EFFECT OF NOISE IN ACTIVE PARTICLE RESERVOIR COMPUTING

Bachelor's thesis by
Shashank Shetty Kalavara

Under the supervision of
Dr. Xiangzun Wang & Prof. Dr. Frank Cichos.

UNIVERSITÄT LEIPZIG

Leipzig, Saxony

November 23rd, 2022

Second examiner Prof. Dr. Bernd Rosenow.
Submitted in partial fulfillment of the requirements for the degree of
B.Sc., International Physics Studies Program

Acknowledgement

I would like to thank Dr. Xiangzun Wang and Prof. Dr. Frank Cichos for their supervision and patience, and Prof. Dr. Bernd Rosenow for graciously accepting to be the second examiner for my thesis. Additionally, I would like to thank Thomas de Paula Barbosa, Daniel Stephen Radyuk, and Eduardo Mayorga for the proofreading. And lastly, I am grateful to my family for their support, Tomasz Niewiadomski for the collaboration at the beginning of the project, Sächsisches Staatsministerium für Kultus for keeping tuition free in Leipzig, and Lieferando.de for keeping me employed.

Contents

1 Introduction

This project attempts to study and simulate a noisy reservoir, for the purpose of time-series predictions in Reservoir-Computing. Reservoir computing is a simple form of Recurrent Neural Network, which can produce good-quality time series prediction. This simplicity of the Reservoir-Computing networks allows great flexibility in probing and understanding the inner working of a Neural network and its ability to make successful predictions. The reservoir is constructed from a dynamical system of active particles, which follow a time-delay-induced oscillation around a fixed particle. The reservoir simulation is based on a physical active particle system and as such the simulation follows constraints imposed by it. One of these major constraints is the effect of Brownian-Noise on the active particle oscillations. Effects of which are significant on the ability of the network in producing good time-series predictions.

The chapter on *Theoretical Background* introduces the theory and formulations of the time-delayed active particle oscillations, Brownian noise and its effect on the time-delayed oscillations, and the regular Echo state networks akin to Reservoir-Computing.

The chapter on *Experimental analogous computational methods* is a computational substitution to the customary experimental methods chapter since computational techniques and simulations form the bulk of this thesis. This chapter builds on the theoretical formulations introduced earlier to describe the computational methods forged in this project. The chapter introduces the working of the reservoir, reservoir activation state collection matrix, Noisy reservoir & its activation states, and finally the functional description of the important code in the project.

The final chapter on *Results and comparative analysis*, shows and interprets the obtained results and summarises a conclusion. This chapter tries to compare and contrast various time series signals with a noisy and noiseless reservoir. Additionally, the Mackey-Glass time series signal is extensively used to further probe the noisy reservoir through the network predictions.

2 Theoretical background

2.1 Time-delayed Oscillations of Active particles

2.1.1 Active particle oscillations

The basis of this project, the reservoir, is built on the dynamics of the active particle oscillations. This section explores the physics and dynamics of this system. Active particles are defined as particles that consume energy to propel persistently or to exert mechanical forces. [Kokot et al. (2022)]. The particular system of interest here is that of an active-particle oscillating around a fixed particle, as can be seen in Figure 1.

In this system, the oscillating active particle has a peculiar property of delayed information reception from the fixed source particle which causes a delayed change in the trajectory of the attracted active particle. This delayed attraction from the fixed particle causes an angular displacement, resulting in a rotation of the active particle. The rotational trajectory again experiences a delayed attraction, and so the cycle keeps inducing a continuous oscillation. Hence the name 'Time delayed active particle oscillation'.

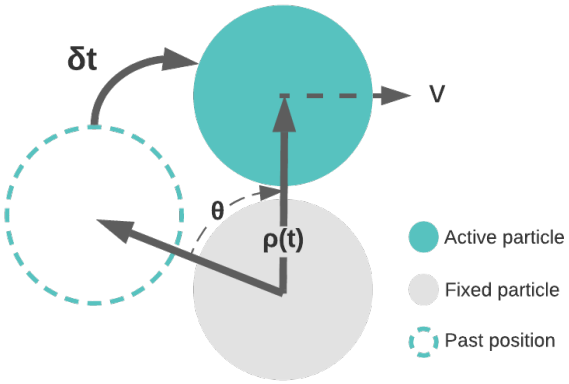

Figure 1: Active particle displacement due to time-delayed attraction

The equation for the system can be derived as follows. The variable θ is the angle of propulsion and can be written as follows [Wang et al. (2022)],

$$\theta = \int_{t-\delta t'}^{t} \omega\left(t'\right) dt$$

Where angular velocity $\omega = \frac{v_0}{\rho} \cdot sin\left(\theta\right)$, additionally the stable angular velocity is written as $\omega_0 = \frac{v_0}{\rho_0}$, v_0 and ρ_0 are the initial velocity and radius. The equation for the system is further expanded with the time-dependent angle ϕ as later described in Equation 3, where t is the time variable. $d\phi(t)$ is the time-dependent short angle of rotation resultant of '$d\omega$' at every discrete time interval update of 'dt' such that,

$$d\phi = dt \cdot \omega\left(t\right)$$

This can be further expanded with the relation for angular velocity as,

$$d\phi = dt \cdot \omega(t) \cdot sin\left(\theta\left(t\right)\right) \tag{1}$$

