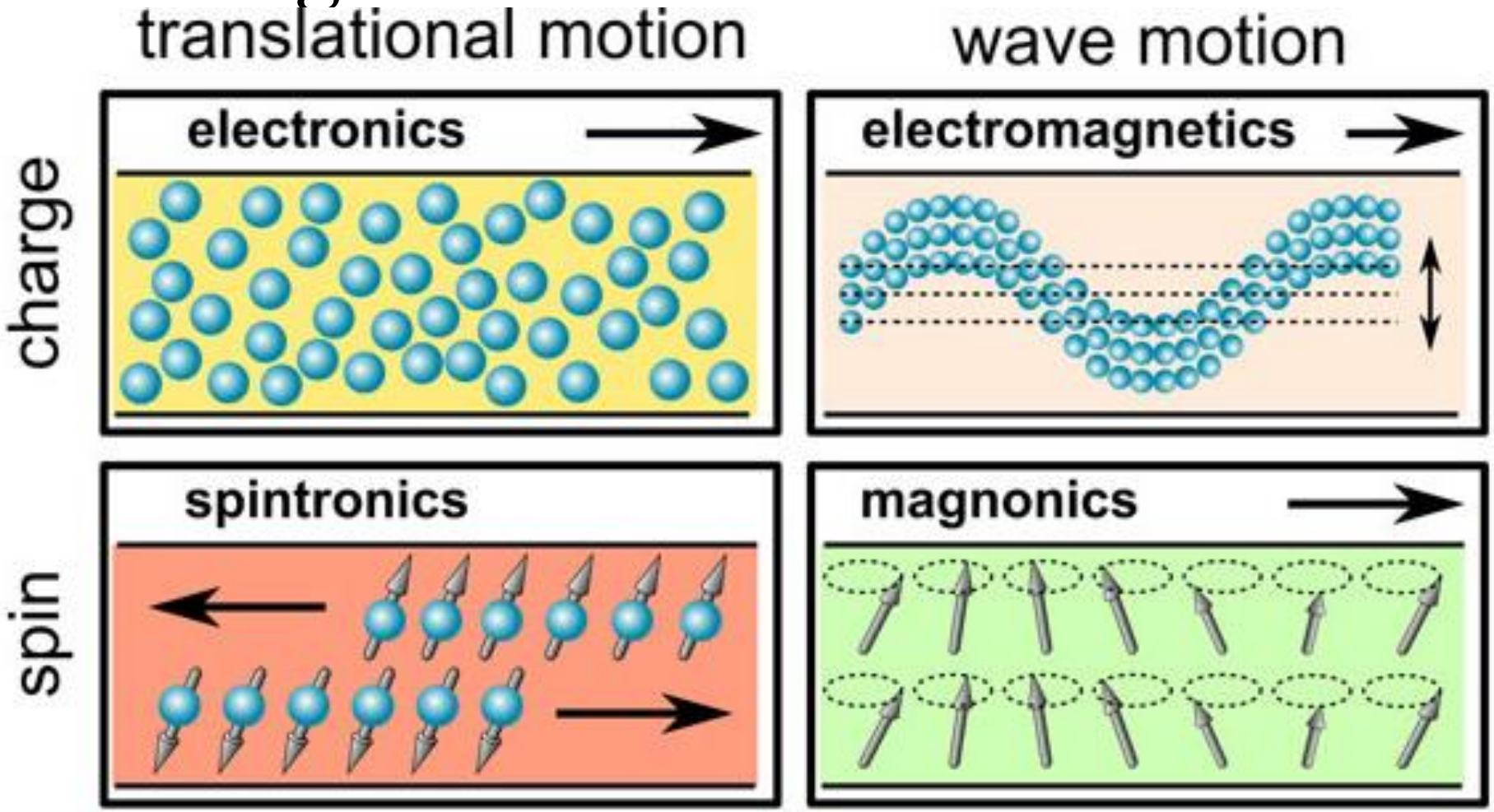
Sparse Identification of Dynamics, the LLG Equation

Vismay Churiwala

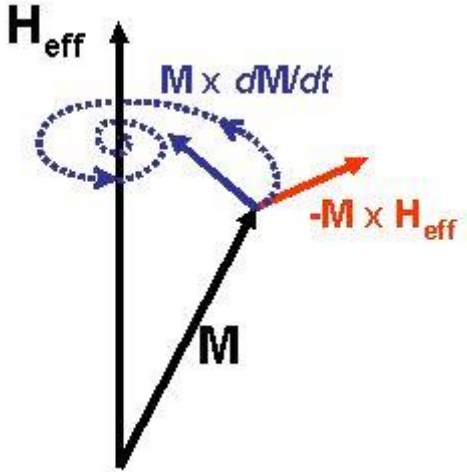
-under the guidance of
Prof Anil Prabhakar

Magnonics


translational motion
wave motion
charge
spin
electronics
electromagnetics
spintronics
magnonics

The Landau–Lifshitz–Gilbert equation

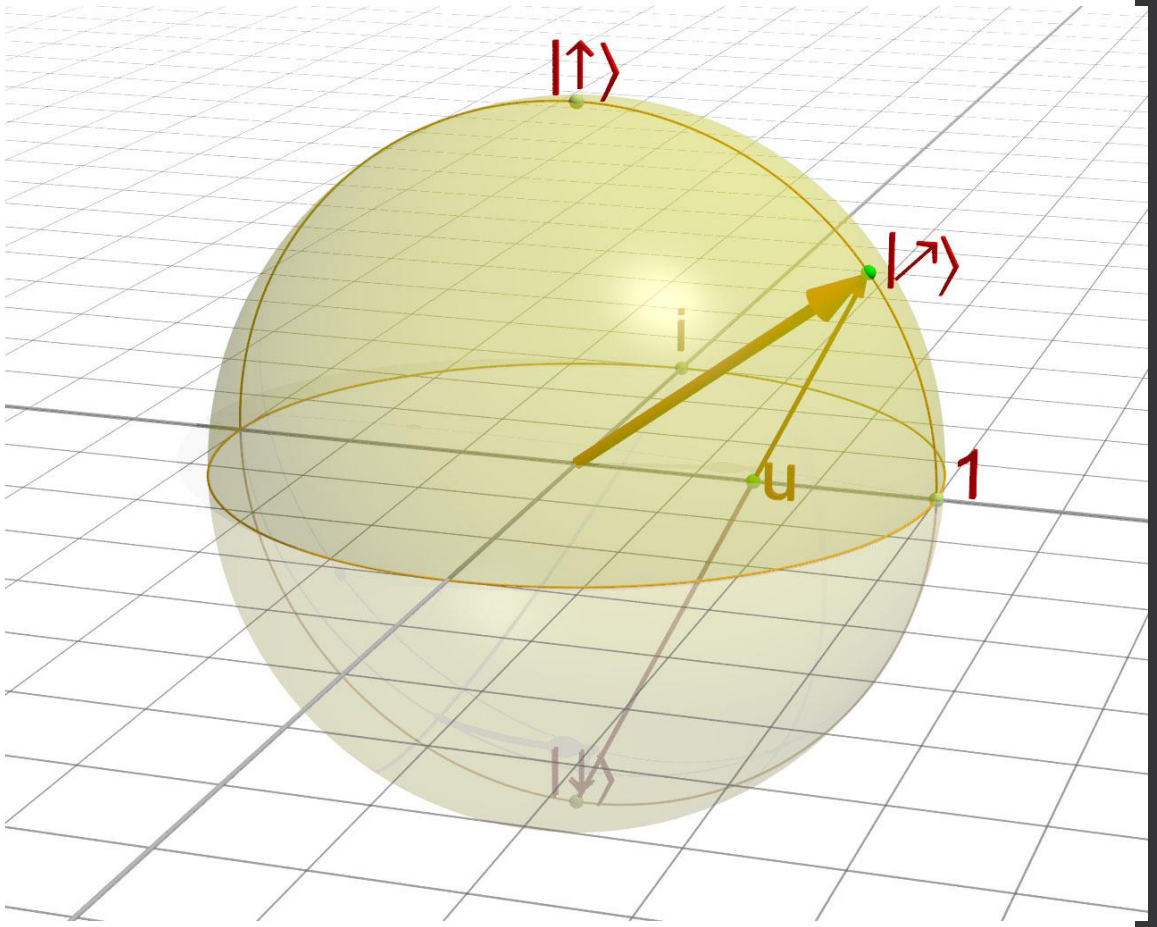
$$\frac{\partial m}{\partial t} = -\gamma m \times H_{eff} + \alpha m \times \frac{\partial m}{\partial t}$$



The constant α is the Gilbert phenomenological damping parameter and depends on the solid, γ is the electron gyromagnetic ratio, m = M/Ms.
The second term represents the damping that causes the amplitude of the spin waves to reduce over distances

The Bloch Sphere

- The resulting magnetization obtained from the LLG equation is normalized, and the time evolution of the state can be interpreted as a spin traversing on the Bloch sphere
- The effective magnetic field appearing in the equation is the sum of various components, including the external magnetic field applied, the demagnetization field, anisotropy field etc.



