The thing with the Golgi apparatus

Gert-Jan Both

Supervised by:
P. Sens
C. Storm

Abstract

Title: Quantifying the Golgi

Abstract: The Golgi apparatus is a key component in intracellular trafficking, maturing and directing proteins essential to the cell. Despite years of research, a model coupling Golgi size and function to the cells' transport properties is lacking. In this thesis we develop such a model, describing the Golgi as an active droplet. New experimental data sheds more insight in the spatial organization of the trafficking and I have also devised two new methods relying on image gradients and neural networks to analyze this data and confront it with our model.

Contents

1
Introduction

2

Model fitting

In the introduction we stated the question:

How can we fit a model to spatiotemporal data if that model is a PDE?

In this chapter we present the answer in the form of a system which allows us to fit PDE-models to data. We start of with a section describing the general idea and subsequent sections elaborate on each step.

The system principally works for any type of data and model, but was developed originally to analyze data from the RUSH experiments. We illustrate our system with several examples and have chosen to use real data from the RUSH experiments (specifically, MANII) over synthetic data. In figure fig. 2.1 we show four typical frames of the MANII movie. Characteristically, MANII moves from the ER to the Golgi and remains there. The upper left frame shows the first frame with all the cargo in the ER, in the lower left and upper right the intermediate regime is shown (i.e. cargo moving) and in the lower right frame all the cargo has moved into the Golgi.
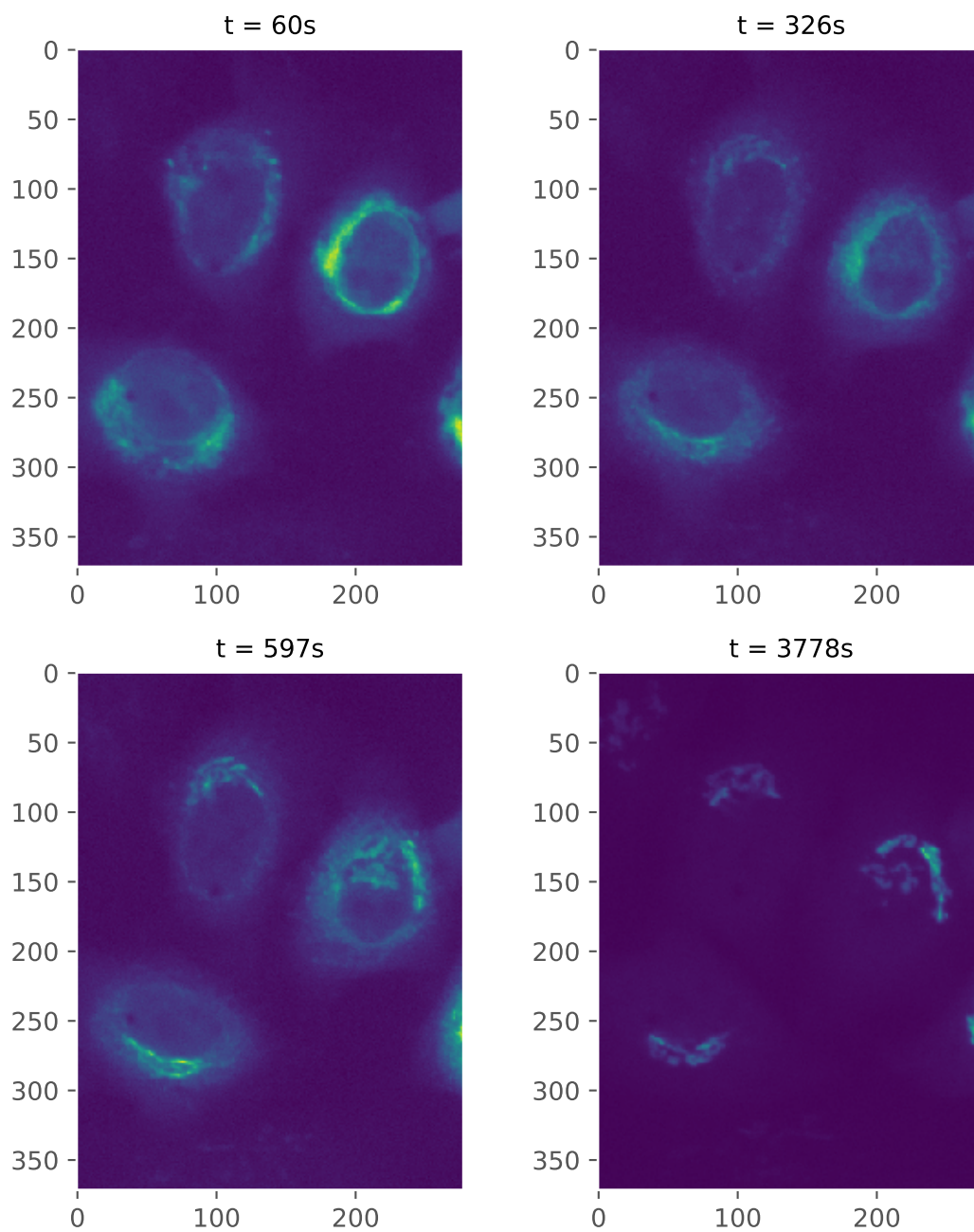
Figure 2.1: Four frames (t = , ,) showing a typical MANII cycle of the RUSH experiment.

2.1 The concept

Assume we have some experimental data describing some process $f(x, t)$. We have also developed a model for this process in the form of a PDE:

$$\partial_t f(x, t) = \lambda_1 \nabla^2 f(x, t) + \lambda_2 \nabla f(x, t) + \lambda_3 f(x, t) + \lambda_4 \qquad (2.1)$$

We now wish to know if this model fits the data $f(x, t)$ and the value of coefficients λ_i. To do so, consider each spatial term of $f(x, t)$ in eq. 2.1 as some variable x_i and $\partial_t f$ as y, so that we can rewrite it as:

$$y = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 + \lambda_4$$

If we have access to the variables x_i and y, we can perform least squares fitting to obtain the coefficients λ_i. In other words, if we can calculate the spatial and temporal derivatives, we can fit the model to the data. Once we know the coefficients, we can use our model to propagate the first frame of the data and compare our model to the experimental data.

Although the concept seems trivial, its implementation is all but. Data is rarely noiseless and obtaining accurate derivatives from noisy data is notoriously hard. We can also have different models in different areas of the data, so that we need to segment the data or the coefficients λ_i might not be constant but space- and time- dependent. The process of fitting the data thus has several steps:

1. Denoising and smoothing
2. Calculating derivatives
3. Segmenting
4. Fitting

In the next sections, we describe each step separately. Note that the method we present here has been developed empirically: there's no theoretical background as to why this particular combination should work optimally. Instead, it's been developed while analyzing the data, adapting each step on the go.

## 2.2 Step 1 - Smoothing and denoising

The first step in our pipeline is to denoise and smooth the data. The smoothing is necessary for accurately calculating the derivatives. Denoising still is a very active area of research

(especially in life sciences) and dozens of different methods exist[1]. For example, one could Fourier transform the signal and use a high pass filter, but this would also get rid of small and sharp features. After evaluating several methods, we have settled on the so-called 'Wav-InPOD' method, introduced by 3 in 2016. In 4 they show that this methods outperforms several other advanced methods. WavInPOD combines Proper Orthogonal Decomposition (POD) with Wavelet filtering (Wav). Both subjects are vast (especially Wavelet transform) and we're only interested in the result of the technique, so we only present a short introduction here, adapted from 3.

POD is a so-called dimensionality reduction technique and is very closely related to the more well-known Principal Component Analysis (PCA) in statistics. In physics it's ofen used to analyze turbulent flows[5]. In POD we wish to describe a function as a sum over its variables:

$$f(x, t) = \sum_{n}^{r} \alpha_n(x)\varphi_n(t)$$

where α_n and φ_n are called respectively the spatial and temporal modes. Associated with each mode n is an energy-like quantity E_n. Modes with a higher 'energy' E_n contribute more to the signal f than modes with a lower energy and we can thus approximate the signal by selecting the k modes with the highest energy. A typical log10 energy spectrum has a 'knee' in the values, as shown in figure fig. ??. Modes with an energy below the knee are noise, and modes above signal.
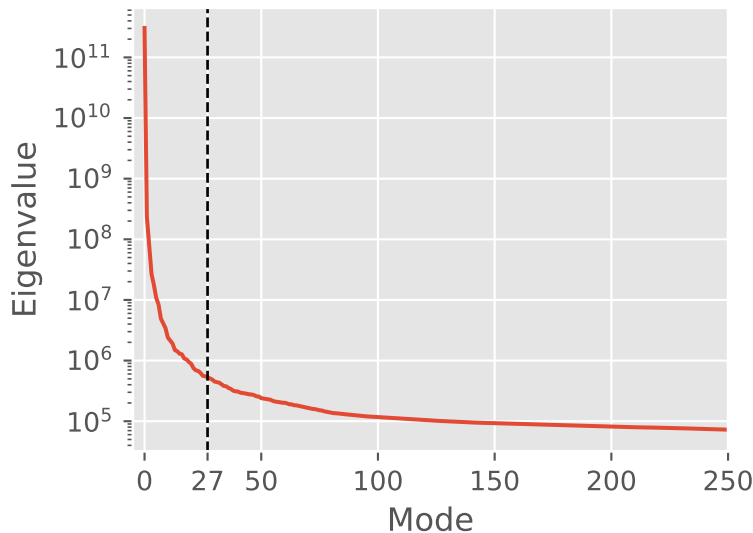
The wavelet transform is very similar to the Fourier transform, but uses wavelets as its basis. A fourier transform gives the frequency domain with infinite precision, but tells nothing about the locality of the frequencies (.i.e when each frequency is present in a signal). By using a wavelet (a wave whose amplitude is only non-zero for a finite time), we sacrifice precision in the frequency domain but gain information on the locality instead. Performing a wavelet transform transforms the signal into the sum of an approximation and its details and we can filter this analogous to a fourier filter. Due to its locality however, noise is filtered out, by sharpness is retained.

WavinPOD combines these two techniques by applying wavelet filtering to the POD modes. In detail, one first decomposes the problem with a POD transformation. The energy spectrum of this transformation is shown in figure fig. ?? and we select a cutoff of 27. All retained modes are wavelet filtered and are then retransformed to give the denoised and smoothed signal. In figure fig. 2.2 we show the results of the smoothing in the time and spatial domain. In the left panel we show the signal of a single pixel in time, while we plot a line of pixels in a single frame in the right panel. The red lines denote the original (unfiltered) signal, the blue line the effect of just applying a POD and the black one the result of the WavInPOD technique. Note that the effect of the wavelet filtering is to smooth the signal significantly and in comparing the original data to the filtered data that we've retained the sharpness of the features whilst obtaining a smooth signal.
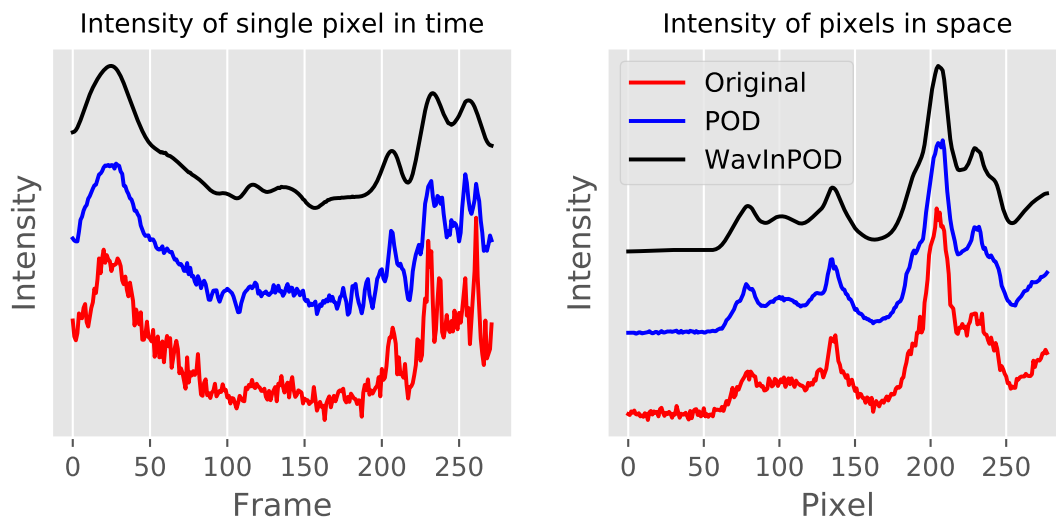

Figure 2.2: Effect of POD with a cutoff of 27 and wavelet filtering with a level 3 db4 wavelet. Left panel shows the result in the time domain, right panel in the spatial domain. Lines have been offset for clarity.