Here $\theta(t)$ is simply the angle formed due to the time delay at a time 't', as can be seen in Figure 1. Time-delayed angle θ is described more concisely with the Equation 2 [Wang et al. (2022)], as simply the difference between the current angle at 't' and the past angle at a delayed time 't - δt'.

$$\theta\left(t\right) = \phi\left(t\right) - \phi\left(t - \delta\, t\right) \tag{2}$$

With the obtained system of equations, it is apparent that the angular evolution of ϕ in time is dependent on its own values from the past, and hence the dynamics of the system could be written in terms of a *Non-linear Delay Differential Equation* (DDEs) [Terpstra (2016)] as below,

$$\phi'\left(t\right) = \omega\left(t\right) \cdot\ sin\left[\phi\left(t\right) - \phi\left(t - \delta\ t\right)\right]$$

The angular evolution, however, needs to be derived in terms of discrete time updates for the purposes of simulation and for its further use in reservoir computing. This is done in Equation 3 using Equation 1 from above.

$$\phi\left(t + dt\right) = \phi\left(t\right) + dt \cdot \frac{v_0}{\rho\left(t\right)} \cdot sin\left(\theta\left(t\right)\right) \tag{3}$$

The variable ρ, introduced in Equation 3 is a time-dependent radius or the distance between the centre of the fixed and active particle. This radial dependence of time makes the angular velocity a function of time $\omega\left(t\right) = \frac{v}{\rho(t)}$ where v is the initial velocity.

The equation for the radius ρ has to be derived from the symmetries in the above oscillation model. Since the radius is a perpendicular component relative to the rotational tangent velocity, the radius update is written as

$$d\rho = -dt \cdot v \cdot cos\left(\theta\right)$$

Hence the final discrete time update equation for radius ρ can be written as in Equation 4 below.

$$\rho\ \left(t + dt\right) = \rho\ \left(t\right) - dt \cdot\ v \cdot\ cos\left(\theta\ \right) \tag{4}$$

The equations derived to model the time-delayed active particle oscillations are numerically simulated using the discrete-time update Equations 2, 3 and 4 in order to gain a better understanding of the nonlinear dynamics involved.

The following Figure 2 is a plot of the evolution of Equations 2, 3, and 4 in time, given a tiny perturbation $\phi$to the system. Figure 2 shows some insights into the previously described equations. In particular, the evolution of θ and ρ show a decay in form since the simulation only has a perturbation at the beginning of the total simulation time. The evolution of ϕ is rather different from the others. The function keeps accumulating over time as is also evident from Equation 3, with a tiny fluctuation at the beginning caused due to the induced perturbation.

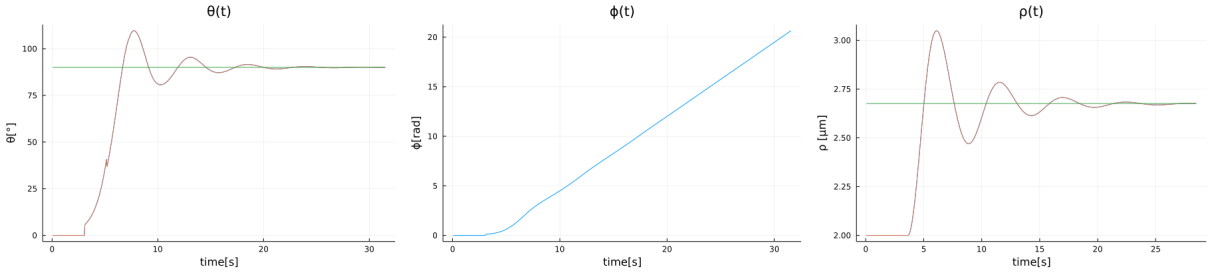

Figure 2: Response of θ, ϕ & ρ to a small perturbation

One of the important interactions for Reservoir computing is the interaction of time series signal input with the physical time-delayed active particle oscillation system. Due to the cyclical dependence of Equations 2, 3 and 4, a time series input can be induced in the dynamics with a simple modification to Equation 3.