Sparse Identification of Non-Linear Dynamics (SINDy)

- A large number of systems that are of interest are differential equations with a relatively few number of dominant terms
- A new data-driven approach is given by SINDy, using sparse-regression to determine the governing equations using the least number of terms, balancing accuracy with model complexity to avoid over-fitting
- Here, we consider dynamical systems of the form:

$$\frac{d}{dt}x(t) = f\big(x(t)\big)$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} = \overrightarrow{\begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_n(t_m) \end{bmatrix}} \text{ state}$$

time

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \dot{\mathbf{x}}^T(t_2) \\ \vdots \\ \dot{\mathbf{x}}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \cdots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \cdots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \cdots & \dot{x}_n(t_m) \end{bmatrix}.$$

State Matrices

SINDy: Methodology

- We sample x(t) and $\dot{x}$(t) over several times and arrange them into matrices
- Next, we construct a library Θ(X) consisting of candidate nonlinear functions of the columns of X. For example, Θ(X) may consist of constant, 2 polynomial, and trigonometric terms:

$$\Theta(\mathbf{X}) = \begin{bmatrix} | & | & | & | & & | & | & \\ 1 & \mathbf{X} & \mathbf{X}^{P_2} & \mathbf{X}^{P_3} & \cdots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \cdots \\ | & | & | & | & & | & | & \end{bmatrix}$$

This sets up a sparse regression problem where we need to determine the sparse vectors of coefficients Ξ = ξ1ξ2..ξn to determine which nonlinearities are active.

$$\dot{X} = \Theta(X)$$

Identifying the LLG Equation using SINDy

Setting up the Problem

- The 3 components of magnetization are determined by the LLG equation, where we can approximate it by considering terms up to the 2nd order. The resulting equation is:

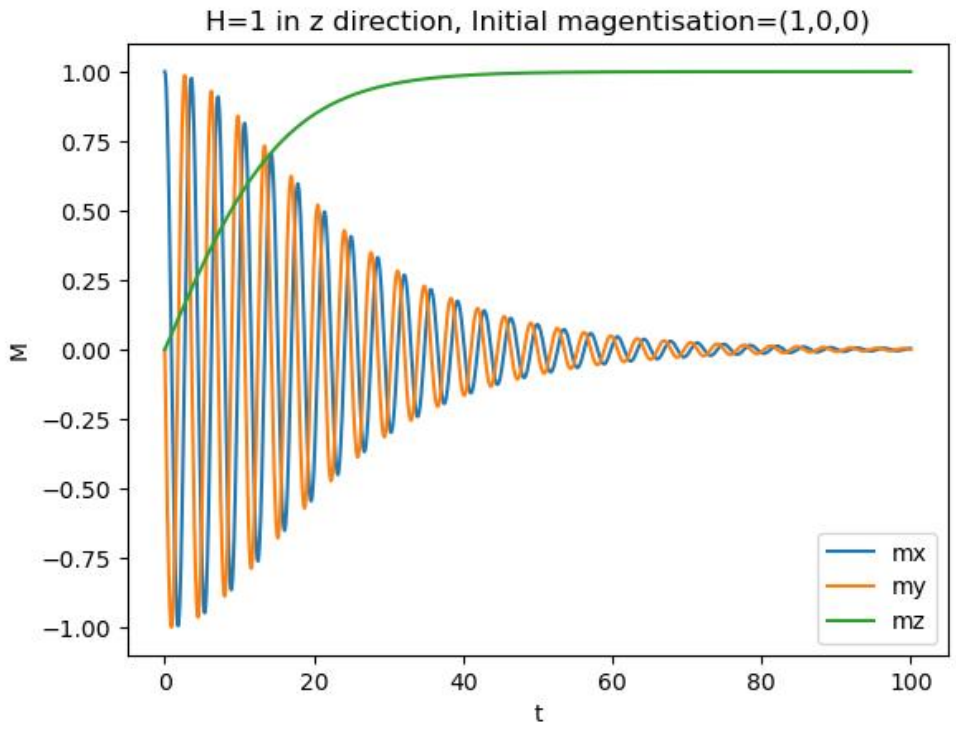
$$\frac{\partial m}{\partial t} = -\gamma m \times H_{eff} - \alpha m \times \left(\gamma m \times H_{eff}\right)$$

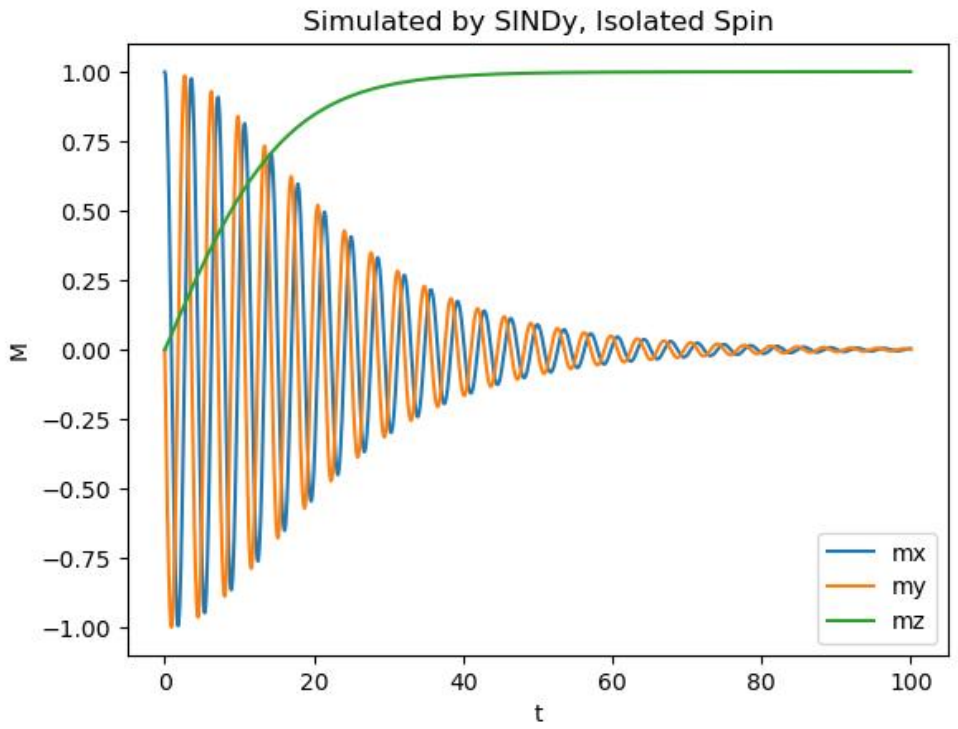
- Where H_{eff} can be determined by the demagnetization tensor (N) as follows:

$$H_{eff} = H_{ext} - N.M$$

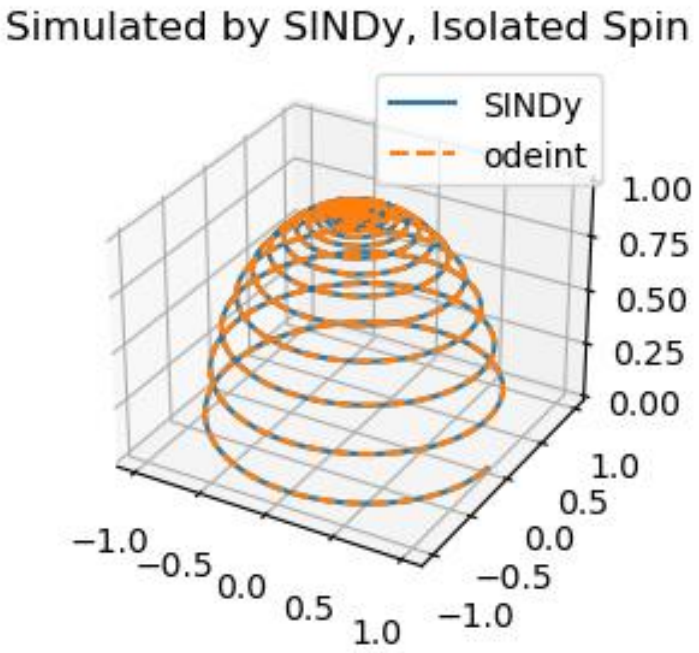
The demagnetisation tensor depends on the geometry of the magnetic body