After having denoised the images, we calculate the spatial and temporal derivatives. Obtaining correct numerical derivatives is hard and becomes much more so in the presence of noise[7]. Next to a finite-difference scheme, one can for example (locally) fit a polynomial and take its derivative or use a so-called tikhonov-regularizer[8]. The computational cost of these methods is high however and don't scale well to dimensions higher than one. For our spatial derivatives these methods are thus not available. In fact, obtaining the gradient of a 2D discrete grid has another subtlety which we need to adress.

Naively, one could obtain the gradient of a 2D grid by taking the derivative using a finite difference scheme with respect to the first and second axis. If there are features on the scale of the discretization ($\sim$ few pixels), such an operation will lead to artifacts and underestimate the gradient. These issues have long been known and several techniques have been developed to accurately calculate the gradient of an 'image'. The most-used one is the so-called Sobel operator. It belongs to a set of operations known as *kernel operators*. Kernel operators are expressed as a matrix and by convolving this matrix with the matrix on which the operation is to be performed, we obtain the result of the operator. We show this for the Sobel operator.

Consider a basic central finite difference scheme:

$$\frac{df}{dx} \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$

where h is defined as $x_{i+1} - x_i$. In terms of a kernel operator, we rewrite this as:

$$\frac{1}{2} \cdot \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

where we have set $h = 1$, as the distance between pixels is one by definition. By convolving this matrix with the matrix A we obtain the derivative of A:

$$\partial_x A \approx A * \frac{1}{2} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Figure 2.3: Left panel: One dimensional finite difference kernel. **Right panel:** Three by three Sobel filter

As stated, this operation is inaccurate and introduces artifacts. To improve this, we wish to include the pixels on the diagonal of the pixel we're performing the operation on as well (see figure fig. 2.3). The distance between the diagonal pixels and the center pixel is not 1 but $\sqrt{2}$ and the diagonal gradient also needs to be decomposed into $\hat{x}$ and $\hat{y}$, introducing another factor $\sqrt{2}$. The kernel thus obtained is the classic 3×3 Sobel filter:

$$G_x = \frac{1}{8} \cdot \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Increasing the size of the Sobel filter increases its accuracy and we've implemented a 5x5 operator. Implementing the derivative operation as a kernel method is also beneficial from a computational standpoint, as convolutional operations are very efficient.

As the time derivative involves just one dimension, we make use of a second order accurate central difference scheme implemented in Numpy.

2.4 STEP 3 - SEGMENTATION

In the case of the RUSH data, obtained images and movies often contain multiple cells. Each of these cells can be further segmented into two more areas of interest: the cytoplasm, which is were we want to fit our model and the Golgi apparatus. We wish to make a mask

which allows us to separate the cells from the backgroud and themselves and divide each cell into cytoplasm or Golgi.

Figure fig. 2.1 shows four typical frames in the MANII transport cycle. Note that no sharp edges can be observed, especially once the MANII localizes in the Golgi. No bright field images were available as well, together making use of techniques such as described in 9 unavailable. We have thus developed two methods which allow us to segment the image and the cells, based on Voronoi diagrams and the intensity.

2.4.1 Voronoi diagram

Consider again the frame on the left of figure fig. 2.1. Note that in early frames such as this one, the cargo (i.e. fluorescence) is spread circumnuclear. Applying a simple intensity based segmentation gives us a number of separate areas, which *very* roughly correspond to a cell. We can then pinpoint each cells' respective center. Given n points, Voronoi tesselation divides the frame into n areas, where point i is the closest point for each position in area A_i. The hidden assumption here is thus that each pixel belongs to the cell center it's closest too. Although this is a very big assumption, in practice we've found this to be reasonable. Furthermore, one can add 'empty' points to make the diagram match observations. Assuming small movements of the cell, this isn't an issue either for this technique, as we are assigning an area to each cell instead of very precisely bounding it. This also allows us to calculate the Voronoi diagram in the early frames and apply the segmentation to the entire movie. The result of this segmentation for MANII is shown in figure fig. 2.4. Each cell centre is denoted by a dot, while the lines denote the border between each voronoi cell.
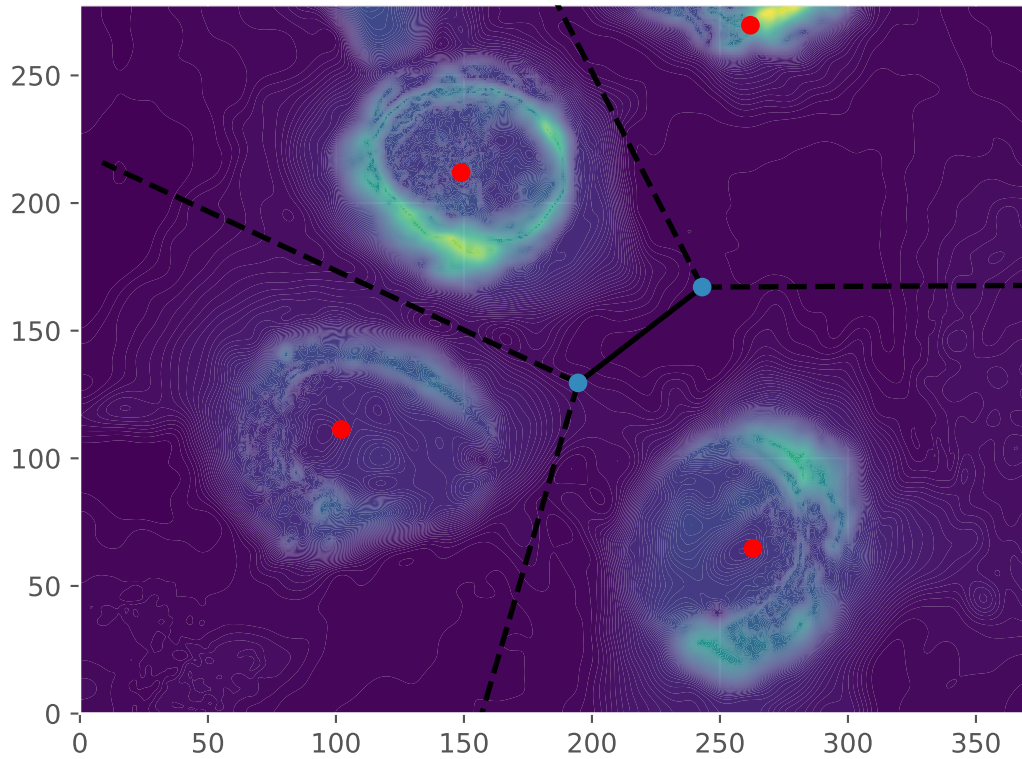
Figure 2.4: The obtained mask. Red dots are cell centers, dashed lines infinite edges and solid lines finite edges.

2.4.2 INTENSITY

The Voronoi technique works very well for an area-based approach, but for analyzing our fitting data we would like a more precise mask - although we still don't require pixel-level accuracy. From the movies, the Golgi is clearly visible and we can separate the cytoplasm from the background, with a big 'gray' area inbetween. We thus turn to an intensity based approach. We have developed the following approach for localizing the Golgi:

1. Normalize the intensity I between 0 and 1.
2. Sum all the frames in time: $\sum_n I(x, y, t_n)$. A typical result is shown in figure fig. 2.5 .
3. Threshold the image to obtain the mask. This is either done automatically through an Otsu threshold or by manually adjusting the threshold until desired result.
4. The mask is postprocessed by filling any potential holes inside the mask.

This procedure was unable to reliably separate the background from the cytoplasm. Noting that while the cytoplasm might not have the intensity as the golgi, its time derivative should be higher than the rest of the areas. We replace step two by $\log_{10}\left(\sum_n I(x, y, t_n) \cdot \partial_t I(x, y, t_n)\right)$, where the time derivative has been normalized between 0 and 1. Figure fig. 2.5 shows our final results. The upper two panels show the images obtained after performing the summing operation for the Golgi and cytoplasm respectively, while the lower left panel shows the final mask obtained after thresholding these two images. For comparison, we plot frame to compare the mask to.
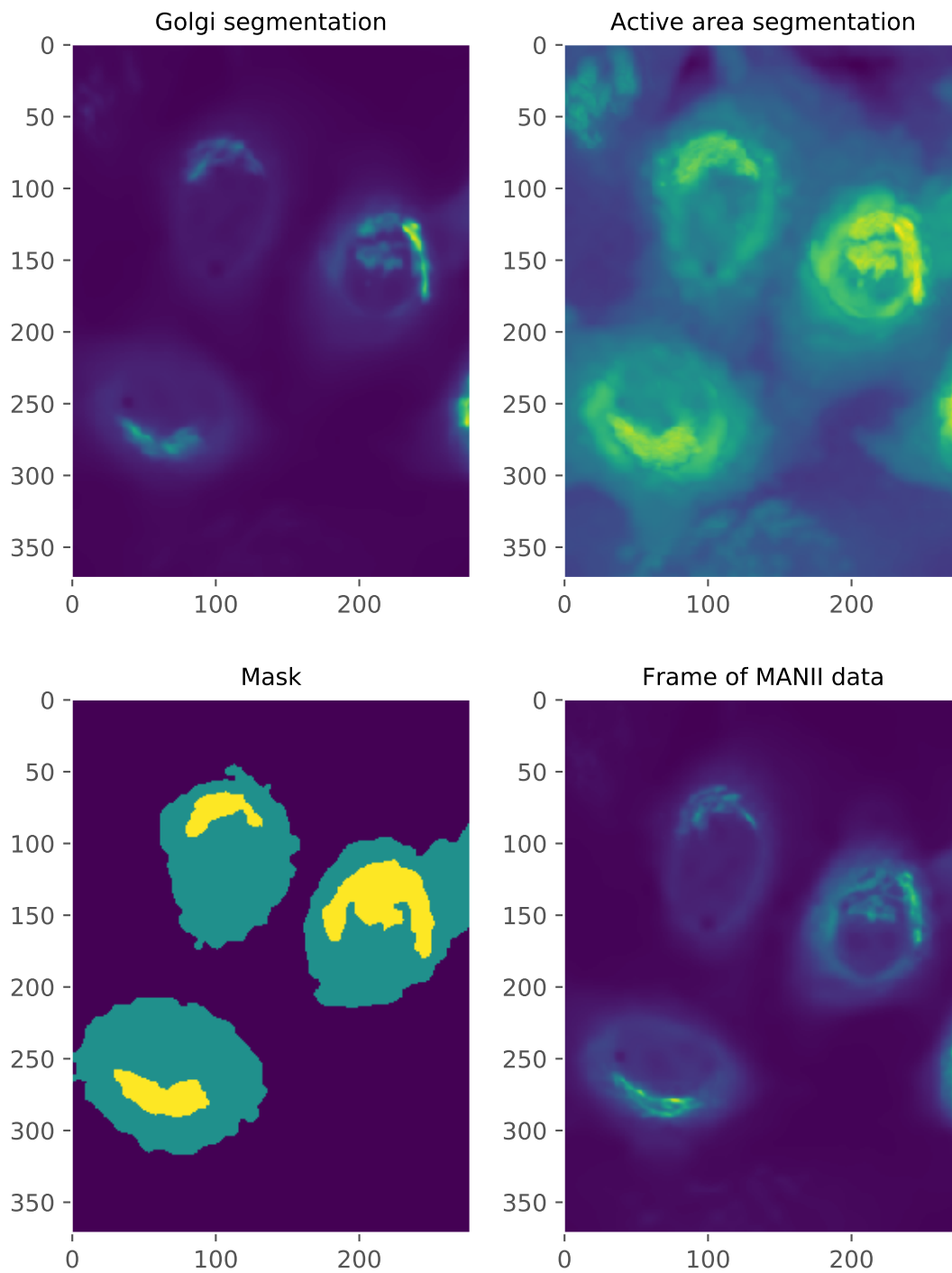
Figure 2.5: Four panels showing the different stages of making the mask. From segmenting the upper two panels we determine the golgi and active area, leading to the mask in the lower left. Compare the to the lower right.