$$\phi\ \ (t+dt) = \phi\ (t) + dt \cdot\ \omega\ (t) \cdot\ sin\,(\theta\ (t) + \mu\,(t)) \tag{5}$$

Equation 5 shows the induction of time series input $\mu(t)$ into active particle oscillation, through modification in Equation 3. The active particle system receives input through a laser external to the system, which can be mathematically interpreted as an angular addition to the time-delayed oscillator angle θ. This is more clearly demonstrated in Figure 3. The $\mu(n)$ is an input induced to θ a time-delayed part of the ongoing time-delayed oscillation, hence the modification from the $sin(\theta)$ to $sin(\theta + \mu)$. This $\mu(n)$ then makes changes and evolves through the system of equations described earlier in this section.

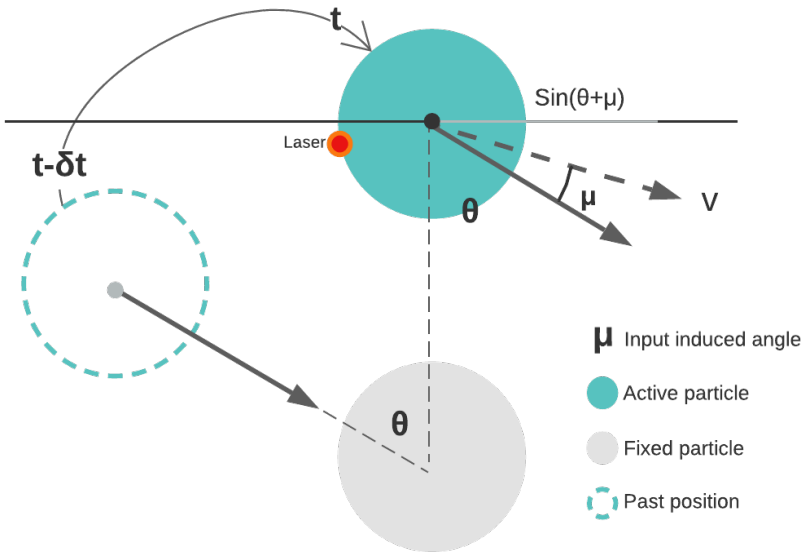

Figure 3: Adding Input $\mu(n)$ to a Time-delayed attraction

Plots in Figure 4 are the time evolving of variable θ, ϕ and ρ to a time series sinusoidal input of $A*(sin(t)+2cos(3t))/2$. These time evolution plots represent the system's response

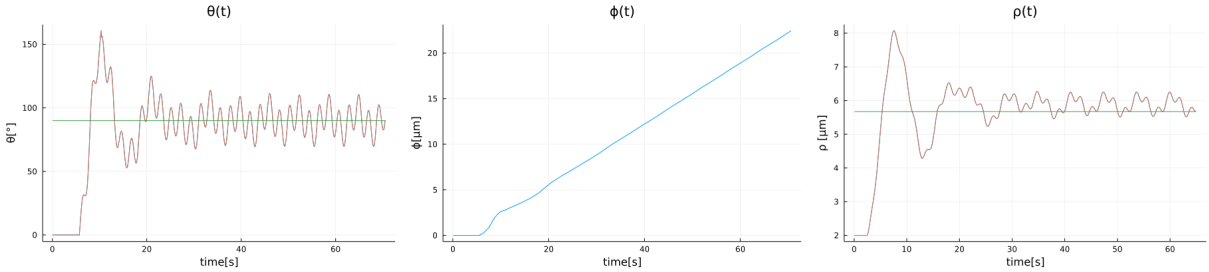

Figure 4: Response of θ, ϕ and ρ to a time series input, using equation 5

to the mentioned input. Unlike the responses seen earlier in Figure 2, the responses in Figure 4 are a consequence of the system receiving a time series input through the time rather than just a perturbation at the beginning of the simulation.

The responses of θ and ρ in Figure 4, have a response where the initial response around the 10-20 seconds mark shows a transient system response to the induced perturbation, which then transitions into a more input-driven steady-state response.