Simulations- No Demagnetizaton Field





$\alpha = 0.02$, $N = (0, 0, 0)$

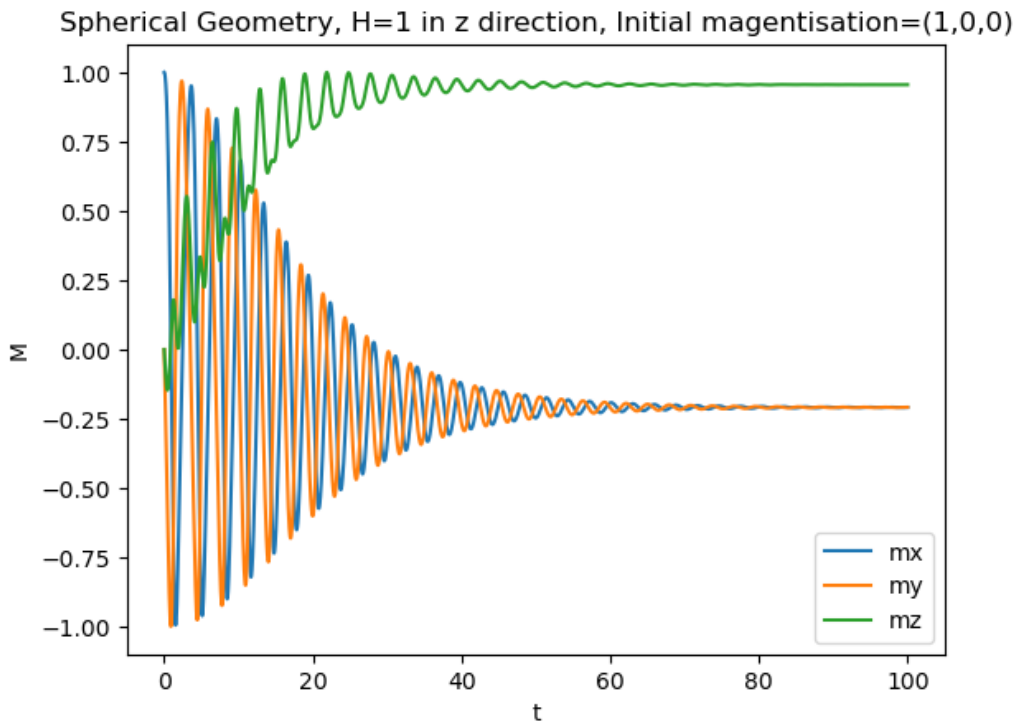


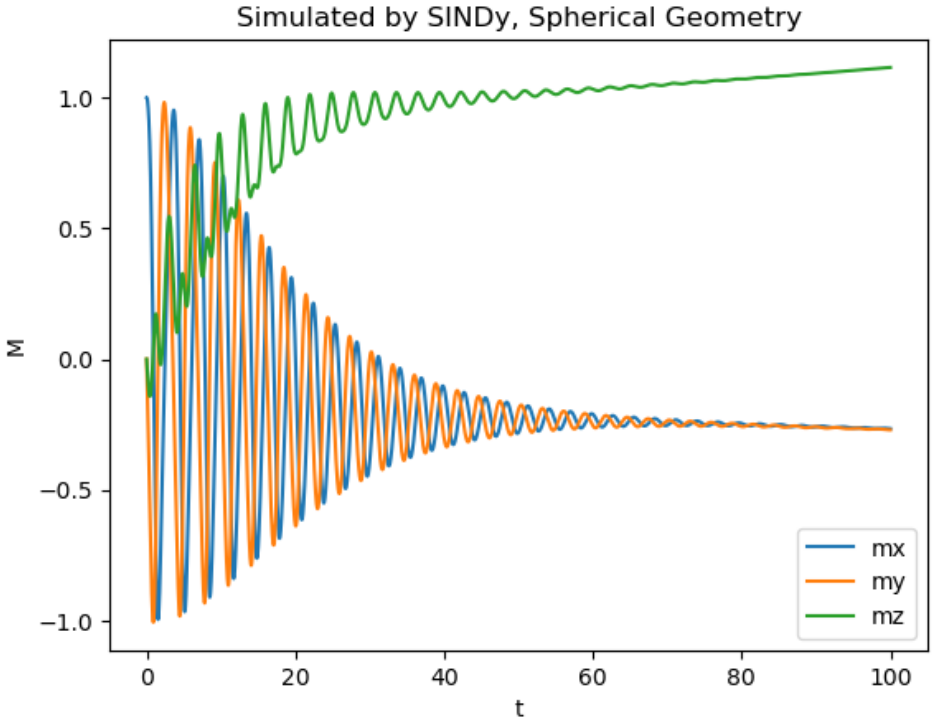

```
model.print()
#Expectation:

# mx'= 1.76 (my - 0.0352 mx mz),
# my'= 1.76 (-mx - 0.0352 my mz),
# mz'=-0.0352 (-1.76 mx^2 - 1.76 my^2)
```

```
(mx)' = 1.760 my + -0.062 mxmz
(my)' = -1.760 mx + -0.062 mymz
(mz)' = 0.062 mxmx + 0.062 mymy
```

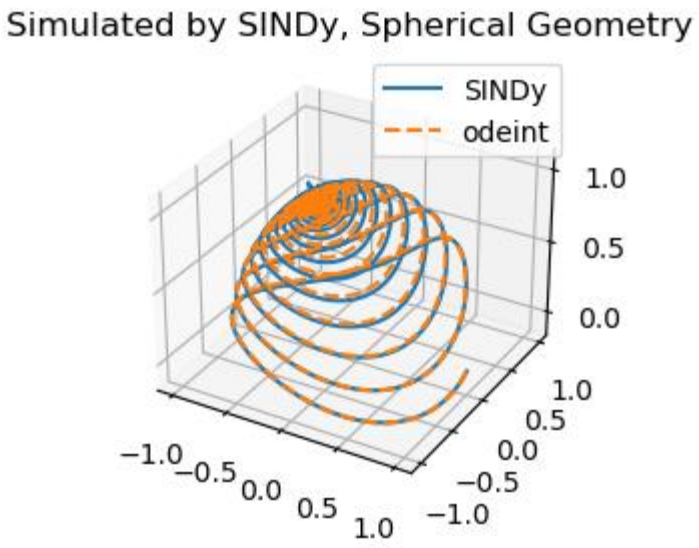
Erms = 0.00031098

Spherical Geometry





$\alpha = 0.02$, $N = (1/3, 1/3, 1/3)$

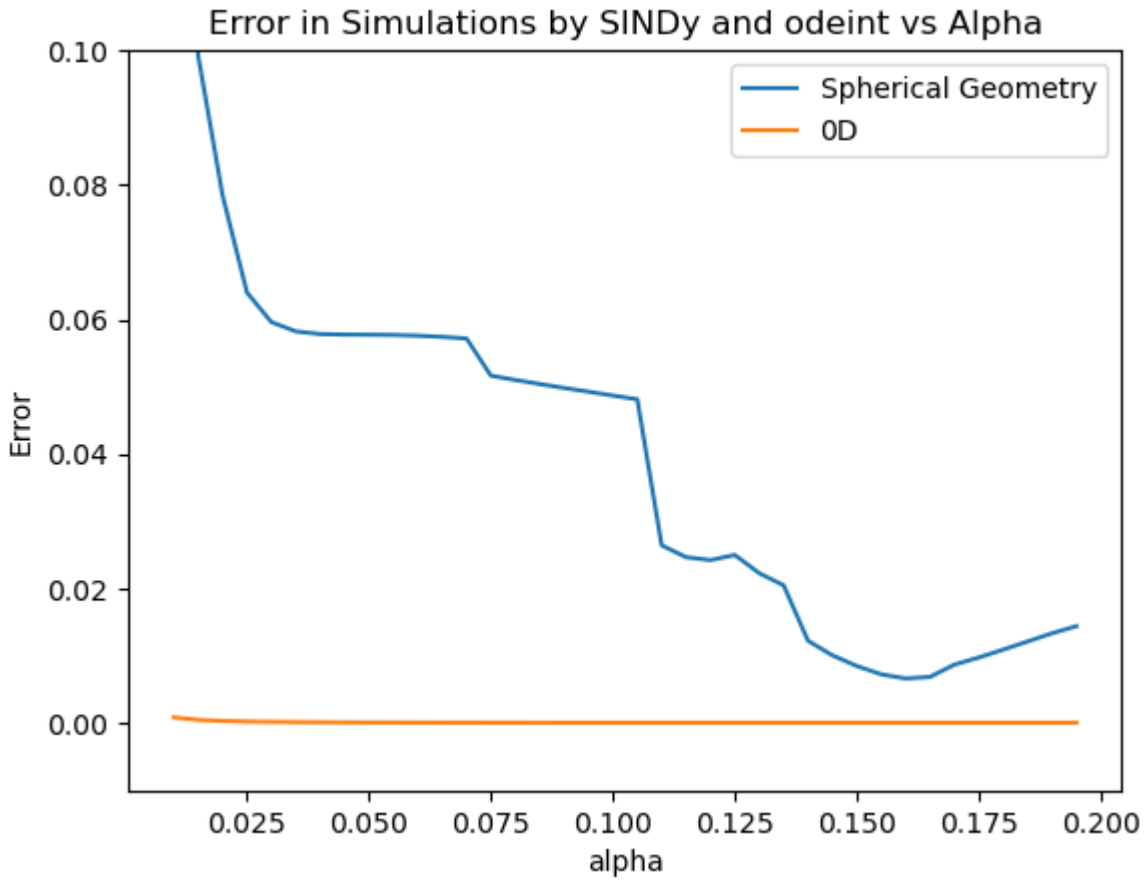


Sindy Model Output:

```
(mx)' = 1.746 my + -0.575 mx my + 0.520 mx mz + -0.589 my^2 + 0.568 mz^2
(my)' = -1.773 mx + 0.585 mx^2 + 0.597 mx my + -0.653 my mz + -0.605 mz^2
(mz)' = -0.524 mx^2 + -0.577 mx mz + 0.651 my^2 + 0.596 my mz
```

Erms = 0.07843

Error as a function of Damping



More Complicated Geometries

- We have encountered relatively simple geometries until now, with demagnetization tensors N = (0, 0, 0) and N = (1/3 , 1/3 , 1/3). Now, let us consider a point inside a flat rectangular prism, where the demagnetization tensor need not be diagonal
- The symmetric demagnetisation tensor N at (x,y,z) inside a flat rectangular prism of dimensions (a,b,c) has the components:

$$N_{ii}(\mathbf{r}) = \frac{1}{4\pi}[\arctan f_i(x,y,z) + \arctan f_i(-x,y,z) + \arctan f_i(x,-y,z) + \arctan f_i(x,y,-z) + \arctan f_i(-x,-y,z) + \arctan f_i(x,-y,-z) + \arctan f_i(-x,y,-z) + \arctan f_i(-x,-y,-z)]$$

$$N_{ij}(\mathbf{r}) = -\frac{1}{4\pi}\ln\left[\frac{F_{ij}(\mathbf{r},a,b,c)F_{ij}(\mathbf{r},-a,-b,c)F_{ij}(\mathbf{r},a,-b,-c)F_{ij}(\mathbf{r},-a,b,-c)}{F_{ij}(\mathbf{r},a,-b,c)F_{ij}(\mathbf{r},-a,b,c)F_{ij}(\mathbf{r},a,b,-c)F_{ij}(\mathbf{r},-a,-b,-c)}\right], \quad i \neq j$$

where,

$$f_x(x,y,z) = \frac{(b-y)(c-z)}{(a-x)[(a-x)^2+(b-y)^2+(c-z)^2]^{\frac{1}{2}}} \quad (6)$$

$$f_y(x,y,z) = \frac{(b-y)(c-z)}{(a-x)[(a-x)^2+(b-y)^2+(c-z)^2]^{\frac{1}{2}}} \quad (7)$$