2.5 Step 4 - Fitting

Having gathered all the data, the final step is to fit the model. Equation eq. 2.1 assumes a model with constant coefficients. In reality, coefficients will be spatially and even temporally varying. To circumvent this issue, we assume the coefficients can be assumed to be locally constant. We thus assume that for a small area we can fit the model using constant coefficients. We perform this operation for every datapoint in a sliding-window manner, as shown in figure fig. 2.6, thus ending up with a coefficient field.
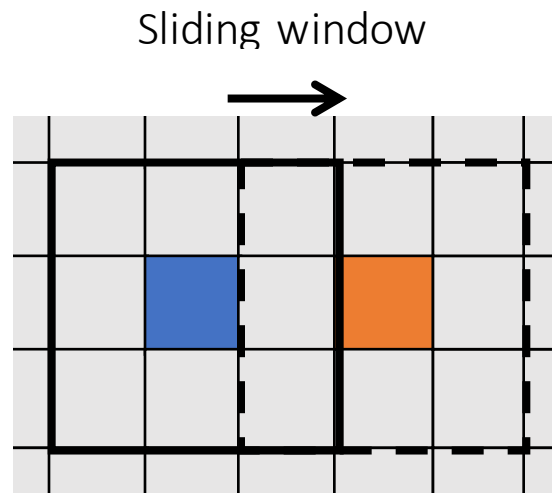
Sliding window

Figure 2.6: Schematic overview of the sliding window technique. The solid black line encompasses an area around its blue coloured central pixel and the fit output is assigned to that pixel. We then move the window (dashed black line) and perform the fit for the orange coloured pixel.

3

Data analysis

In the previous chapter we developed a general system to fit our advection-diffusion model to the data obtained from the RUSH experiments. In this chapter we apply said system to the MANII movies. MANII has been chosen as after arriving at the Golgi, it remains there. Dealing with only one transport direction makes everything a lot easier for us. This chapter consists of five sections:

- Section 1 presents a general analysis of the fluorescence data of the movies
- In section 2 we discuss the time derivative of the fluorescence, $\partial_t I$, as this also has some interesting points.
- Section 3 contains our analysis of the least squares fit.
- Section 4 is similar to section 3, only now we analyse a constrained fit with $D > 0$.
- Section 5 summarizes our findings and presents possible improvements for the system.

3.1 Analysis of fluorescence

We first perform a general analysis of the fluorescence intensity. We plot the average normalized fluorescence (that is, we normalize the fluorescence between 0 and 1 to get rid of the background in our statistics, average each frame and then normalize that

between 0 and 1). The average fluorescence decays with almost 30%, which is a lot. We can also observe two 'regimes': between frame 0 and 100 we roughly see a very strong decay and after frame 100 we see a slower decay. The first regime is where all the cargo moves towards the Golgi, showing that the assumption that $I \propto c$ might not necessarily be true, while the second regime is probably caused by fluorescence extinction.
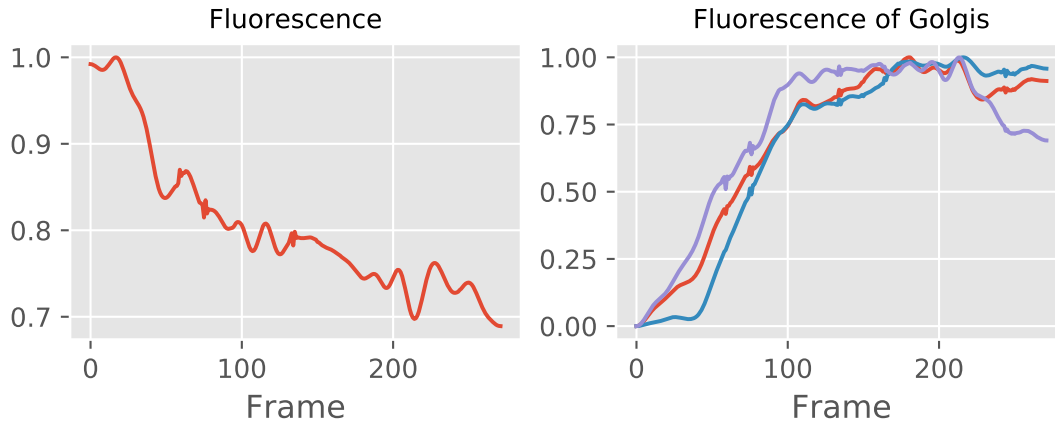
In the right panel we show the fluorescence in each of the golgi bodies. This has been calculated by averaging the fluorescence in each of the golgi for each frame, taking into account the decay of the fluorescence due as shown in the left panel and finally normalizing between 0 and 1. We observe that the fluorescence increase in each of the three cells is nearly similar; although one cells transport starts later than the rest, it seems to catch up to them. ALl three rise mostly before frame 100 and then plateau off. This suggests a common pattern, but what exactly??

3.2 Analysis of time derivative

For the fit we have also determined the time derivatve of the fluorescence. This can also give us some qualitative data. We show 4 frames in figure .
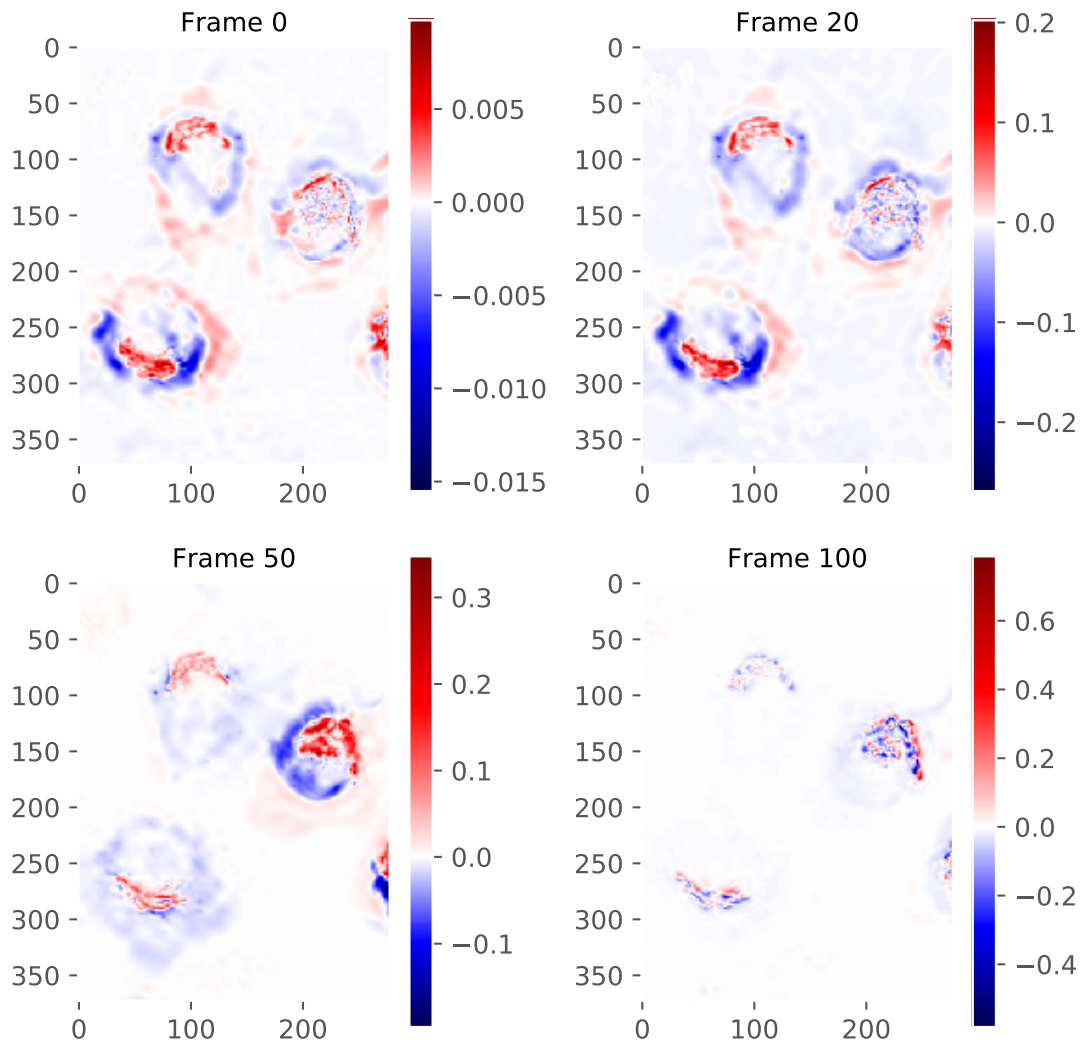
We see that the initial cargo comes from an area close to the golgi and then slowly moves outwards. In the end all we see changing in time is the golgi. This shows that initially stuff gets released from an inner ring and maybe diffusies outward, before it condenses into the golgi.

3.3   ANALYSIS OF LS-FIT

We now turn towards an analysis of our LS fits. We used a window with of 7x7 in the spatial domain and 1 in the temporal domain, i.e. the fit of each frame is independent. We first

show several

3.4 Analysis of constrained LS-fit

3.5 Conclusion

Bayesian heeeeee

4

Physics Informed Neural Networks

In the previous chapters we showed the difficulties in fitting a model in the form of a partial differential equation to spatio-temporal data. The method we developed was a classical numerical approach, separating the problem into several substeps such as denoising, smoothing and numerical differentiating. In the last few years machine learning has been slowly making its way into physics. Very recently, a technique generally referred to as Physics Informed Neural Networks (PINNs) have shown great promise as both tools for simulation and model fitting (10, 11, 12, 13, 14). In this chapter, I will evaluate the use of this technique to fit the model to the RUSH data. I've divided the chapter into three parts:

- Neural Networks - This part will cover the basics of neural networks: their inner workings, how to train them and other general features.
- Physics Informed Neural networks - In this second part we introduce the concept behind PINNs, use it to solve a toy problem and apply it to our RUSH data.
- Conclusion - Finally we summarize the results and observations from the previous sections.