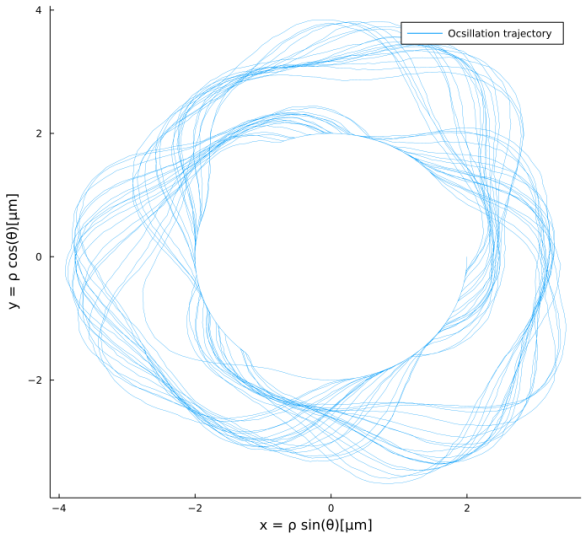

Figure 5: Example of an active particle trajectory with a sinusoidal input

Figure 5 is a graphical representation of the spatial movement of the oscillating active particle to induced input. This is simply done by transforming the Equations 2, 3 and 4 written in the polar form to a Cartesian form to obtain the plot in Figure 5. The plot shows a visually pleasing trajectory of the oscillating particle around the source. Additionally, if observed closely there is an abrupt circular formation in the centre caused due to the limits of closeness between the radius of the source and active particle during collisions.

2.2 Brownian motion-induced noise

2.2.1 Brownian Motion

A real-valued stochastic process $(B_t)_{t\geq 0}$ defined on a probability space $(\Omega,\ F,\ P)$, indexed by a set $T \in [0,\ \infty]$, is a standard Brownian motion or a Wiener process if it satisfies the following:
1. $B_0 = 0$, or $B_0^x = x$ for Brownian motion on x
2. For any $n \in N$, $0 \leq t_0 < t_1 < ... < t_n$, the increments are $B_{t_1} - B_{t_0}$, $B_{t_2} - B_{t_1}$, ..., $B_{t_n} - B_{t_{n-1}}$ are independent.
3. For any $s, t \geq 0$ the random variable $B_{s+t} - B_s$ is distributed normally with parameters 0 and t which means that

$$P\left(B_{s+t} - B_s\ \in A\right)\ = \frac{1}{\sqrt{2 \cdot \pi \cdot t}} \int_A e^{\left(-\frac{|x|^2}{2 \cdot t}\right)} dx$$

where A is a $\sigma - algebra$ of Borel sets of R, $A \in B\left(R\right)$.
(*Increments are Guassian*)
4. all sample paths $t\ \rightarrow B_t\left(\omega\right)$ are continuous. (But not differentiable anywhere)
The Brownian motion as a phenomenon was discovered by the biologist Robert Brown in 1827 while studying pollen particles floating in the water on a microscope. He observed the minute pollen particles making a jittery motion, based on which he concluded that the pollen might be alive. This was not the case, as Albert Einstein would later show in 1905 through probabilistic modeling. Einstein observed that at relatively higher temperatures/kinetic energy, the water molecules moved at random, and therefore a particle suspended in this fluid would be impacted by a random number of collisions of random strength from arbitrary directions at any observed short period of time. It was this random molecular bombardment on the particles that caused the confusion as observed by Robert Brown. The first rigorous proof of the true existence of the Brownian motion on some probability space was given by Norbert Wiener in 1923, before the establishment of the foundations of modern probability by Andrey Kolmogorov in 1933. Hence, the Brownian motion is also often referred to as the Wiener process[The Editors of Encyclopedia Britannica (2022) [Szabados (2010).

Diffusion constant 'D' a macroscopic variable, and its relation to the atomic properties of the matter obtained by Albert Einstein, around the regime of 1 micron, can be written as [Einstein and Cowper (n.d.):

$$D = \mu \cdot k_B \cdot T = \frac{R \cdot\ T}{N_A \cdot\ 6 \cdot\ \pi\ \cdot\ \eta\ \cdot\ a} = \frac{k_B \cdot T}{6 \cdot\ \pi\ \cdot\ \eta\ \cdot\ a} \tag{6}$$

where μ is the mobility, R is the gas constant, $N_A = \frac{6.06 \cdot 10^{23}}{mol}$ is Avogadro's number, T is the temperature, η is the viscosity of the liquid, a is the radius of the Brownian particle and $k_B = \frac{R}{N_A}$ is the Boltzmann constant. This equation follows the idea of the *Fluctuation-dissipation theorem* from statistical mechanics, which here expresses that thermodynamic or kinetic energy fluctuations in a physical variable can contrariwise predict the resistance to the fluctuations which cyclically then produce heat through drag, in a system that obeys Ludwig Boltzmann's principle of 'detailed balance'.