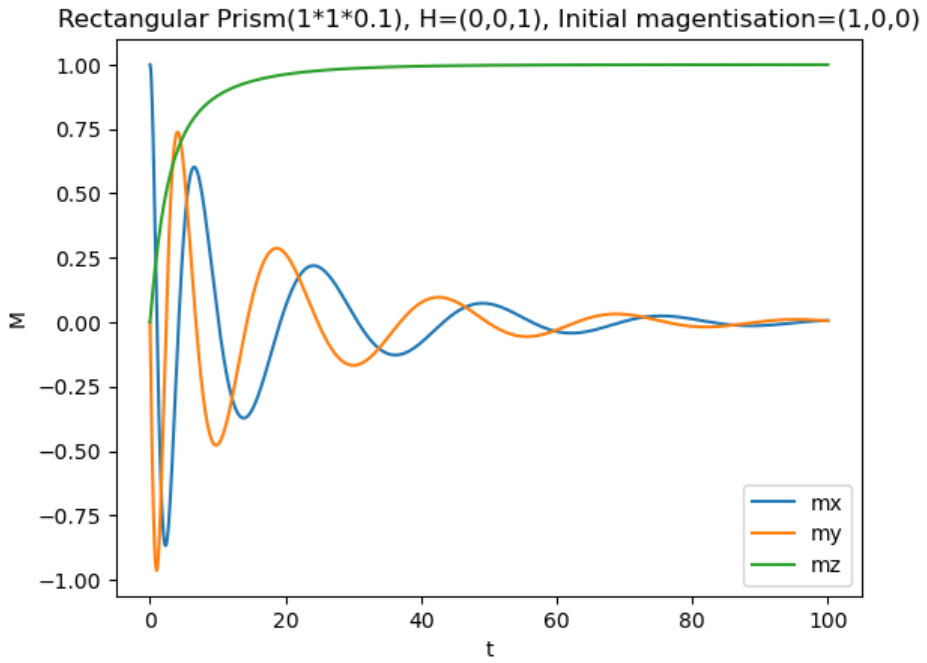
$$f_z(x,y,z) = \frac{(c-z)(a-x)}{(a-x)[(a-x)^2+(b-y)^2+(c-z)^2]^{\frac{1}{2}}} \quad (8)$$

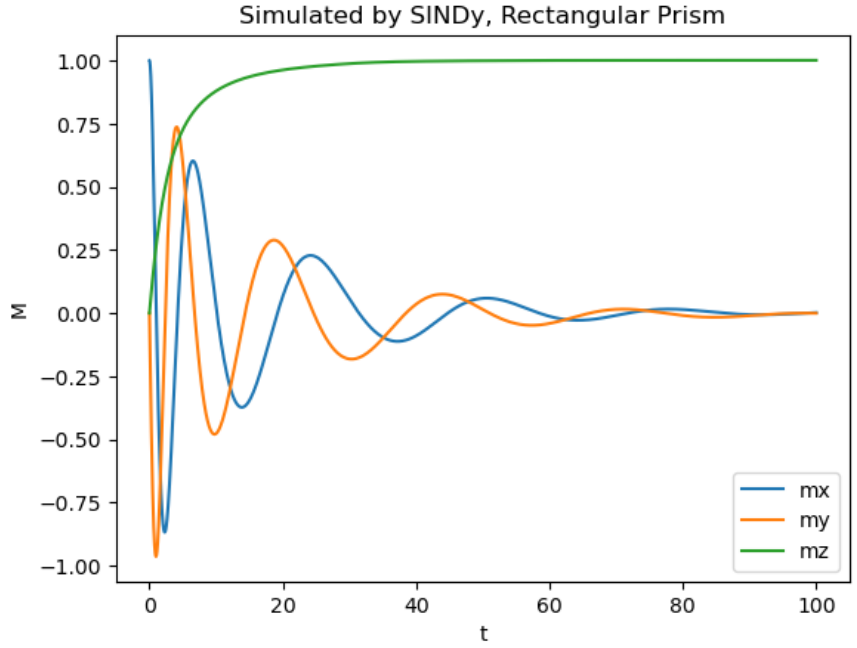
Where,

$$F_{xy}(r,a,b,c) = (c-z) + [(a-x)^2+(b-y)^2+(c-z)^2]^{1/2} \quad (9)$$

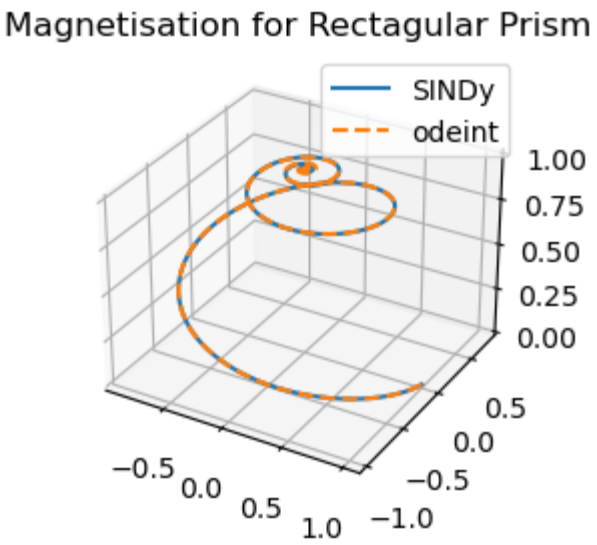
$$F_{yz}(r,a,b,c) = (a-x) + [(a-x)^2+(b-y)^2+(c-z)^2]^{1/2} \quad (10)$$

$$F_{zx}(r,a,b,c) = (b-y) + [(a-x)^2+(b-y)^2+(c-z)^2]^{1/2} \quad (11)$$


Rectangular Prism(1*1*0.1), H=(0,0,1), Initial magentisation=(1,0,0)
M
t
mx
my
mz