4.1 Neural Networks

Artificial Neural Networks (ANNs) are networks inspired by biological neural networks. Contrary to other ways of computing, ANNs are not specifically programmed for a task -

instead, ANNs are *trained* using a set of data. Research on artificial neural networks started in the '40s but never gained any critical mass, as no efficient training algorithm was known. Once an efficient training algorithm was found in 1975 by Werbos, interest resurged but it wasn't until the late '00s that deep learning started gaining widespread traction. The use of GPU's allowed ANNs to be efficiently trained and widely deployed at reasonable cost.

The advancements in machine learning in general and especially neural networks in the last ten years have yielded a wealth of techniques and approaches. In supervised learning, the network is given pre-labeled data so that it is trained by learning the mapping from the given inputs to the given outputs. Other types such as supervised learning, where the network needs to learns to discriminate between unlabeled data, and reinforcement learning don't have any obvious use for PINNs yet and I've thus chosen to omit them. In the next sections, I'll present the mathematics of an ANN and show how they are trained using the so-called *backpropagation* algorithm.

4.1.1   ARCHITECTURE

An excellent introduction is given by Michael Nielsen in his freely available book "Neural networks and deep learning." The following section has been strongly inspired by his presentation.

At the basis of each neural network lies the neuron. It transforms several inputs non-linearly into an output and we can use several neurons in parallel to create a *layer*. In turn, we several layers in series make up a network. The layers in the middle of the network are known as *hidden layers*, as shown in figure fig. 4.1
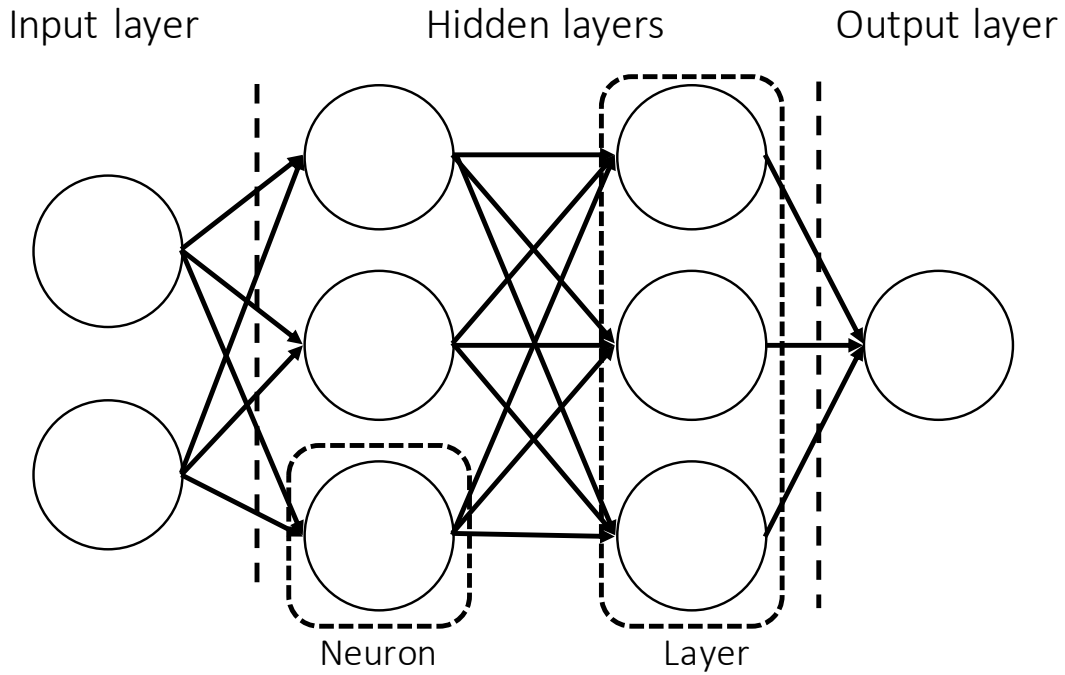
Figure 4.1: Schematic view of a neural network.

In the schematic shown in fig. 4.1, each neuron is connected to every neuron of the previous and next layer. This is known as a *fully connected* layer. Using only this type of layers, we've created a feed-forward network and it has been proven that a single hidden layer with enough neurons is a *universal function approximator*, i.e. a neural network can represent any continuous function using enough neurons.

As stated, a neuron takes several inputs and transforms them into an output. This is a two step process, where in the first step the neuron multiplies the input vector x with a weight vector w and adds a bias b:

$$z = w\mathrm{x} + b \qquad (4.1)$$

z is called the weighted input and is transformed in the second step by the neuron *activation function* σ. This in turn gives the output of the neuron a, also known as the activation:

$$a = \sigma(z) = \sigma(w\mathrm{x} + b) \qquad (4.2)$$

The role of the activation function is to introduce non-linearity into the system. The classical and often used activation function is the *tanh*, as it is bounded between +1 and -1. Since we're working with multiple layers, it is useful to rewrite function eq. 4.2 in terms of the activation a^l of layer l:

$$a^l = \sigma(z^l) = \sigma(w^l a^{l-1} + b^l)$$

where w^l and b^l are respectively the weight matrix and bias of layer l.

4.1.2 TRAINING

In supervised learning the task of training a machine means adjusting the weights and biases until the neural network predictions match the desired outputs. We thus need some sort of metric to define this 'distance' between prediction and desired output. Training the network than means minimizing the metric with respect to the weights and biases of the network. This metric is known as the cost function $\mathcal{L}$ and the most used form is a mean squared error:

$$\mathcal{L} = \frac{1}{2n} \sum_i |y_i - a_i^L|^2 \tag{4.3}$$

where n is the number of samples, y_i the desired output of sample i and a_i^L the activation of the last function - the prediction of the network. Minimizing this is not trivial, as the problem can have many local minima. A solution can be found however using gradient descent techniques.

Gradient descent techniques are based on the fact that given an initial position, the fastest way to reach the minimum from that position is by following the steepest gradient. Thus, given a function $f(x)$ to minimize w.r.t to x, we guess an initial position x_n and iteratively change until it convergences:

$$x_{n+1} = x_n - \gamma \nabla f(x_n)$$

where γ is known as the learning rate. If a global minimum exists, this technique will converge on it. More advanced versions of this technique exist which are able to deal with local minima as well, since convexity of the cost function is not at all guaranteed.

Making use of gradient descent requires knowledge of the derivatives of the cost function w.r.t to the variables to be optimized. In the case of neural networks, we thus need to know

the derivative w.r.t to each weight and bias. A naive finite difference scheme would quickly grow computationally untractable for even shallow networks. A solution to this problem was found by Werbos in the form of the backpropagation algorithm. Despite many years of ongoing research, it is still the go-to algorithm for each neural network implementation.

BACK PROPAGATION AND AUTOMATIC DIFFERENTIATION

As we wish to minimize the cost function w.r.t. to each weight w and bias b using gradient descent, we need to find the derivative of the cost function w.r.t to each. Our argument simplifies if we move away from vector notation and introduce w_{jk}^l, the weight of the j-th neuron in layer $l - 1$ to neuron k in layer l and b_j^l, the bias of the neuron j in the l-th layer. We introduce the error of neuron j in layer l as:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

We can rewrite this using the chain rule as:

$$\delta_j^l = \sum_k \frac{\partial C}{\partial a_{jk}^l} \frac{\partial a_{jk}^l}{\partial z_j^l}$$

However, the second term is always zero except when $j = k$, so the summation can be dropped. Remembering eq. 4.2, we note that $\partial a_{jk}^l / \partial z_j^l = \sigma'(z_j^l)$. For the last layer $l = L$, the first term turns into the derivative of the cost function, finally giving us:

$$\delta_j^L = |a_j^L - y_j| \sigma'(z_j^L) \tag{4.4}$$

Equation eq. 4.4 relates the error in the output layer to its inputs. This in turn is a function of all the previous inputs and errors and we thus need to find an expression relating the error in layer l with the error in an layer $l+1$. Since we have an expression for the error in the last layer, we propagate the error going down the layers, hence the name *back*propagation. Again using the chain rule gives:

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_{jk}^{l+1}} \frac{\partial z_{jk}^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_{jk}^{l+1}}{\partial z_j^l}$$

Using equation eq. 4.1, we obtain after substitution:

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'\left(z_j^l\right) \tag{4.5}$$

Using equations eq. 4.4 and eq. 4.5 , we can calculate the error in C due to each neuron. Finally, we need to relate the error in each error to $\partial C / \partial w_{jk}^l$ and $\partial C / \partial b_j^l$. Making use yet again gives us the last two backpropagation relations:

$$\frac{\partial C}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} = \frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{4.6}$$

$$\sum_k \frac{\partial C}{\partial w_{jk}^l} \frac{\partial w_{jk}^l}{\partial z_j^l} = \delta_j^l \rightarrow \frac{\partial C}{\partial w_{jk}^l} = a_j^{l-1} \delta_j^l \tag{4.7}$$

Now that we now that all back propagation equations, we state the algorithm. It consists of four steps:

1. Complete a forward pass, i.e., calculate the expected outcomes with the current weights and biases.
2. Calculate the error using eq. 4.4 and do a backward pass to obtain the error in each neuron using eq. 4.5. This can be used to calculate the gradients using eq. 4.6 and eq. 4.7
3. Adjust the weights and biases using the choosen optimizer (e.g. gradient descent)
4. Return to step 1 until the optimization problem converges.

Mathematically, back propagation is a special case of a technique known as automatic differentiation. Automatic differentiation is a third type of differentiation, next to numeric and symbolic. It allows for machine precision calculation of derivatives by writing it as a chain of simple operations combined with the chain rule, similar to backpropagation. Note that:

$$\delta_j^o = \frac{\partial C}{\partial x_j} \frac{\partial x_j}{\partial z_j^o}$$

so that:

$$\frac{\partial C}{\partial x_j} = a_j^\circ \delta_j^\circ$$

Thus neural networks also give us access to high precision derivatives with regard to each coordinate.

4.2 Physics Informed Neural Networks

On the face of things, the goal of physics and neural networks are oppsite: whereas physics tries to build an understanding of things using models to make predictions, neural networks learn a *modelless* mapping to make predictions. Recent advancements however have merged the two approaches together in a concept known as Physics Informed Neural Networks (14, 13). In this approach, we encode physical laws into the network, so that the network respects the physics. This can be used to both numerically solve equations or fit a model to spatiotemporal data. Even more so, it should allow us to infer coefficient fields.

4.2.1 The concept

Consider a set of 1D+1 spatiotemporal data, consisting of some property $u(x, t)$ at coordinates (x, t). The neural network can be learned the underlying physics by minimizing the cost function:

$$\mathcal{L} = \frac{1}{2n} \sum_i |u_i - a_i^L|^2$$

The process of learning requires a lot of data and is prone to overfitting. Now assume that we know that $u(x, t)$ is governed by some process which is written as a partial differential equation:

$$\partial_t u = f(1, u, u_x, u_x x, u^2, ...)$$

where f is a function of u or its spatial derivatives. Rewriting it as:

$$g = 0 = -\partial_t u + f(1, u, u_x, u_x x, u^2, ...) \tag{4.8}$$

we see that in order to satisfy the PDE, $g \to 0$. The idea of PINNs is to add this function g to the costfunction of the neural network:

$$\mathcal{L} = \frac{1}{2n} \sum_i |u_i - a_i^L|^2 + \lambda \sum_i |g_i|^2 = MSE + \lambda PI$$

where λ sets the effective strength of the two terms. Observe that the cost function is higher if the PDE is not satisfied. Minimizing the costfunction will thus mean minimizing g and hence satisfying the PDE. We effectively penalize solutions not satisfying the physics we put in equation eq. 4.8; the added term acts a 'physics-regularizer'. Concretely, the adding of physics contrains the solution space, preventing overfitting and making the neural network much more data efficient. The most useful feature however is that we don't need a vast set of training data to train the network, as we solve the problem *by* training the network.

