k'} \frac{c_{km} c_{k'm'}}{m + m' + 2}. \tag{47}$$

Now that W is defined, it can be combined with the coefficient matrices C^z and C^d to solve for x.

6.2 Linear Regression in Coefficient Space

The loss equation without $L1$-regularization involving C^z and C^d is

$$L = (C^z x - C^d) W (C^z x - C^d). \tag{48}$$

Then, our goal is to minimize

$$C^z W (C^z x - C^d), \tag{49}$$

and the problem becomes a matrix equation,

$$C^z W C^z x = C^z W C^d. \tag{50}$$

Evaluating $C^z W C^z$ yields a square matrix with dimensions equal to the number of library terms, and the right side of the equation evaluates to a vector with length equal to the number of library terms. Adding in $L1$-regularization can be achieved using alternating direction method of multipliers (ADMM) for Lasso [11], another description for $L1$-regularization.

7 Results

Using this method, we successfully recovered the diffusion coefficient for simulated data. Our only library term was $\nabla^2 \rho$ because the only effect on the particles was diffusion. We generated the data by generating 1000 random particles centered around the Cartesian point $(0.4, 0.4)$. Then, the positions were incremented by stepping in random directions scaled by the diffusion parameter 0.1 and the square root of the time step, 0.05, simulating Brownian motion, which leads to the particles diffusing over 2000 time frames. We generated Fourier-Bessel coefficients by coarse graining space with a parameter $\beta = 0.2$ and time with a parameter $\gamma = 0.1$. Then, we translated the coefficients to real space. diskfun can generate coefficients for real values on the disk in space, but does not also handle slices across time, so we apply the IFFT in the time direction to generate time coefficients. Then, we create diskfun objects for each time coefficient, and generate the laplacian of each diskfun object. Finally we generate a set number of coefficients from each diskfun and its laplacian and store them in a C^z and C^d matrix. Once U, V, and X from section 6.1 have been generated they can be stored and used for all problems. Using the formula in equation 50, $x = 0.0053$. Based on the diffusion parameter, the expected output was 0.005, so this was highly successful.

Next, we considered enhanced diffusion, where the particles both randomly diffuse and also repel each other. The library for this simulation includes $\nabla^2 \rho$, describing the diffusion, and $\nabla^2 \rho^2$ to describe two-particle interactions. To add in a repulsion term, we used a potential equation $U(r) = \frac{1}{re^r}$. Then, the repulsion acting on a single particle was the sum of the forces, or derivatives of potentials, from each particle. The distance between particles was scaled by a parameter l to make sure that interactions were local, and the force was scaled by a parameter α to determine how enhanced the diffusion is. The expected coefficient for the $\nabla^2 \rho$ term remains 0.005, and with $l = 0.05$ and $\alpha = 0.1$, the expected coefficient for the $\nabla^2 \rho^2$ is 0.24975. Using the same method as above, the result is $(0.031, -0.001)$. This is not accurate like the first example was, perhaps because the second coefficient was expected to be so large compared to the first. Another change from the first part is that the time step is 0.0005 in this simulation as compared to 0.1 in the first simulation, an adjustment made so that the repulsion steps did not cause affects that were too extreme.

8 Next Steps

Our goals for this project are to expand our library terms and use this method on data from [6]. In order to do so, we need to make sure that our method works for more than one library term in scenarios where we know the expected result, so we have confidence that the results returned when the true coefficients are unknown are accurate.

References

[1] https://sites.math.rutgers.edu/~speer/527f16/527pde_ch2_sec5.pdf

[2] https://en.wikipedia.org/wiki/Bessel_function#

[3] https://wiki.documentfoundation.org/Documentation/Calc_Functions/BESSELJ/en

[4] https://dlmf.nist.gov/10

[5] https://www.pnas.org/doi/10.1073/pnas.2206994120#supplementary-materials

[6] https://journals.aps.org/pre/abstract/10.1103/PhysRevE.108.014609

[7] https://en.wikipedia.org/wiki/Zernike_polynomials#

[8] https://www.osti.gov/servlets/purl/885409

[9] https://en.wikipedia.org/wiki/Chebyshev_polynomials

[10] https://www.chebfun.org/docs/guide/guide16.html

[11] https://web.stanford.edu/class/ee364b/lectures/admm_slides.pdf