The Langevin Equation [Pavliotis (2014)] is a stochastic differential equation that describes the evolution of a system that encompasses deterministic and probabilistic variables [Sjögren (n.d.)]. The Langevin equations of motion for the Brownian particle are as follows:

$$\frac{d\,x\,(t)}{dt} = v\,(t) \tag{7}$$

$$\frac{d\,v\,(t)}{dt} = -\frac{\gamma}{m} \cdot v\,(t) + \frac{1}{m} \cdot \xi\,(t) \tag{8}$$

In the Equations, 7 and 8, the variables $x(t)$ and $v(t)$ are the time-dependent position and velocity of the Brownian particle. $\gamma = 6\pi\eta a$, γ is the frictional coefficient described by the stokes law as a function of viscosity η and radius a. And finally $\xi(t)$ is a stochastic variable of random force that describes the noise caused by the collisions with smaller molecules of the surrounding fluid medium. This effect of fluctuating force can be mathematically described using the first and the second moments,

$$< \xi\ (t) >_{\xi} = 0,\ \&\ < \xi\ (t_1)\ \xi\ \ (t_2) >_{\xi} = g \cdot \delta\,(t_1 - t_2) \tag{9}$$

The 1st and the 2nd moment in the above equation 9 correspond to the mean and variance of the stochastic variable respectively. Since the mean is 0, using Equation 8 the average force in the Langevin equation can be obtained as $f(t) = -\gamma \cdot v(t)$. The variance is strangely represented using a Dirac delta function multiplied by variable 'g' the amplitude of the fluctuating force. While the amplitude g_i changes at different time intervals $dt_i = (t_1 - t_i)$, the defined $\delta(t_i - t_{i+1})$ does not change since the discrete interval remains constant. This is consistent with the observed dynamics since the frequency of fluctuating collisions is so high in a regular time interval that memory between forces at different times will simply be lost due to the newer collisions. In summary, it can be concluded based on the relations in Equation 9 with mean 0, a changing variance, and from the defined dynamical model in Equation 7 and 8, that the fluctuating force has a *Gaussian distribution.*

The solutions to the Langevin Equation 7 and 8 can be derived as follows,

$$x\,(t) = x_0 + \int_0^t v\,(s)\,ds \tag{10}$$

$$v\,(t) = e^{-\frac{t}{\tau_B}} \cdot v_0 + \frac{1}{m} \cdot \int_0^t e^{-\frac{(t-s)}{\tau_B}}\,dW\,(s) \tag{11}$$

Where 'W'is a Wiener process (Brownian process) and $dW(t) = \xi(t)\,dt$; v_0 and x_0 are the initial velocity $v(t)$ and position $x(t)$ respectively. Additionally $\tau_B = \frac{m}{\gamma}$ is the Brownian time scale for the particle velocity relaxation. Through Equations 10 and 11 position x(t) and velocity v(t) are conclusively Gaussian processes since dW is a Gaussian process.

Using 1st and 2nd order moments of Equation 10 and 11 and further derivations, the following relations are concluded for the mean and variance of position and velocity.

$$< v\,(t) > = v_0 \cdot e^{-\frac{t}{\tau_B}} \tag{12}$$

Equation 12 is the mean velocity with a free initial velocity variable v_0, using the first moment. The expansion for the 2nd moment of velocity yields the following,

$$<< v^2\,(t) >_\xi >_{eq} = \left(< v_0^2 >_{eq} - \frac{g \cdot \tau_B}{2 \cdot m^2} \right) \cdot e^{-\frac{2 \cdot t}{\tau_B}} \; + \; \frac{g \cdot \tau_{\;B}}{2 \cdot \; m^2}$$

where '**g**' denotes the variance of W(t). As seen earlier in Equation 9 variable 'g' denotes the amplitude of the fluctuating force or rather the amplitude of random noise. Deriving the above equation for 'g', at equilibrium $<< v^2\,(t) >_\xi >_{eq} = \frac{k_B\,T}{m}$ gives the following expression,

$$g = \frac{2 \cdot \; m \cdot \; k_B \cdot \; T}{\tau_B} = 2\,\gamma\,k_B\,T \tag{13}$$

This equation, similar to the diffusion constant, springs from the *Fluctuation dissipation theorem*, this time expressing the balance between system-impeding force friction γ, and the amplitude of noise strength g of the system-driving force. This balance is constrained by thermal equilibrium for long time factors.
Using equations 12 and 13 the expression for the variance of velocity is obtained by taking the 2nd central moment of v(t) and is written as follows,

$$\sigma_v^2\,(t) = << \left(v\,(t) - v_0 \cdot e^{-\frac{t}{\tau_B}} \right)^2 >_\xi >_{eq} = \frac{k_B \cdot T}{m} \cdot \left(1 - e^{-\frac{2 \cdot t}{\tau_B}} \right) \tag{14}$$