We can also remove the mean squared error term from the cost function and add initial and boundary conditions, similar to the PI term. If we now train the network, it will learn the solution to the given PDE whilst respecting the given boundary and initial conditions. This alternative means of numerically solving a model doesn't need advanced meshing of the problem domain or carefully constructed (unstable) discretization schemes, as it requires the physics to be fullfilled at every point in the spatiotemporal domain. A useful analogy here is calculating the trajectory of a launched object. A classical numerical solver would take small steps in time, updating the position and speed of the object each step. A PINN however uses a completely different approach. Given the initial (random) state of the neural network, it calculates a first trajectory and keeps adjusting the weights of the network until the cost is minimized, i.e. until we obtain a solution satisfying the included physics and initial and boundary conditions. A classical numerical approach tries once using a correct and methodical approach, whereas a PINN tries many times until the result satisfies its constraints.

We can also use this framework to fit models to spatiotemporal data by letting the coefficient of each term be a variable to be minimized as well. More concretely, where before the cost was a function of the weights and biases, $\mathcal{L} = f(w^l, b^l)$, we now let it be a function of the coefficients λ of the PDE as well: $\mathcal{L} = f(w^l, b^l, \lambda_1, \lambda_2, ...)$. This is shown for several PDEs such as the Burgers, Schrodinger or Navier-Stokes equation in the papers of M. Raissi(14, 13). In the case of the Navier-Stokes equation, it's shown that it's also possible to infer the pressure field, which appears as a separate term. This is achieved by adding another output neuron to the PINN (shown in figure fig. 4.2), so that it predicts both the pressure and the flow. In theory it should also be possible to infer spatially and temporally varying *coefficient* fields. We investigate this claim in the next section.
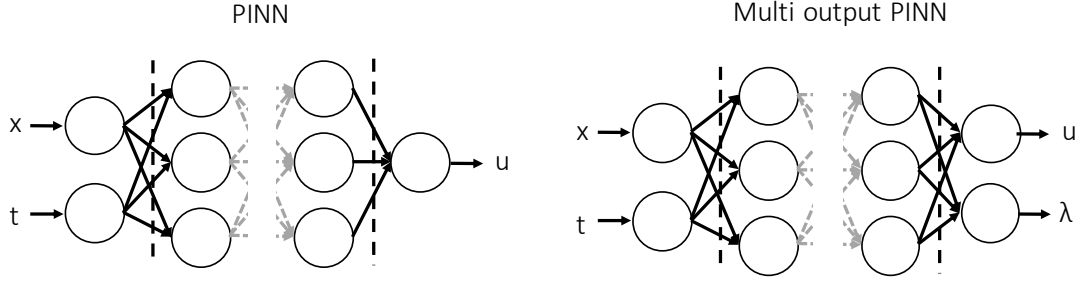
Figure 4.2: Left panel: a normal single output PINN. Right panel: a multi-output PINN. The network now also predicts the coefficients values at each data point.

4.2.2   PINNs in practice

We now wish to evaluate the use of PINNs to analyze the RUSH data. Using a diffusive process as a toy problem, we first show how PINNs are able to accurately determine the diffusion constant, even in the presence of noise. Next, we prove that PINNs are indeed capable of inferring coefficient fields and finish by analyzing some parts of the RUSH data.

In our toy problem we have an initial concentration profile:

$$c(x, 0) = e^{-\frac{(x-0.5)^2}{0.02}}$$

diffusing in a 1D box according to:

$$\frac{\partial c(x, t)}{\partial t} = \nabla \cdot [D(x)\nabla c(x, t)]$$

on the spatial domain $[0, 1]$ with perfectly absorbing boundaries at the edges of the domain:

$$c(0, t) = c(1, t) = 0$$

If $D(x) = D$, this problem has an analytical solution through a Greens function. If the diffusion coefficient is spatially dependent though, the problem needs to be solved numerically. The code used to generate our data can be found in the appendix. Although this toy problem is simple and in 1D, our results easily generalize to higher dimenions and complexity at the cost of higher computational cost.

We now consider the mentioned problem with a diffusion coefficient of $D(x) = D_0 = 0.1$ and simulate it between $t = 0$ and $t = 0.5$. Using a spatial and temporal resolution of 0.01, we have a datagrid of 101 by 51, so that our total dataset consists of 5151 samples. The neural network consists of 6 hidden layers of 20 neurons each and $\lambda = 1$. Figure fig. 4.3 shows the ground truth for the problem and the absolute error of the neural network.
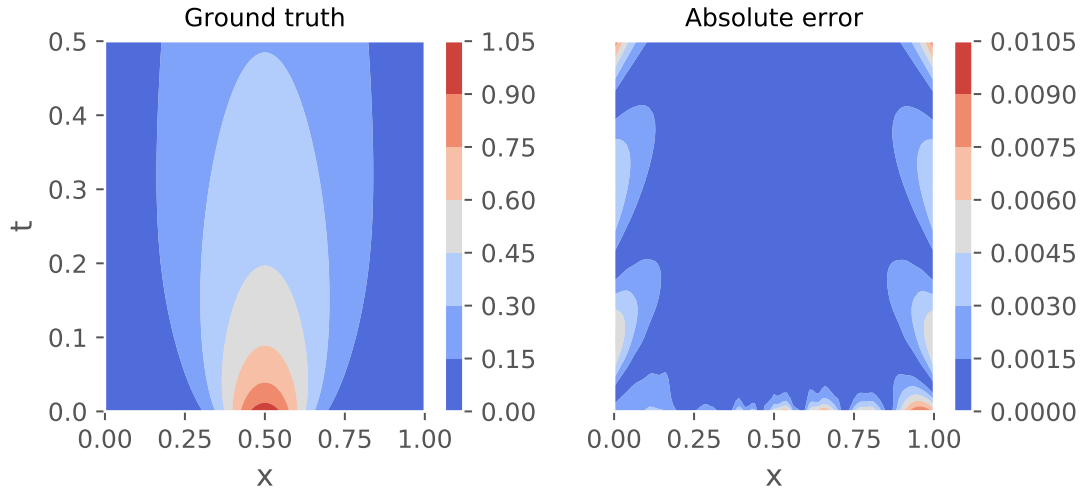

Figure 4.3: Left panel: Simulated ground truth of the problem. **Right panel**: The absolute error of neural network. Note that most of the error is located at areas with low concentration, i.e. signal.

The predicted diffusion coefficient is $D_{pred} = 0.100026$, giving an error of 0.026%. In 14, the authors obtain similar accuracies for significantly more complex problems such as the Schrodinger equation, which means that our accurate inference is not just due to the simplicity of the problem. Furthermore, Raissi et al. show that the result is robust w.r.t the architecture of the network. From the absolute error we observe that the error seems to be higher in areas with low concentration. This is a feature we've consistently observed: in areas with low 'signal', the neural network seems to struggle.

As good as these results are, the input data is noiseless and thus of limited practical interest. We now show that PINNs perform equally well with noisy data by adding 5% white noise to the data and performing the same procedure. The network is now doing two tasks in parallel: it's both denoising the data and performing a model fit. In the left panel of figure fig. 4.4 we show the concentration profile at times $t = 0, 0.1$ and 0.5, with the prediction of the PINN superimposed in black dashed lines at each time. On the right panel we show

again the absolute error from the ground truth. Observe the similarities with the noiseless case: most of the error localizes in areas wit low concentration.
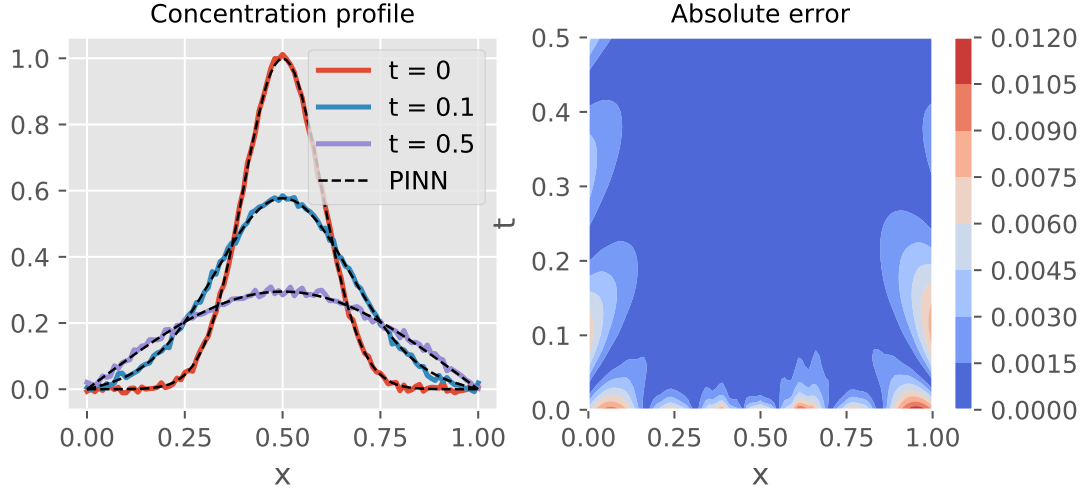

Figure 4.4: Left panel: The original noisy concentration profile with the neural network inferred denoised version super imposed. **Right panel**: The absolute error of neural network with respect to the ground truth. Note that most of the error is located at areas with low concentration.

The inferred diffusion constant is $D_0 = 0.10052$, giving an error of 0.52%. Although the error is slightly higher than in the noiseless version, it's extremely impressive that we obtain the diffusion constant to this precision.