Simulated by SINDy, Rectangular Prism
M
t
mx
my
mz

$\alpha = 0.1$

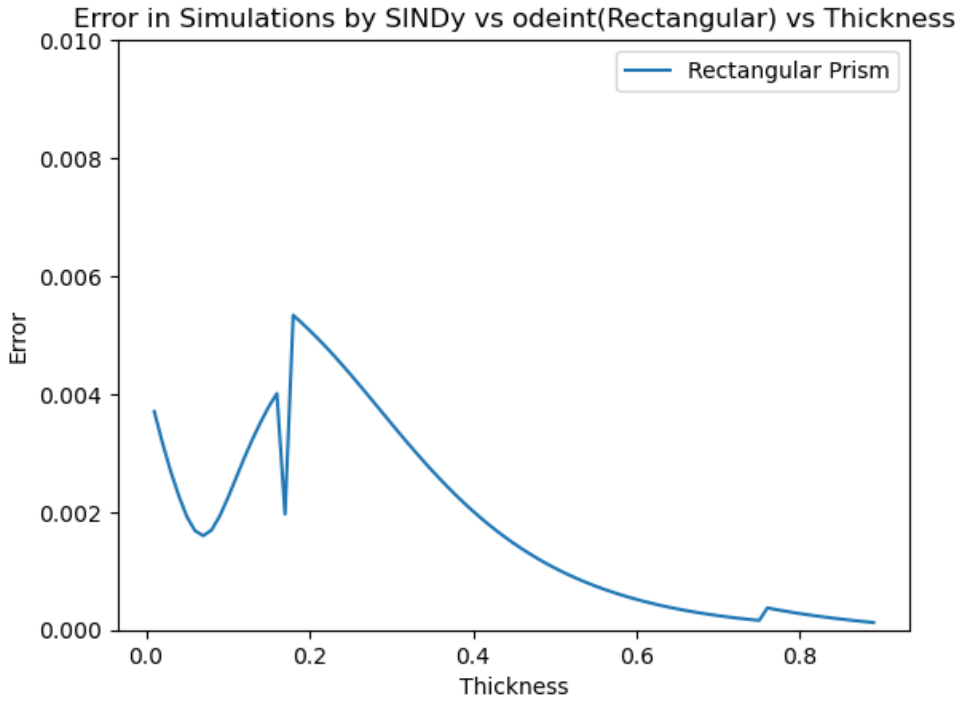


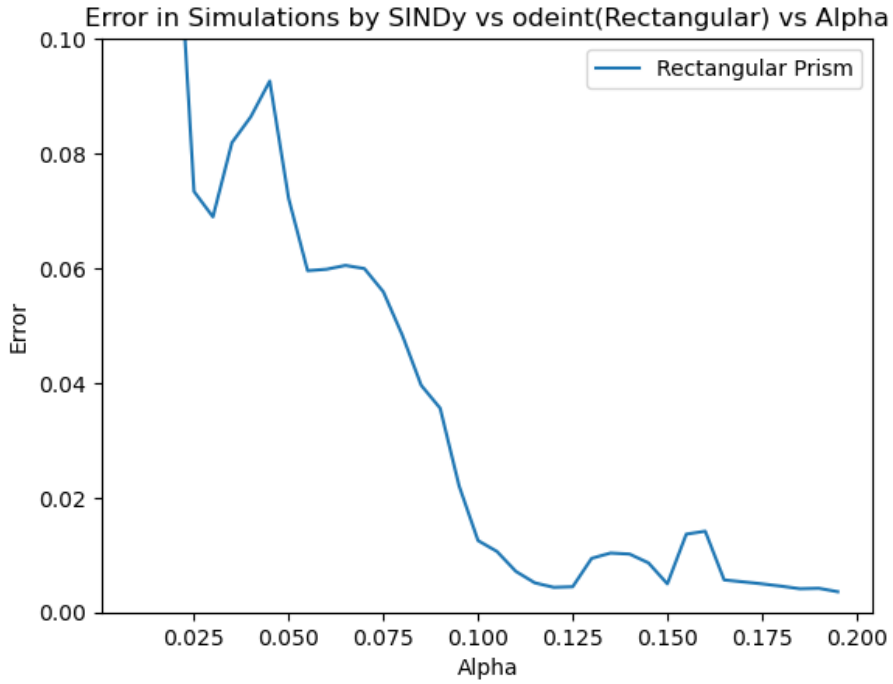
SINDy Model:

```
(mx)' = -0.154 mx + 1.794 my + -1.176 mz + 0.160 mxmx + 0.152 mymy + 1.176 mzmz + -0.042 mxmy + 0.108 mxmz + -1.560 mymz + 0.05
2 mxmymz + 0.437 mxmxmz + 0.457 mymymz
(my)' = -1.771 mx + -0.134 my + -0.109 mz + -0.064 mymy + 0.109 mzmz + 0.041 mxmy + 1.535 mxmz + 0.083 mymz + -0.067 mxmymz +
0.015 mxmxmy + 0.061 mxmxmz + 0.140 mymymz
(mz)' = 0.310 mxmx + 0.310 mymy + -0.268 mxmxmz + -0.268 mymymz
```

Erms = 0.0125

Error as a function of Thickness and Alpha





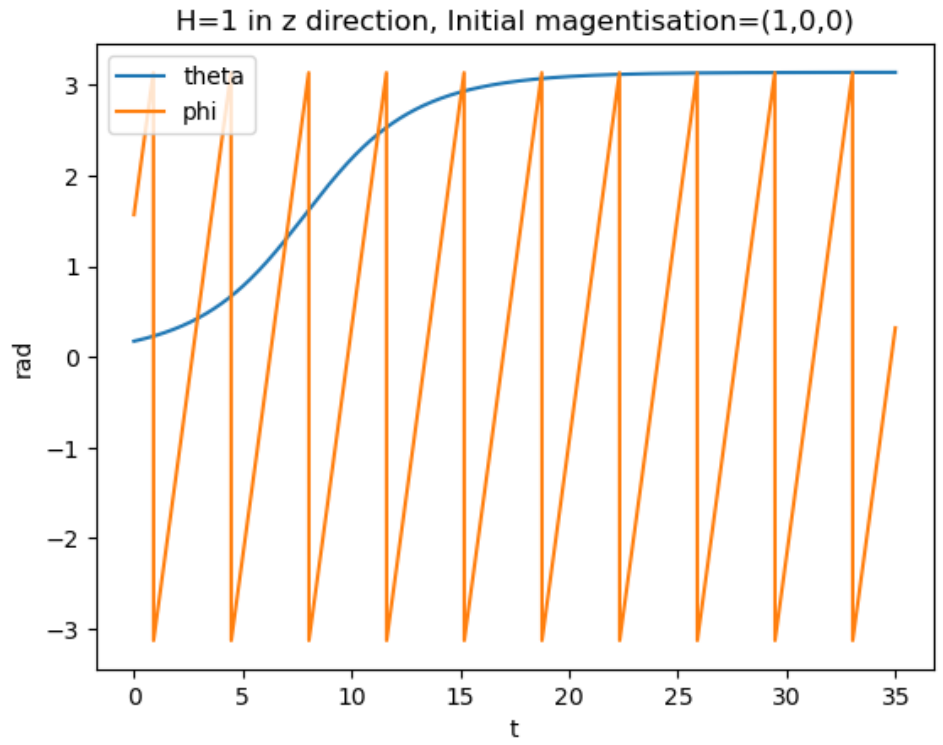
$\alpha = 0.02$

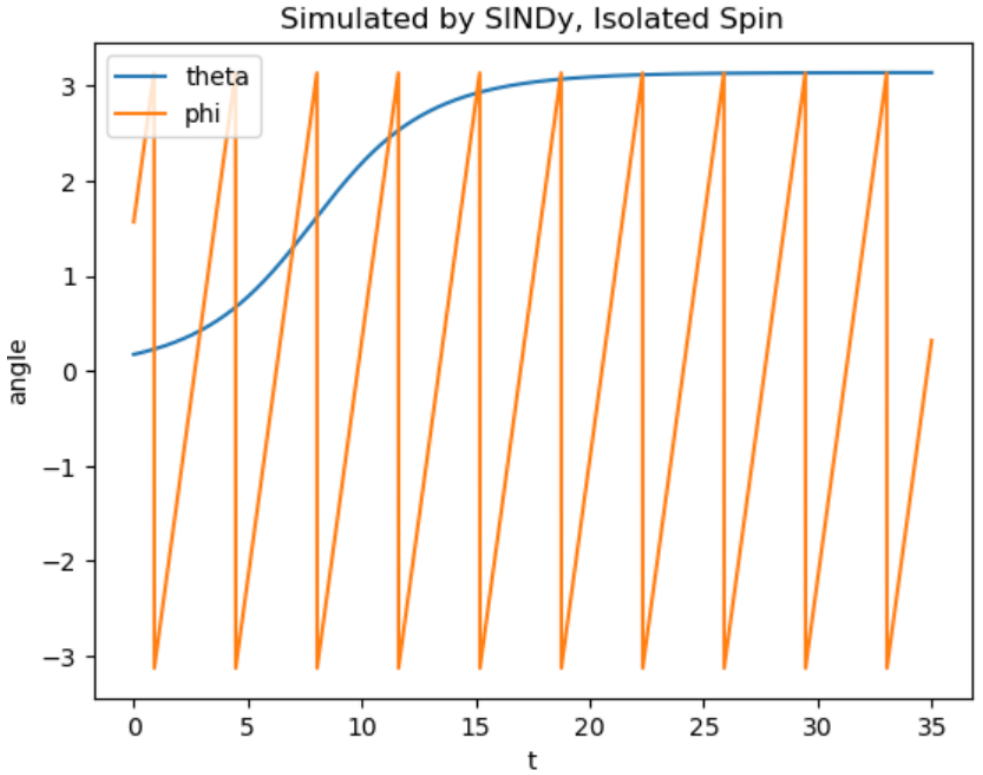
Reducing Dimensions

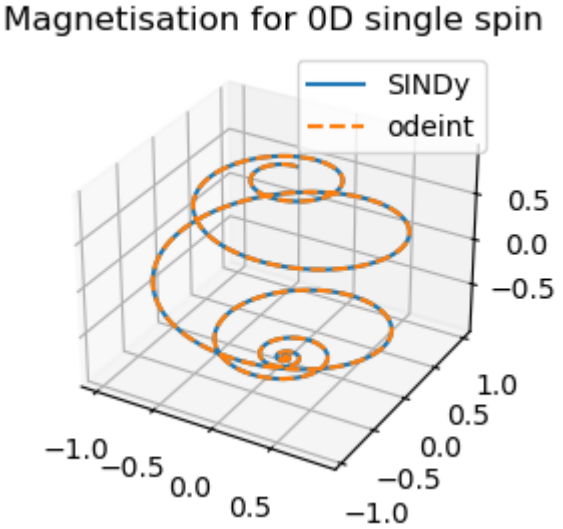
- The reduction of dimensions in the solutions to the LLG comes out naturally, without the use of techniques such as POD or PCA
- This is due to the fact that the magnetization is normalized, and the state can be defined as a function of just (θ,φ)
- The LLG equation in this case can be represented as functions of θ and φ as:

$$\dot{\theta} = \frac{d\theta}{dt} = \frac{\gamma\alpha}{\alpha^2+1}(H\sin\theta - H_k\sin\theta\cos\theta) \qquad \frac{-d\varphi}{dt} = \left(\frac{\gamma}{\alpha^2+1}\right)(H - H_A\cos\theta)$$

Here, I changed to the appropriate basis functions in SINDy by defining a custom function Library. Also, the derivates were fed directly to SINDy to compensate for the discontinuities in the angles at 0 and 2π