With the knowledge of the mean from Equation 12, the variance from Equation 14, and the distribution from Equation 10 and 11, the conditional probability distribution function for velocity, conditioned on the mean '$\mu_v(t)$' from Equation 12, can be written as follows,

$$f_v\,(v\,(t) \;|\, \mu_v\,(t)) = \frac{1}{\sqrt{2\,\pi \;\; \sigma_{\;v}^{\;2}\,(t)}} \cdot \;\; exp\left(- \frac{(v - \mu_{\;v})^2}{2 \cdot \;\; \sigma_{\;v}^{\;2}\,(t)} \right)$$

Similarly, the positional mean and variance relation can be derived from the 1st and 2nd moments of x(t). Taking the 1st moment of position x(t) by substituting Equation 11 into Equation 10, the mean position can be written as in Equation 15, with the initial position and velocity free-variables x_0 and v_0.

$$< x\,(t) >_\xi = x_0 + v_0 \cdot \tau_B \cdot \left(1 - e^{-\frac{t}{\tau_B}} \right) \tag{15}$$

The displacement variance of the particle, from the starting point, can be calculated using the 2nd central moment of position x(t) with respect to the initial position x_0.

$$< (x\,(t) - x_0)^2 >_\xi = \; \tau_B^2 \cdot \left(1 - e^{-\frac{t}{\tau_B}} \right)^2 \cdot \left[v_0^2 - \frac{g}{2\,m\,\gamma} \right] \; + \; \frac{g}{\gamma^2} \cdot \left[t - \tau_B \cdot \left(1 - e^{-\frac{t}{\tau_B}} \right) \right]$$

Constraining the above equation to equilibrium, the first term vanishes, and along with it any dependence of initial position x_0 and velocity v_0 from calculating the variance in further calculations. This constraint gives the following two solutions for limits of time close to 0, and limits of time approaching infinity,

$$\lim_{x \to 0} \; << x\,(t) - x_0)^2 >_\xi >= \; t^2 \cdot \frac{k_B \cdot T}{m}$$

Since the functional behavior of interest in this project is the long-term trajectory of the Brownian particle, the expansion from here is in the limit of time approaching large values.

$$\lim_{t\to\infty} << (x(t) - x_0)^2 >_\xi >= t \cdot \frac{2 \cdot k_B \cdot T}{\gamma}$$

Hence the expression for the 2nd central moment can be written more clearly using Einstein's diffusion relation as, [Pavliotis (2014)]

$$< (x(t) - x_0)^2 >= t \cdot \frac{2 \cdot k_B \cdot T}{\gamma} = 2 \cdot D \cdot t \tag{16}$$

Expanding on the Equation 16 the variance for a time-dependent mean can be derived as follows in Equation 17.

$$\sigma_x^2(t) =< (x(t) - \mu_x(t))^2 >= \frac{2 \cdot k_B \cdot T \cdot \tau_B^2}{m} \cdot \left(\frac{t}{\tau_B} - \frac{3}{2} + 2 \cdot e^{-\frac{t}{\tau_B}} - \frac{e^{-\frac{2 \cdot t}{\tau_B}}}{2} \right) \tag{17}$$

And finally, the expression for the variance of displacement for a Brownian particle, starting at a stationary point is derived as follows in Equation 18 [Sjögren (n.d.)].

$$\sigma_x^2(t) =< (x(t) - x_0)^2 >= 2 \cdot D \cdot t \tag{18}$$

With the knowledge of the mean from Equation 15, the variance from Equation 17 or Equation 18, and the distribution from Equation 10 and 11, the conditional probability distribution function for the Brownian particle displacement, conditioned on the mean 'μ_x' can be written as follows,

$$f_x(x(t) \mid \mu_x(t)) = \frac{1}{\sqrt{2\,\pi\,\sigma_x^2(t)}} \cdot exp\left(-\frac{x - \mu_x}{2 \cdot \sigma_x^2(t)} \right). \tag{19}$$

2.2.2 Brownian motion on oscillating active particle

The active particle oscillations described in Section 2.1.1 is an idealized description of the system. In the actual physical system, these oscillations are affected by the Brownian noise and hence the equations derived in Section 2.1.1 are updated to take into account the noise in an effort to make a more accurate description of the active particle oscillation model. This section reasons and derives this updated model.