4.2.2.2 VARYING D

As stated, it should be possible to infer coefficient fields by using a two output neural network. One output predicts the concentration while the other predicts the diffusion coefficient. Such a network is indeed capable of generating the right coefficient field as shown in figure fig. 4.5 . Here the network has been trained on the constant diffusion coefficient data we used before including 5% white noise, so that we should observe a diffusion field constant at $D(x, t) = D_0 = 0.1$. In the upper left we show the data on which the network is trained, with the upper right panel the predicted concentration profile, which shows a very good match. In the lower right panel we show the inferred diffusion field. We observe a good match in the middle of the plot, but the neural network again struggles in areas with low concentration, such as the lower left and right area. A more quantitative analysis of the predicted diffusion and concentration is presented in the lower left corner. Here we plot

the Cumulative Distribution Function (CDF) of the absolute relative error. Note that the PINN predicts the concentration very well, but struggles more with the diffusion coefficient. This is expected, as the mean squared error of the cost function is quite explicit in its use of the concentration, whereas the diffusion coefficient is determined self-consistently in the PI part. We also observed similar but distinctive results in different runs, owing to the non-convexity of the problem. Overall the result is still remarkable, given that we've inferred a diffusion field from just concentration data with 5% noise.
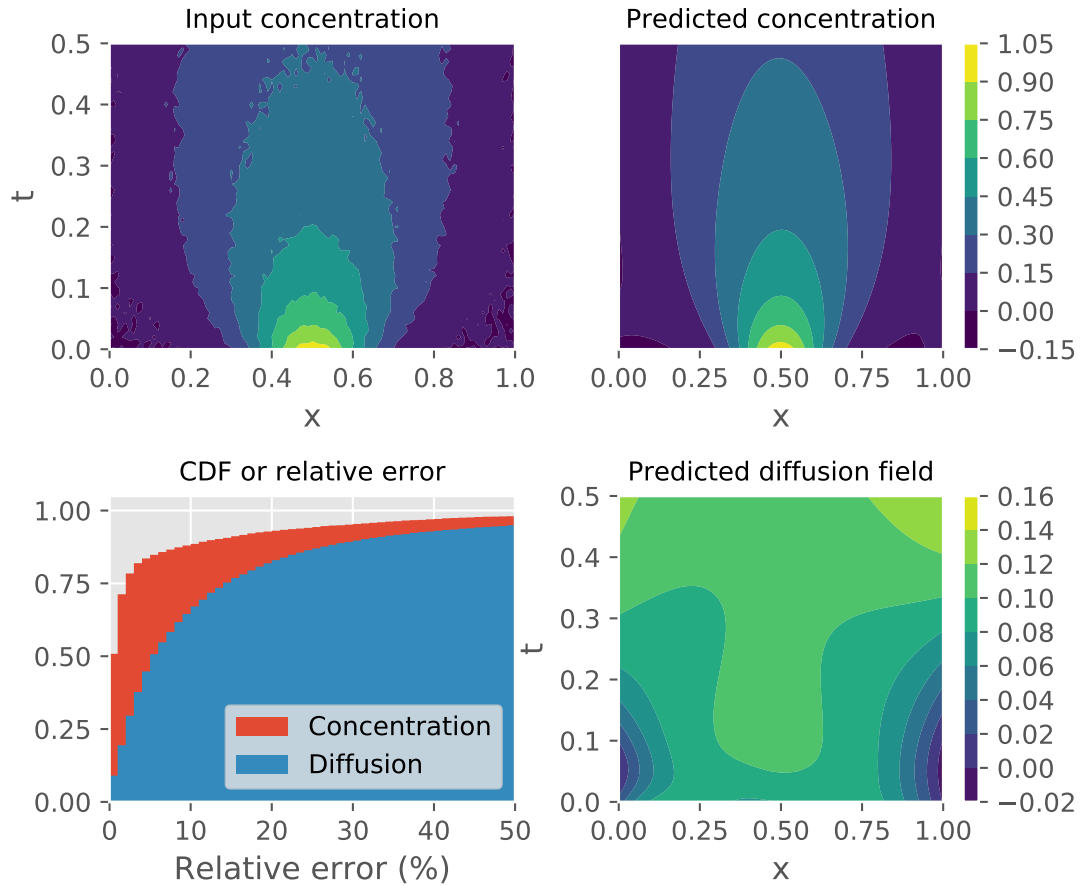
Figure 4.5: We show the training data and predicted concentration profile in the upper left and right panels. The lower right panel shows the inferred diffusion field while the lower left panel shows the CDF of the relatice error of the diffusion and concentration.

Input concentration

Predicted concentration

CDF of relative error

Relative error (%)

Concentration
Diffusion

Predicted diffusion field

Projection of time axis
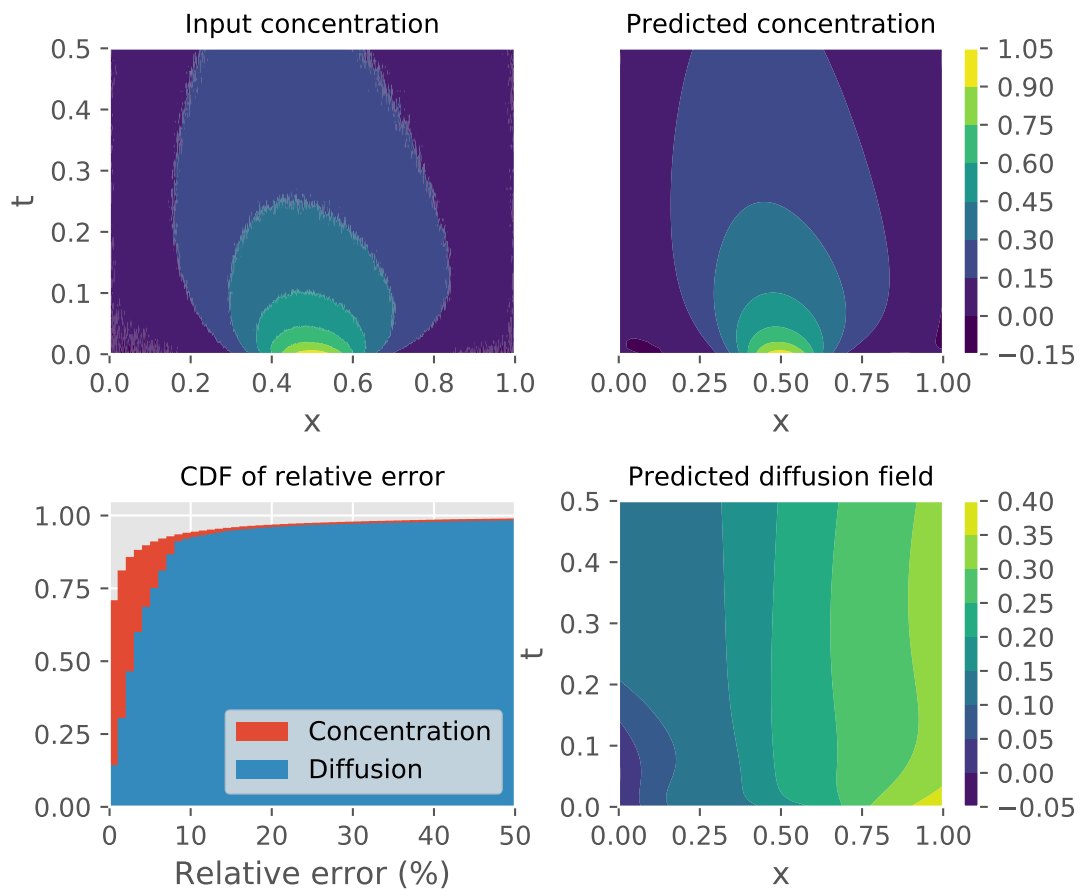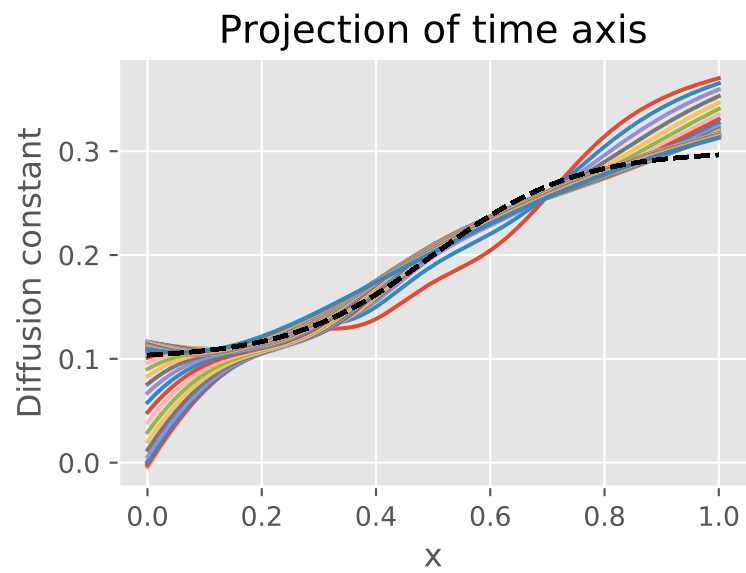
4.2.2.3   Real cell

4.3   Conclusion

4.3.1   Weak points and how to improve

5

Introduction to Golgi as a phase separated droplet

In this part of the thesis we develop a model relating the Golgi function and size to its transport properties. This chapter serves as a background to the next, both on biological and physical perspective. In the first section we state the general idea of the model and some observations from the biological observations. The second section obtains a short introduction to phase separation and introduces the Cahn-Hilliard equation.

5.1 Considerations of Golgi for general model

In no particular order, there's a few notable things about golgi which warrant a phase separation description:

- The Golgi is able to form de novo (Bevis)
- Microtubules position the golgi near the mtoc (Sengupta)
- Nocadazola depolymerizes microtubules -> golgi stacks move towards ERES (sengupta)
- Golgi size is dependent on amount of trafficking (Sengupta)
- Separate ministacks seems to be fully functional (Wei)
- Golgi disassembly is due to imbalanced trafficking: exit from ER is blocked, while outflow still continues (Ronchi)

- When golgi is completely cut out, stack like structures form around ERES (Ronchi) and then later move towards mtoc.

We think we can describe this using an active droplet. Our model basically consists of two parts: We hypothesize that we can describe the transport between the ER and the golgi as a advection-diffusion equation and use a Cahn-Hilliard equation (see next section) to describe the phase separation. We thus need a a way to couple the transport model to the phase separation model.

5.2 Phase separation

Consider a mixture of two molecules, type A and B. One is the solvent and one the solute. One can then try and use statistical mechanics to derive a mean-field model and get all sorts of cool results. This is quite complex and not particularly enlightening for us. Instead, we follow a method devised by Landau. We know that phase separation occurs when the free energy has two convex region, so we can construct a free energy density function f with two minima in a minimalist way as:

$$f(c) = \frac{b}{2(\Delta c)^2}(c - c_o^-)^2(c - c_o^+)^2$$

where b characterizes interactions, $\Delta c = |c_o^- - c_o^+|$, and $c_o^{\pm}$ is where the two minima of the function are located. Figure shows this free enery function. A free energy like this leads to phase separation into a dense and dilute phase. The interface has surface tension, so when we calculate the full free energy of the system we add a term for the interface:

$$F(c) = \int dV(f(c) + \frac{1}{2}\varkappa\nabla^2 c)$$

We thus have constructed a Landau free energy with order parameter c. The first term accounts for the bulk energy, while the second term penalizes interfaces. Suittable choices of parameter lead either to a mixed state, where $\bar{c} = constant$ in the whole system, or to a phase separated state such as shown in figure. When we quench a system from a mixed state into a phase-separated state, this happens through a process called *coarsening dynamics*. Basically small domains form and these keep growing, showing a maze-like structure as in figure .... In liquid-liquid phase separation, the order parameter is conserved, as the molecules cant just change. The phases then exchange diffusion-like and can only exchange locally, so we state that:

$$\partial_t c = -\nabla \cdot \mathbf{j}$$

where j is a flux. We know that j is related to the to the chemical potential through:

$$j = -\Lambda \nabla \mu$$

where Λ is an Onsager coefficient. We also know that $\mu = \delta F / \delta \varphi$ so that we end up with

$$\frac{\partial \varphi}{\partial t} = m \nabla^2 \frac{\delta F}{\delta \varphi}$$

where $\delta / \delta \varphi$ is a functional derivative. Given the landau free energy in eq , we obtain:

$$\frac{\partial \varphi}{\partial t} = m \nabla^2 [f'(\varphi) - k \nabla^2 \varphi]$$

which is known as the Cahn-Hilliard equation. The Cahn-hilliard equation is what governs the maze domain evolution shown in figure and is the basis for studies of phase separation. Due to its non-linearity its very hard to use, and many times a scaling relation is derived. However, since we want to include spatial inhomogenity, this probably will not work. We present a different approach using effective droplets in the next section.