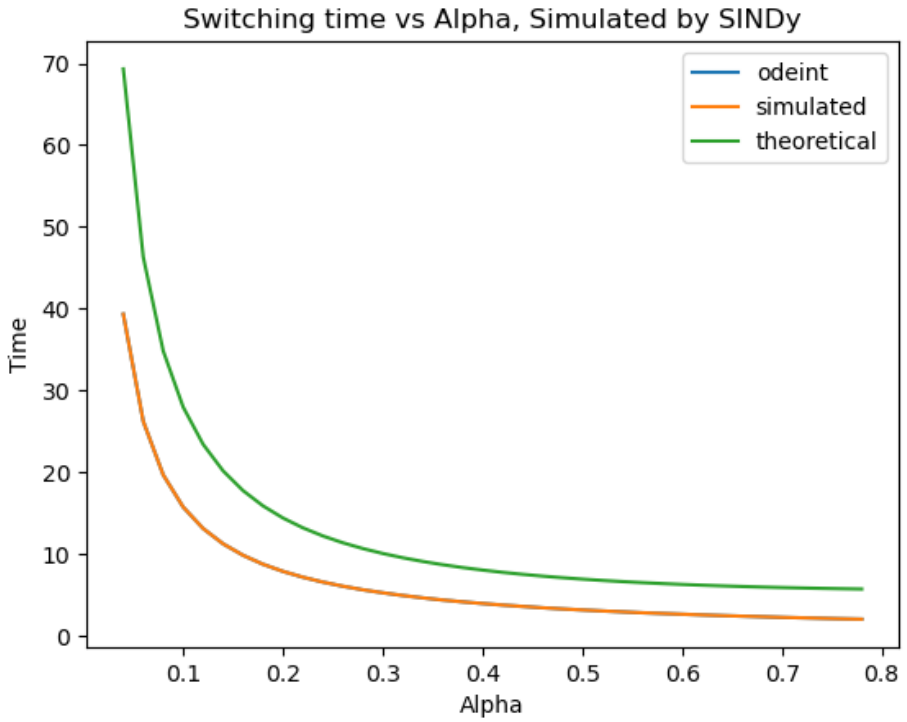




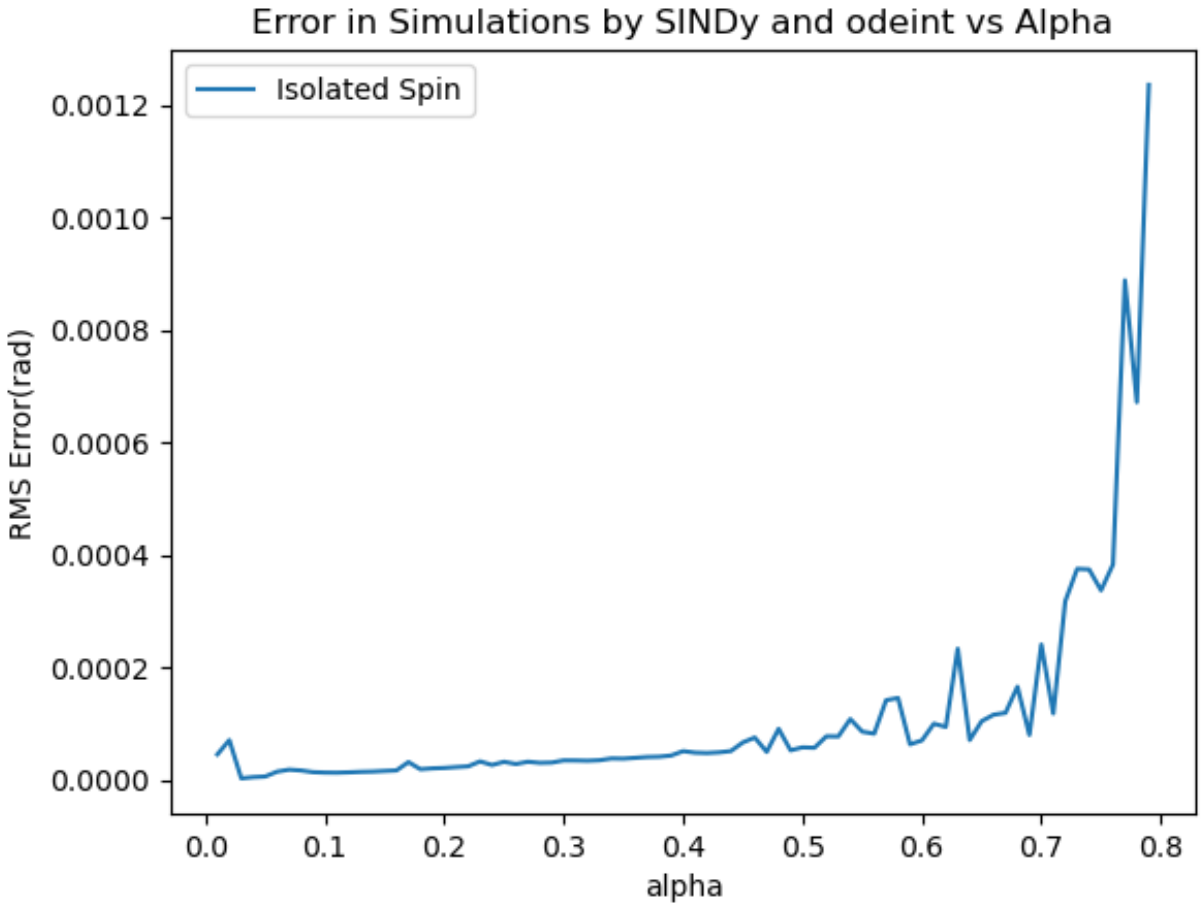
α = 0.1, θi=10°
off the z-axis,
H=(0,0,-1)

Switching Time

- We define switching time as the time it takes for the spin to go from 10° to 170° after turning on the field as H = (0,0,-1).
- Let us look at the switching times as a function of α, simulated by odeint, SINDy and compare it to the theoretical results.



Errors vs Damping

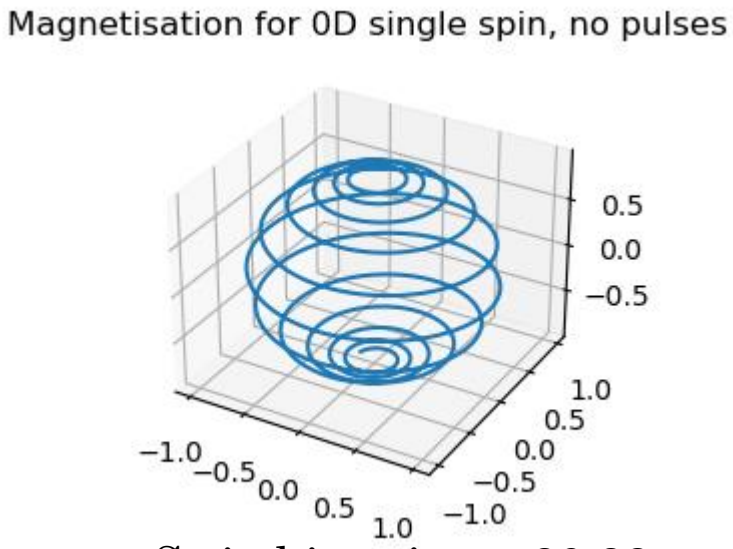


We can see that the errors are lower across a higher range of alphas when compared to simulations in 3d

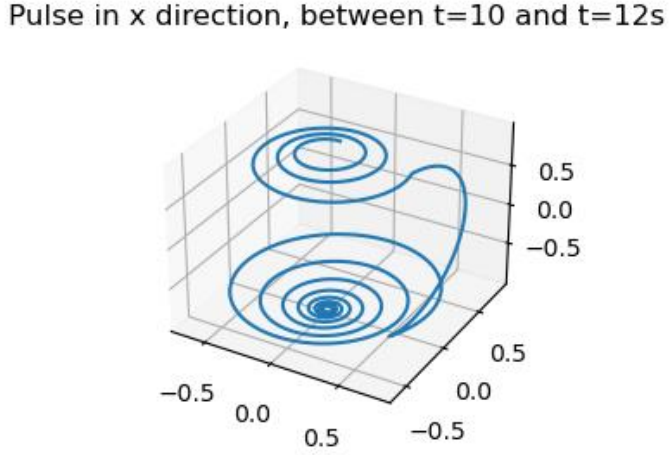
Pulse Programming

Pulse Programming

- We have demonstrated the robustness of SINDy for identifying the dynamics of a non-linear system such as the LLG equation.
- We now look to possible avenues where this might be of use. One such avenue is pulse programming.
- Given that we can control pulses in the z and now the x-direction, our aim is to identify the optimal pulse lengths, timings and amplitudes to manipulate the spin so that it reaches the desired state in the least amount of time.
- More specifically, we can start with finding the optimal pulses to reduce the switching time.