From the previously described definition of Brownian motion in Section 2.1.1, discrete-time update Equations 3 and 4 are modified. Since Equations 3 and 4 are updated at every time increment 'dt', it is known from Section 2.1.1 that an added Brownian random variable here should have "Gaussian increments," and that these "increments are independent." This is achieved by adding Gaussian noise in a polar coordinate system since the Equations 3 and 4 are polar themselves. Additionally, since the oscillating active particle is a 2-dimensional system, both experimentally and in the simulation, the adapted Brownian noise in the model is also 2-dimensional. These state requirements are formulated into the Equations 3 and 4 by the addition of a normally distributed random

variable to the two polar coordinates of radius ρ, and angle ϕ. Hence the equations for Brownian-induced, time-delayed active particle oscillation can be written as follows with the additional term $\mu(t)$ for the time series input.

$$\phi\,(t+dt) = \phi\ \ (t) + dt \cdot \left(\omega\,(t) \cdot sin\,(\theta\,(t) + \mu\ \ (t)) + \frac{N\,(0,\ \sigma^2)}{\rho\,(t)}\right) \tag{20}$$

$$\rho\ \ (t+dt) = \rho\ \ (t) - dt \cdot\ v \cdot\ cos\,(\theta\ ) + N\left(0,\ \sigma^2\right) \tag{21}$$

Equations 20 and 21 in addition to Equation 2, describe the full set of equations for the Brownian-induced, time-delayed active particle oscillation.

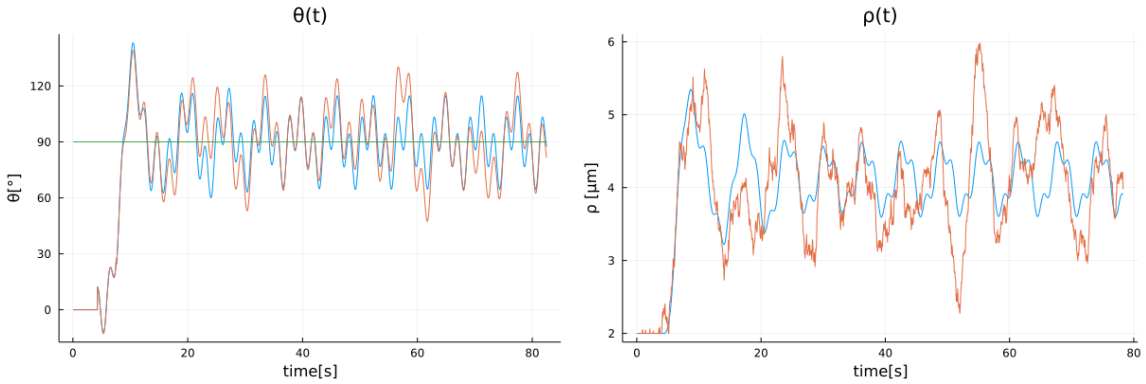

Figure 6: Response of Noisy system to a sinusoidal time series input

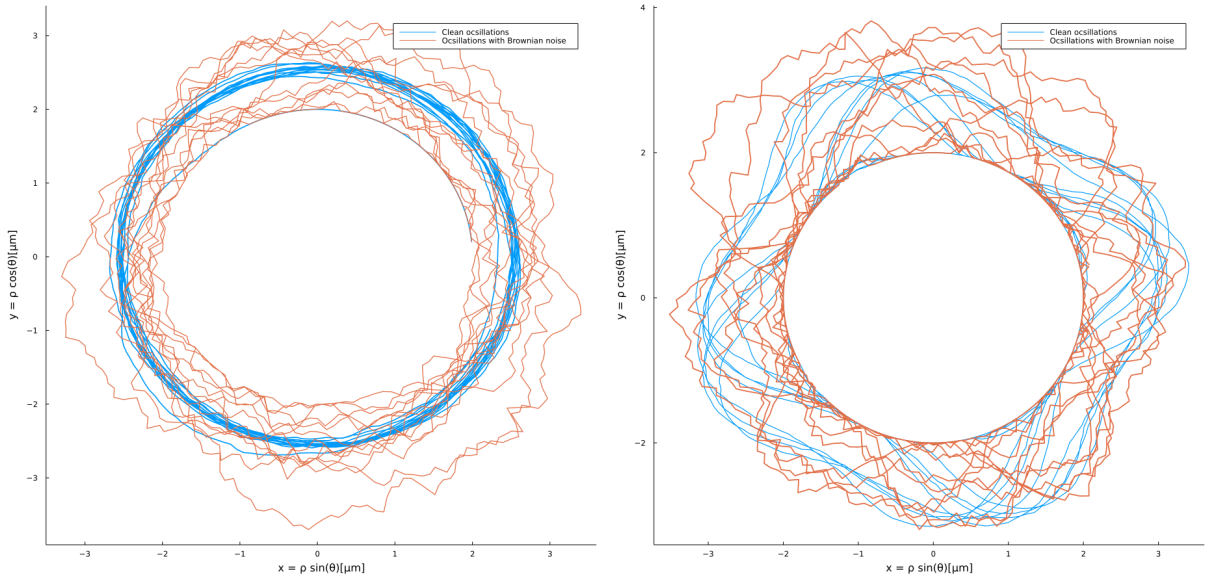

Figure 7: [Left] - Brownian noise in an active particle time-delayed oscillator; [Right] - Brownian noise in an active particle time-delayed oscillator with input $\mu(n)$