6

Effective droplet

In the previous chapter we introduced the Cahn-Hilliard equation. Its non-linearity makes it extremely hard to study and in this chapter we present an approximation known as effective droplet theory. In this chapter we introduce the effective droplet approximation, discuss our models for the Golgi apparatus and show the results for these models. We've divided the chapter into three sections:

- Effective droplet theory - In this section we introduce effective droplet theory.
- Golgi as an effective droplet- Here we introduce our model and show
- Two-component model- This third section contains a possible extension of the effective droplet model to two-components.

We end the chapter with a short summary of our conclusions.

6.1   Effective droplet

Consider phase separation in a one-dimensional box. The system will separate into a dilute and a dense phase, separated by an interface of width w:

$$c^*(x) = \frac{c_o^- + c_o^+}{2} + \frac{c_o^+ - c_o^-}{2} \tanh\left(\frac{x}{w}\right)$$

The idea of the effective droplet model is that if the width of the interface w is very small, we neglect the interfacial contribution and describe the system as two separate bulk phases. In each bulk phase we then solve a linearized version of the Cahn-Hilliard equation and match the solutions at the interface of the phases. Growth of the droplet can then be described in terms of fluxes across the interface. We show this in figure fig. ??.

We first linearize the Cahn-Hilliard equation in the bulk phases. Consider again the Cahn-Hilliard equation, where we've changed the order parameter φ to a concentration c:

$$\frac{\partial c}{\partial t} = -m\nabla J$$
$$J = -\nabla \mu$$

$$\mu = f'(c) - k\nabla^2 c \tag{6.1}$$

where $f'(c) = \partial f / \partial c$. We're assuming an infinitely thin interface, so the interfacial term is neglegible. Linearizing the chemical potential in c around the dense phase yields $\mu = f''(c_0^+)c$, so that eq. 6.1 becomes:

$$\frac{\partial c}{\partial t} = D\nabla^2 c \tag{6.2}$$

where we've replaced all the constant in front of the right term by D. We thus see that linearizing the Cahn-Hilliard equation in the bulk leads to a diffusion equation. The effective droplet theory is most accurate in predicting steady states and thus the time-dependence of eq. 6.2 is ignored. As we've replaced a single equation by two, we need an extra set of boundary conditions at the interface.

6.1.1 BOUNDARY CONDITIONS

To determine the droplet boundary conditions, consider a system which is phase separated and has an infinitely thin interface. The total free energy of the system is then:

$$F = V_1 f(\varphi_1) + V_2 f(\varphi_2)$$

where V_i and φ_i are respectively the volume and density of phase i and $f(\varphi_i)$ is the free energy density. In equilibrium, this free energy is minimized. Assuming incompressibility $(V_1 + V_2 = V)$ and conservation of particles $(V_1\varphi_1 + V_2\varphi_2 = V\varphi)$, we further constrain the

system to two independent variables. Choosing to minimize the free energy with respect to φ_1 and V_1 gives us two conditions:

$$f'(\varphi_1) = f'(\varphi_2) = 0$$

$$0 = f(\varphi_1) + f(\varphi_2) + (\varphi_2 - \varphi_1)f'(\varphi_2)$$

Since $f'(\varphi) = \mu(\varphi)$, the first condition states that both phases must have the same chemical potential, while the second one refers to the matching of osmotic pressure. One obvious solution to these equations is a completely mixed state with $\varphi_1 = \varphi_2$. A non-trivial (phase-separated) solution exists as well, where φ_1 and φ_2 are the two minima of the free energy density function $f(\varphi)$. In our system, this corresponds to c_o^+ and c_o^-. Assuming that our droplet is locally in thermodynamic equilibrium, we can apply this solution as boundary conditions. Concretely,

$$c(R) = \begin{cases} c_o^-, & \text{Outside droplet} \\ c_o^+, & \text{Inside droplet} \end{cases}$$

Now that we have defined boundary conditions, equations such as eq. 6.2 can be solved. The solution will give us the concentration profile in each phase, but the droplet behaviour is determined by the fluxes across the interface. We show this in the next section.

6.1.2   FLUXES AND MOVEMENT OF INTERFACES

Given a concentration profile $c(x)$, the (diffusive) flux can be calculated by applying Ficks' law:

$$J(x) = -D\frac{\partial c}{\partial x}$$

Using this expression, we can calculate a flux on the inside of the interface J_{in} and on the outside of the interface J_{out}. Note that in and out respectively refer to inside and outside the droplet and not to the direction of the flux. Figure ???: eff_droplet shows a typical concentration profile of an active droplet. Observe that due to the activity of the droplet, the concentration profile inside the droplet is not flat but convex. Due to this convexity, the flux J_{in} will be pointed inwards, while the flux J_{out} will also be pointed inwards typically due to transport or production. Intuitively, J_{in} is the flux lost due to the decay or activity in

the droplet, while J_{out} replaces the converted material inside with material from outside. A steady-state can be achieved once these two fluxes are equal. Note that while the system is at steady state, it is not in equilibrium, as the fluxes across the interface are not zero; the fluxes are balanced, rather than equilibrated. This is a typical characteristic of an active system.
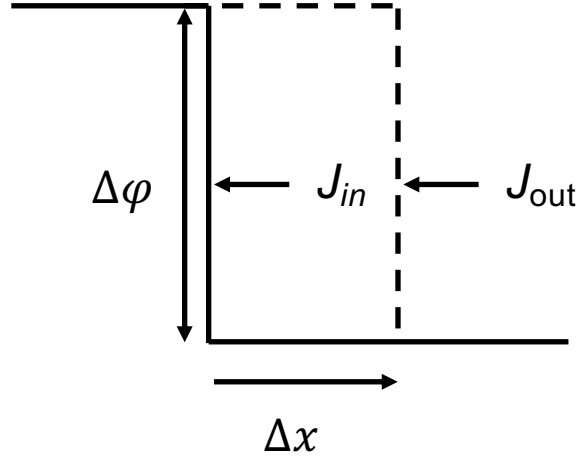

Figure 6.1: Figure illuminating the relation between a moving interface and the fluxes across it.

If the fluxes J_{in} and J_{out} are not balanced, a net flux exists, leading to either growth or decay of the droplet. We wish to derive an expression for interface speed v_n in terms of the fluxes across it. Consider figure fig. 6.1. If we wish to move the interface a distance Δx, we need a net material gain of $\Delta x \Delta c$. This net gain is supplied by a net flux in a time Δt, so that:

$$\Delta x \Delta c = (J_{in} - J_{out})\Delta t$$

which can be rewritten as:

$$\frac{\Delta x}{\Delta t} = v_n = \frac{J_{in} - J_{out}}{\Delta c} \tag{6.3}$$

Thus if $|J_{in}| < |J_{out}|$, $v_n > 0$ and the interface will move to the right. If we consider a 'free' droplet (with two interfaces in 1D), the radius of the droplet is determined by both interfaces. Since the fluxes need not be similar on both sides of the droplet, not only the radius but also the position of interfaces plays a role. We thus characterize the droplet in terms of its radius R and its geometric center x_0. Consider a droplet of radius R at position

$x_\circ$. The left and right interfaces move respectively with velocities v_l and v_r. In a time dt, the droplet than moves to a new postion $x_\circ + dx$ and will have a new radius $R + dR$:

$$x_\circ - R + v_l dt = x_\circ + dx - (R + dR)$$
$$x_\circ + R + v_r dt = x_\circ + dx + (R + dR)$$

Solving this set of equations for dx and dR gives:

$$\frac{dR}{dt} = \frac{1}{2}(v_r - v_l) \tag{6.4}$$

$$\frac{dx_\circ}{dt} = \frac{1}{2}(v_l + v_r) \tag{6.5}$$

Equations eq. 6.4 and eq. 6.5 give the change in radius and position of the droplet. Note that these results are quasi-steady state: we've calculated the fluxes from stationary concentration profiles. Thus equations eq. 6.4 and eq. 6.5 are best used to calculate (quasi-) steady states and not dynamics. Nevertheless, equation eq. 6.5 shows an interesting feature: active droplets can move on their own. This movement is the result of an imbalance in fluxes between the left and right interface: the droplet center is displaced because one side of the droplet grows faster than the other. This imbalance is caused by a concentration gradient and droplets will move up the gradient, as the flux on the high concentration side will be higher than on the low side. Finally combining eq. eq. 6.3 with eqs. eq. 6.4 and eq. 6.5 yields the position and radius of the droplet in terms of the fluxes across its interface:

$$\frac{dR}{dt} = \frac{1}{2\Delta c} \left[(J_{in}^{x=R} - J_{in}^{x=-R}) + (J_{out}^{x=-R} - J_{out}^{x=R}) \right]$$

$$\frac{dx_\circ}{dt} = \frac{1}{2\Delta c} \left[(J_{in}^{x=-R} + J_{in}^{x=R}) - (J_{out}^{x=-R} + J_{out}^{x=R}) \right]$$

This completes the effective droplet model. It allows us to rewrite the non-linear Cahn-Hilliard equation into two or mroe separate linear problems with proper boundary conditions. Furthermore, these linear problems can be easily extended, such as by adding decay, sources or advection and we will do so in the next section. To solve a typical problem with the effective droplet theory, we thus perform the following steps:

- Identify the domains and correct equations.

- Solve equations inside each domain to obtain concentration profile.
- Calculate the fluxes across the interfaces.
- Obtain steady states at the balance of fluxes.

In the next section we present our model for the Golgi in the effective droplet model.

6.2 EXPANDING EFFECTIVE DROPLET: ADVECTION AND DECAY

We use and modify the effective droplet model as a toy model for the behaviour of the Golgi in the cytoplasm. More precisely, the Golgi apparatus in mammalians is a ribbon, consisting of multiple golgi stacks (identical) together. We model a stack as an active droplet in a concentration gradient. Our model is shown in figure . We consider a 1D box of length L with an influx at $x = 0$ of J_{in} and a zero flux boundary at $x = L$. Inside the system is a droplet of radius R at position x_0. Transport in the system is described by a advection-diffusion-decay equation with a diffusion constant D, advection speed v and decay constant a outside and k inside the droplet. The phase separation is governed by a free energy density with two minima at c_0^- (dilute phase) and c_0^+ (dense phase). We thus end up with the following set of equations and boundary conditions in a quasi steady state approach:

$$D\partial_x^2 c(x) - v\partial_x c(x) - kc(x) = 0 \tag{6.6}$$

$$D\partial_x^2 c(x) - v\partial_x c(x) - ac(x) = 0 \tag{6.7}$$

$$(-D\partial_x c + vc)|_{x=0} = J_{in}$$

$$(-D\partial_x c + vc)|_{x=L} = 0$$

$$c(x_0 \pm R) = \begin{cases} c_0^-, & \text{inside of interface} \\ c_0^+, & \text{outside of interface} \end{cases}$$

We solve this set of equations in the next section.

6.2.1 SOLUTION

The general solution of equations eq. 6.6 and ??? is given by

$$c(x) = C_1 e^{-\frac{x}{l^-}} + C_2 e^{\frac{x}{l^+}}$$

where we have defined a lengthscale l as:

$$l^{\pm} = \frac{2D}{\sqrt{4kD + v^2} \pm v} \tag{6.8}$$

where ofcourse k should be replaced by a outside the droplet. This lengthscale defines the typical lengthscale of the problem, defined by the diffusion, advection and decay. Setting the decay to zero, $l^{\pm} \to D/v$ so that both lengthscales become similar. Also, setting $v \to 0$ gives $\sqrt{D/k}$, which up to a dimension dependent numerical factor is similar to Zwickers.