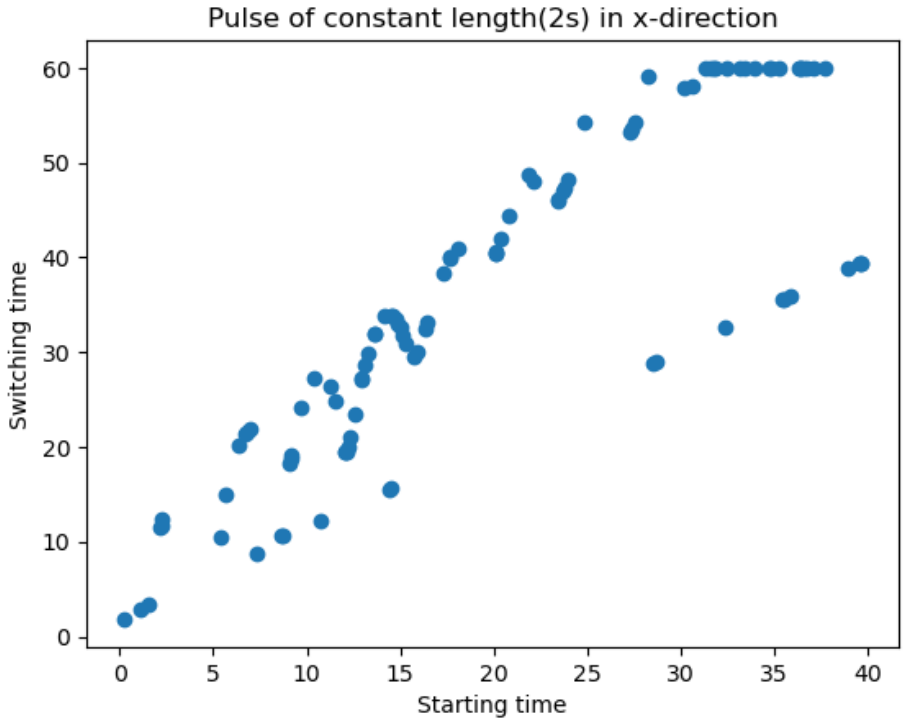
Switching time =39.33s

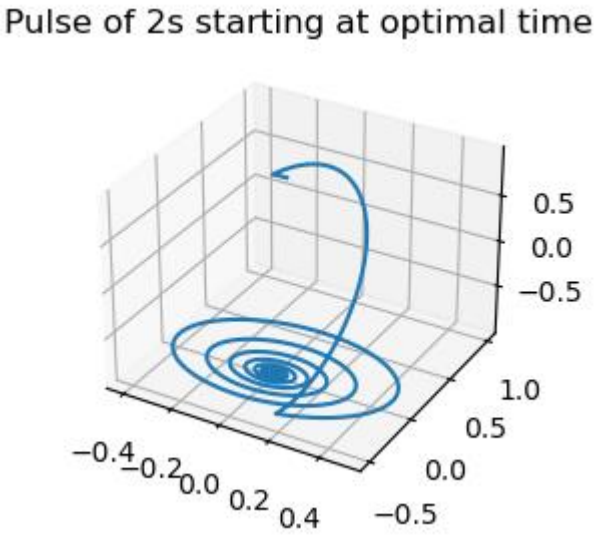


Switching time =25.81s

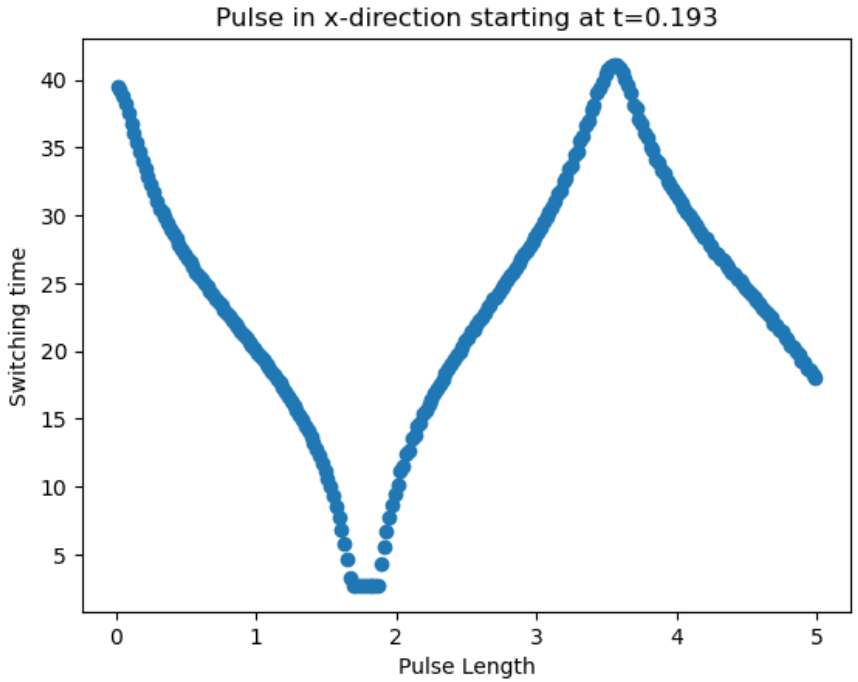
As a first step, we try to optimize the parameters separately.

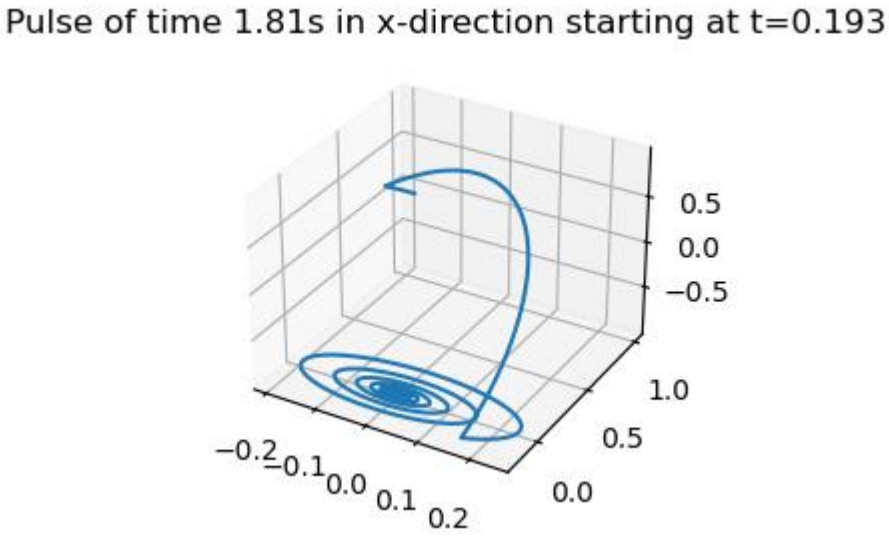
1. Optimizing time to start pulse, keeping pulse length constant





2. Optimizing pulse length





If the spin goes to 170° and comes back, before settling down, this behaviour is undesirable, we can account for this by redefining the switching time function, by requiring that switching is done when 25 consecutive values are at >170°.

Next Steps: We have seen how we can optimize the pulses by varying a single parameter at a time, additionally, we need to see how to optimize them all together. This is a very computationally challenging task, and we might explore quantum algorithms in the future to do this task efficiently

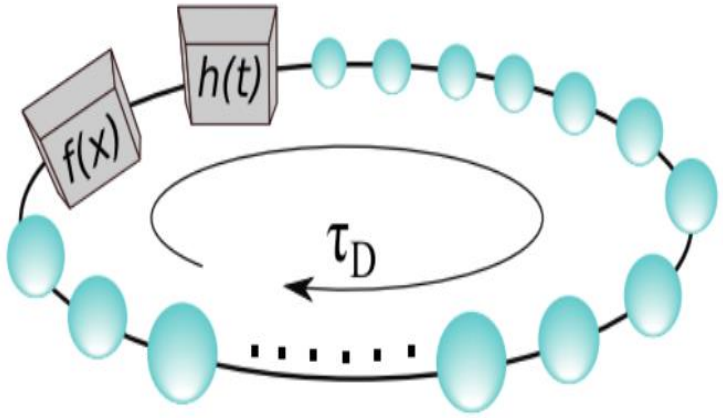
Simulating Photonic Neural Networks in Delay Systems

Summer Project

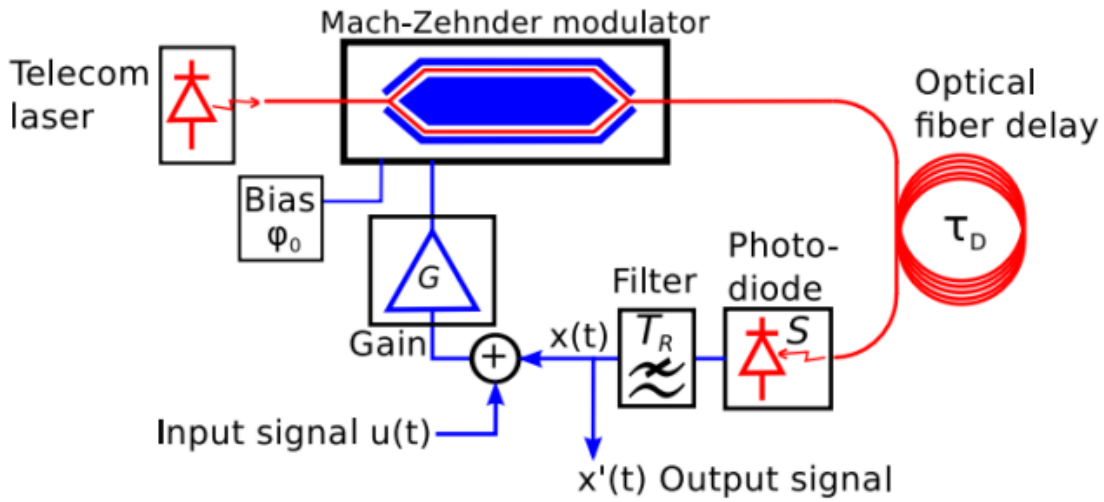
Delay Systems

- Photonic systems use optical delays to represent ANNs in a purely temporal domain, instead of the spatio-temporal systems are now widely used.
- Delay systems are described by the characteristic equation:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{x}(t - \tau_D))$$



The neurons are assigned temporal positions with a constant separation δt, and the ANN state is represented temporally with each time step(1...T) having time length τ_D
We can determine the evolution of the state in time by numerically solving the resulting Delay Differential Equation(DDE), after modulating the inputs and outputs.

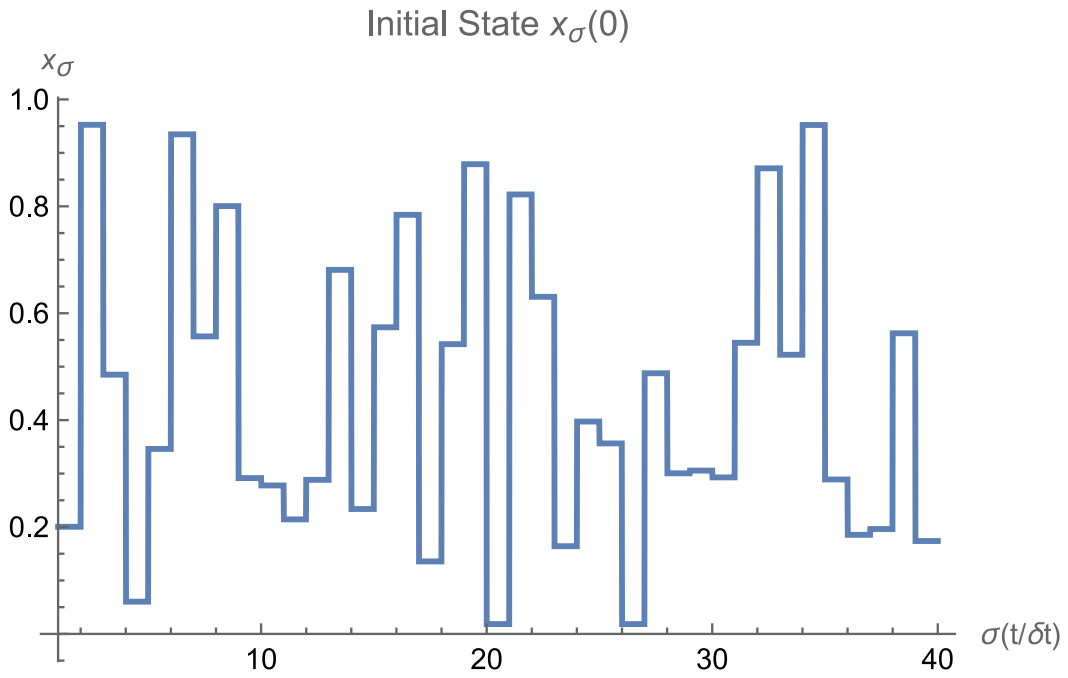


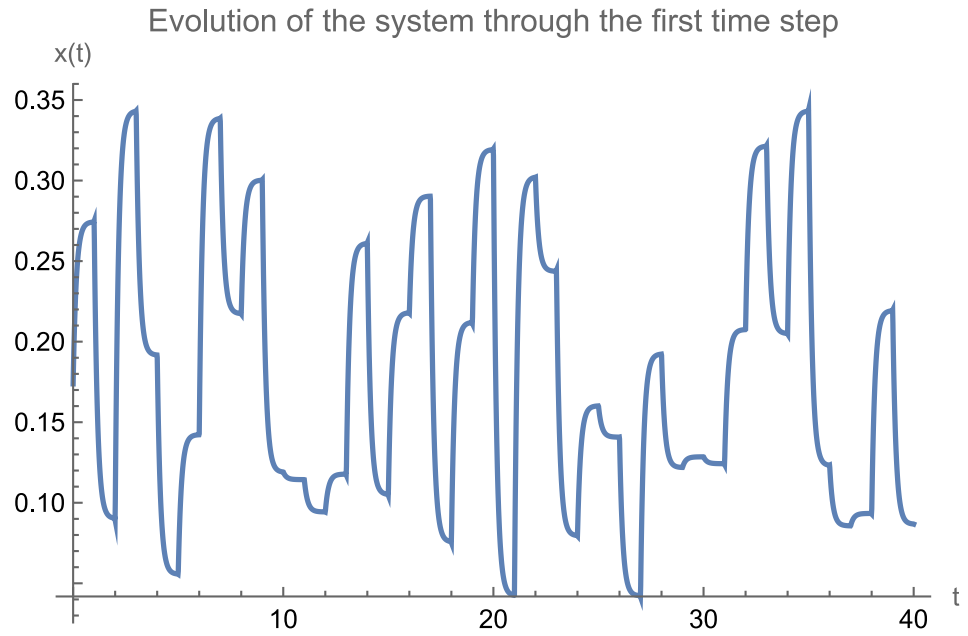
The MZM has a sin^2 non-linearity, and for the here employed low-pass system, the resulting DDE is:

$$\varepsilon \dot{x}(s) + x(s) = \beta \sin^2[\mu x(s-1) + \rho u(s) + \Phi_0]$$

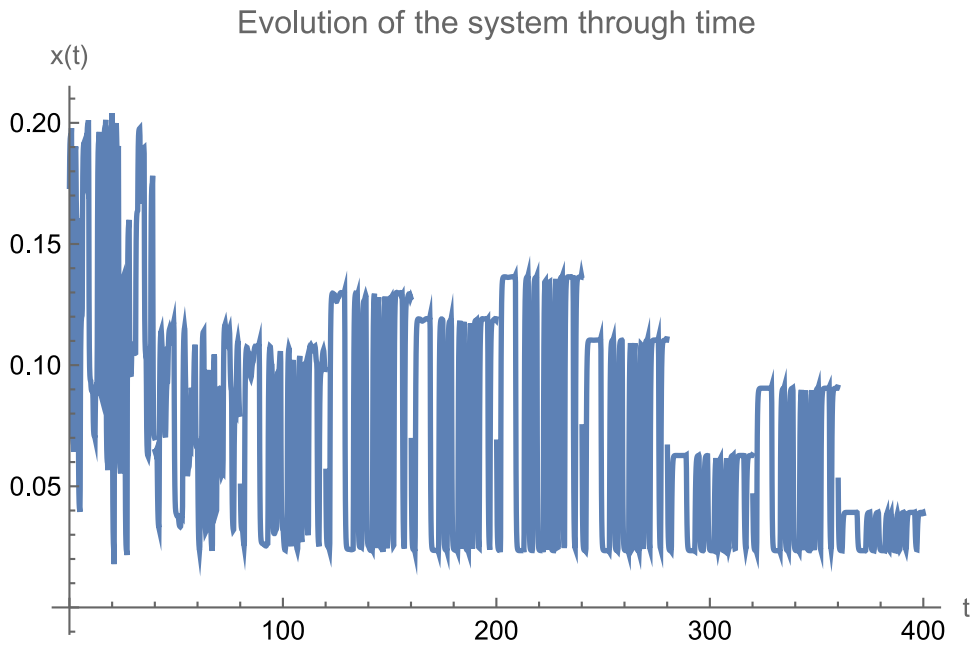
Methodology

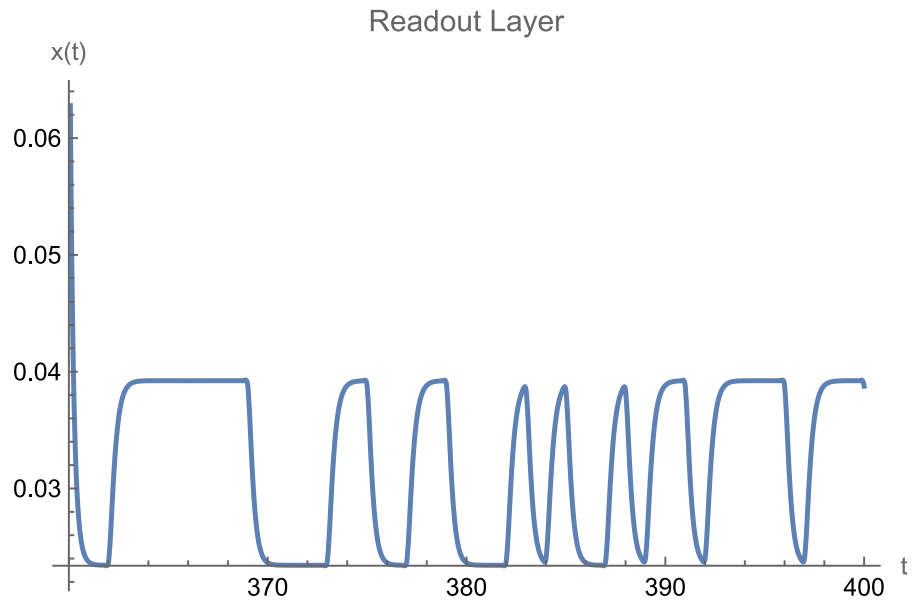
The weights of the neurons (the state x(t)) are initially randomized, and the DDE is solved using Runge-Kutta of the 4^{th} order, using the state at a previous time as one of the inputs





Next, we use evolve the system through T time steps to obtain the state of the system at time T





Now, the output weights matrix Wout can be obtained from the following ridge regression:

$$\mathbf{W}^{out} = (\mathbf{M}_x \cdot \mathbf{M}_x^{\mathrm{T}} + \lambda \cdot \mathbf{I})^{-1}(\mathbf{M}_x \cdot \mathbf{T}^{\mathrm{T}})$$

where T is the teacher matrix and the test output on
input can be determined by:

$$\mathbf{y}(n) = W^{out}\mathbf{x}(n)$$

I have used all these steps to implement a rudimentary network to solve XOR operations. This was just a demonstration on the basic working principle and the sanity of the code. Further optimizations in the many hyperparameters need to be made for better results.

The network was trained on 40 training data consisting of 4 bits, where the teacher matrix consisted of the XOR of these 4 bits. After the training, the network was tested on 10 data points, and the correlation between the target vector and the output of the network came out to 0.772741