Applying the boundary conditions and calculating the flux at position $x_0 + R$ and $x_0 - R$ gives the following fluxes outside:

$$J_{out}^{x=-R} = J_{in} \frac{\left(1 + \frac{l_-}{l_+}\right) e^{\frac{-(x_0 - R)}{l_-}}}{Pe_- + Pe_+ \frac{l_-}{l_+} e^{\frac{-x_1}{l}}} + \frac{c_0^{out} D}{l_+} \frac{Pe_+ \left(1 - e^{\frac{-x_1}{l}}\right)}{1 + \frac{l_-}{l_+} \frac{Pe_+}{Pe_-} e^{\frac{-x_1}{l}}} \tag{6.9}$$

$$J_{out}^{x=R} = -c_0^{out} D \frac{Pe_- Pe_+ \left(1 - e^{\frac{-x_2 + L}{l}}\right)}{l_+ Pe_- + e^{\frac{-x_2 + L}{l}} l_- Pe_+} \tag{6.10}$$

where we have introduced the coordinates x_1 and x_2, which are defined respectively as $x_0 \pm R$, in other words, the left and right interface the Peclet-like numbers $Pe^{\pm}$:

$$Pe^{\pm} = 1 \mp \frac{v l^{\pm}}{D}$$

and the 'average lengthscale' $l = \frac{l^+ l^-}{l^+ + l^-}$. Again note that if $v \to 0$, $Pe^{\pm} \to 1$ We can recognize that the second term of eq. 6.9 and eq. 6.10 are similar, due to the same boundary conditions. The flux on the left however has the added effect of the influx, clearly visible in the first term. We present approximations of these functions in the next sections, to make them more clearly.

For the fluxes inside, the expressions are less clear. Note however from eqs ???: that we're not interested in the separate expressions but their sum or difference. Introducing $J_{Rad} = J_{in}^{x=R} - J_{in}^{x=-R}$ and $J_{Pos} = J_{in}^{x=R} + J_{in}^{x=-R}$, we obain:

$$J_{rad} = \frac{-2c_o^+ D}{l} \frac{\sinh \frac{R}{l^-} \sinh \frac{R}{l^+}}{\sinh \frac{R}{l}}$$

$$J_{pos} = 2c_o^{in} D \left[\frac{Pe_-}{l_-} \frac{\sinh \frac{R}{l_+} \cosh \frac{R}{l_-}}{\sinh \frac{R}{l}} - \frac{Pe_+}{l_+} \frac{\sinh \frac{R}{l_-} \cosh \frac{R}{l_+}}{\sinh \frac{R}{l}} \right]$$

We study all these equations in depth in the following sections, but please note that everything so far is completely analytic; we have made no approximations. (But the whole theory is an approximation, ha-ha.)

An important point about the validity of effective droplet theory must now be made, which is most clear without advection, although our general remarks hold with advection as we'll see. Setting $v = 0$ and taking the limit of $J_{rad} \to \infty$, we get:

$$\lim_{R \to \infty} = -2c_o^+ \sqrt{kD}$$

where the 2 arises from two interfaces. In other words: the influx due to decay saturates, which is not what we would expect. The answer can be found found by studying the minimum concentration in the droplet. For $R \gg l$, the concentration in the center of the drop goes to zero, which is unphysical and an artifact of the effective droplet model. Thus we always require $R \ll l$. When advection is added this becomes slightly more complicated, as we show in the next section.

6.2.2   NO DECAY IN THE DILUTE PHASE: THE EFFECT OF ADVECTION

We first study the effect of advection our test system but without decay outside, i.e. $a = 0$. In this case, $l^+ = l^-$ and the influx becomes becomes constant on the left side, i.e. J_{in}^{x=-R} = J_{in} and the outflux on the right side 0. Since we're working in a quasi steady state and there's no decay outside, this is expected.

$$J_{in}^{x=-R} = J_{in}$$

$$J_{out}^{x=R} = 0$$

while the equationa for the inner flux remain unchanged because they are independent of the decay outside. Armed with these equations we can study the effect of advection. In figure .. we show the minimum concentration, dx_0/dt and dR/dt as a function of R and v.

In the left panel we show the minimum concentration as a function of R and v. The white line shows the concentration corresponding to $l = 0.5$ at $v = 0$ and we indeed see that advection increases the minimum concentration. In the middle panel we show the increase in radius dR/dt as a function of again R and v. Red colors mean positive, blue negative and white zero, so we've plotted with x's the line $dRdt = 0$, which corresponds to a stable droplet. On the right we show dx_0/dt; where the $dx_0/dt = 0$ line crosses with the $dR/dt = 0$ line there's a droplet with stable radius and position.

To show this more clearly, we plot the droplet radius and minimum concentration as a function of the advection velocity in figure . Observe that advection decreases the droplet radius by almost 10%, while simultaneously also increasing the minimum concentration. The advection is thus 'compacting' the droplet. By what process is this happening? To see this, consider an active droplet without advection. The concentration profile is symmetric and convex; the flux on both sides due to diffusion is inwards and thus in opposite directions. If we now turn on advection, this symmetry is broken. On the left side of the droplet, the diffusion and advection point in the same direction but on the other side they are aligned opposite. This causes an assymetry in the concentration and a difference in the flux to the centre: more goes to it from the left than is taken away by the right, hence compacting the droplet.

Can advection stabilize a droplet? Zwicker et al show that an active droplet such as ours can divide through a shape instability. Our model is 1D however and not easily scalable to higher D's because of the advection. Simulations would be needed to state anything on this.

We can make a few conclusions about the situation. For a very specific set of parameters, the droplet can both be stable in radius and position. There's no position dependence so not that interesting. Also observe that there are only two situations: either the droplet is moving to the left infinitely, or moving to the right infinitely, as there's no stable position on either side. Which situation happens is controlled by the gradient of the concentration versus the advection. note that we need a non-zero convection to move the droplet from one mode to the other. The explanation is simple: the convection needs to overcome the gradient to reverse droplet movement. At the inflection point, $dx_0/dt = 0$.

We now make some very speculative biological connection to the observation of Golgi properties. A strong marker is that once the microtubules are depolymerized, the stacks move away from the perinuclear area and colocate with the ERES. This is what we see when we turn of the advection, the droplet moves up the gradient to the source. A similar thing

is seen with biogenesis, the stacks are made around the ERES but need microtubules to be transported to the center.

We have a model which described some features but its unstable. Can we make some position in our system inherently stable? We try in the next section by adding decay to the dilute phase.

6.2.3   DECAY IN THE DILUTE PHASE

We now wish to find a model in which the position has a stable position. So not just a set of parameters where $dx_o/dt = 0$, but a position x_o where the droplet is stable. The flux on the inside is independent of the position of the droplet, so to get a stable position we need a position dependent flux on the outside. To this end, we introduce a decay term in the dilute phase. We then end up with the full equations ... . We first numerically solve these equations before attempting an approximation. The result is shown in figure ... and shows a 'phase-diagram' of our active droplet model. In the left panel we show the minimum concentration in the droplet, while the middle and right panel shows the numerical value of dx_o/dt and d_R/dt, all as a function of the CoM position x_o and radius R. From the right two plots we've extracted the lines $dx_o/dt = 0$ and $dR/dt = 0$, which we've superimposed on all three panels. Where these lines cross a stable state exist. We first discuss each line independently and then discuss the stable points.

First consider the line $dR/dt = 0$, a droplet with stable radius. We observe that left of the line $dRdt > 0$, with the opposite on the right. This means that each radius is stable: a perturbation w.r.t to the radius will not propagate. The $dx_odt = 0$ is more interesting. It has the apperance of a 'finger' and has a stable and unstable branch. The arm below the point is stable, where the arm above the point is not. This has important implications for the crossings: the lower intersection is stable w.r.t. to all perturbations, while the upper intersection is unstable. Pertubations above the dxodt line will propagat, while perturbations below will move the droplet to the lower intersection.

This is all fine and dandy, but there's one problem: the lower intersection, the stable point, is unphysical. We characterize the droplet in terms of its center of mass x_o and radius R, meaning that the left interface is at a position $x_o - R$. The red line in the plots shows the line $x_o = R$ and everything below this line is unphysical, as it means the left interface is at a position $x_o - R <$, past the system edge. Why does this happen? First, in a dynamical description this wouldn't happen, as the left interface boundary condition prevents moving past. In a static description this can however since we specify the edge of the system as a flux boundary condition. Inspection of the concentration (see figure) learns that the droplet moves on top of the source: the flux inside the droplet at a point is similar to the boundary

condition. In 2D this wouldnt be a problem but in our 1D description it is as the BC is specified as a flux and not as a source. That means we're left with just an unstable point. Why is this point unstable? Investigating the fluxes shows that despite the decay the left interface is an order of magnitude bigger than the right interface, thus behaviour is mainly determined by the left interface. Once the left interface starts moving due to a perturbation, the rest follows.

6.3 TWO-COMPONENT MODEL

Two components jeeej

6.4 CONCLUSION

7
Conclusion

Appendix 1: Some extra stuff

Add appendix 1 here. Vivamus hendrerit rhoncus interdum. Sed ullamcorper et augue at porta. Suspendisse facilisis imperdiet urna, eu pellentesque purus suscipit in. Integer dignissim mattis ex aliquam blandit. Curabitur lobortis quam varius turpis ultrices egestas.

References

1. Garcia, D. Robust smoothing of gridded data in one and higher dimensions with missing values. *Computational Statistics & Data Analysis* 54, 1167–1178 (2010).

2. Sbalzarini, I. F. Seeing Is Believing: Quantifying Is Convincing: Computational Image Analysis in Biology. in *Focus on Bio-Image Informatics* (eds. De Vos, W. H., Munck, S. & Timmermans, J.-P.) 219, 1–39 (Springer International Publishing, 2016).

3. Zimoń, M., Reese, J. & Emerson, D. A novel coupling of noise reduction algorithms for particle flow simulations. *Journal of Computational Physics* 321, 169–190 (2016).

4. Zimoń, M. *et al.* An evaluation of noise reduction algorithms for particle-based fluid simulations in multi-scale applications. *Journal of Computational Physics* 325, 380–394 (2016).

5. Grinberg, L., Yakhot, A. & Karniadakis, G. E. Analyzing Transient Turbulence in a Stenosed Carotid Artery by Proper Orthogonal Decomposition. *Annals of Biomedical Engineering* 37, 2200–2217 (2009).

6. Grinberg, L. Proper orthogonal decomposition of atomistic flow simulations. *Journal of Computational Physics* 231, 5542–5556 (2012).

7. Bruno, O. & Hoch, D. Numerical Differentiation of Approximated Functions with Limited Order-of-Accuracy Deterioration. *SIAM Journal on Numerical Analysis* 50, 1581–1603 (2012).

8. Knowles, I. & Renka, R. J. METHODS FOR NUMERICAL DIFFERENTIATION OF NOISY DATA. 12

9. Rizk, A. *et al.* Segmentation and quantification of subcellular structures in fluorescence microscopy images using Squassh. *Nature Protocols* 9, 586–596 (2014).

10. Karpatne, A., Watkins, W., Read, J. & Kumar, V. Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling. *arXiv:1710.11431 [physics, stat]* (2017).

11. Sharma, R., Farimani, A. B., Gomes, J., Eastman, P. & Pande, V. Weakly-Supervised Deep Learning of Heat Transport via Physics Informed Loss. *arXiv:1807.11374 [cs, stat]* (2018).

12. Pun, G. P. P., Batra, R., Ramprasad, R. & Mishin, Y. Physically-informed artificial neural networks for atomistic modeling of materials. *arXiv:1808.01696 [cond-mat]* (2018).

13. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. *arXiv:1711.10566 [cs, math, stat]* (2017).

14. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv:1711.10561 [cs, math, stat]* (2017).