The N in the Equation 20 and 21 is the normal distribution $N(\mu, \sigma^2)$ set to have a mean of 0, $\mu = 0$, and the relation for the standard deviation σ and variance σ^2 is derived from

Equations 18 of Brownian motion. And hence the relation for variance is written as,

$$\sigma = \sqrt{2 \cdot D \cdot t} \tag{22}$$

This might seem overly simplistic since the derived equation for the distribution and variance of a Brownian motion particle with changing mean in Equations 17 and 19 are not this simple. This is not a problem with Equations 20 and 21, even though the mean is rotationally moving, the equations themselves only describe the position in the next discrete time. So one can assume the mean of 0 centering the Gaussian along the current position in the evolution. Similarly, the variance is chosen from Equation 18.
Since the oscillator simulation runs on a discrete loop equation 23 can also be written as

$$d\sigma = \sqrt{2 \cdot D \cdot dt} \tag{23}$$

As the oscillator simulation runs over the simulation time, the variance in Equation 23 accumulates over time at the rate of the square root of the simulation time step dt, $\sqrt{dt}$. Hence the variance keeps growing, producing an ever-long stretched Gaussian curve with time. The parameter 'D', the diffusivity constant from Equation 6, used for the variance relation in Equation 23, is set to be a value of **"D=0.08"** as per the diffusivity coefficient of the real experimental setup.

Once again the equations in this section are numerically simulated with Equations 2, 20 and 21 for a better understanding of the dynamics involved. Figure 6 shows the response of the formulated system to a sinusoidal time series input through the plots of time-varying delayed angle θ and radius ρ. Figure 7 shows the plot of Brownian noise in an active particle time-delayed oscillation system. The blue line represents the simulation of oscillations without any Brownian noise and the orange line represents the oscillations with the presence of Brownian noise. The graph on the right in Figure 7 shows the plot of the same simulation with an added sinusoidal input to the system, following Equations 20 and 21. This visually pleasing nonlinear input response in Figure 7 is utilized as a nonlinear expansion of a given input in the reservoir computing which is discussed later.

2.3 Reservoir computing

2.3.1 Recurrent neural networks, a brief discussion

Machine learning came about with attempts to mathematically model the workings of neurons and neural networks in a larger effort to understand the brain. In 1943, Walter Pitts and Warren McCulloh published a paper titled "*A logical calculus of the ideas immanent in nervous activity*" [McCulloch and Pitts (1943)], marking the first attempts of a logical/mathematical model of neural networks and human cognition at large. In 1950, Alan Turing proposed the famous *"Turing test"*. In 1952 Arthur Lee Samuel writes a program for the game of checkers which improved after every game, coining the term "Machine Learning". Kick-starting the ubiquitous presence and study of the field of Machine Learning [Molnar et al. (2020)].

The idea of **Recurrent Neural Networks (RNNs)** came about in 1986 with the publication of the paper "Learning representations by back-propagating errors" by David Rumelhart, Geoffrey Everest Hinton, and Ronald J. Williams, and is today one of the major pillars of Machine Learning. In the most basic terms, RNNs map some defined state space "s" to an action "a" with an emphasis on the usage of reward functions unlike the sister method of supervised learning, which uses human supervision for training. The Networks automatically adapt a mapping from the state spaces to actions using the reward functions [Sherstinsky (2018)].

$$[s\,(t)\,,\ a\,(s,\ t)\,,\ R\,(s,\ a)\,,\ s'\,(R)]$$

Above is a generic description where for every time step "t" there is a state space "s(t)" which leads to an action "a," the state and the action in turn produce a value in the reward function which then leads to a new state space "$s'(R)$." Additionally, a "return function" can be defined as the sum of the reward function weighted by a discount factor γ, and can be written as

$$return\ :\rho =\ R\,(t=1)+\gamma\cdot R\,(t=2)+\gamma^2\cdot R\,(t=3)\,...\ (until\ a\ terminal\ s\,(t))$$

In this example, if $\gamma < 1$, the return function plays with the reward function to make the network mapping between the state space and action more time impatient as the reward decreases significantly with every time step. Hence the network is geared towards the rewards as fast as possible, often described as "added time impatience." This can be modified to make the network induce a spacial impatience or an interest rate function and so forth based on the task. Altogether, the goal in reinforcement learning is to find a policy function Π such that for every state "s" mapped to an action "a," $\Pi\,(s) = a$ maximizes the value of the return function $|\rho|$ [Terpstra (2016)].

One of the distinguishing features of RNNs is the presence of cyclical connection topology in the trained networks. This cyclical nature of RNNs allows the system to develop activations even with a null input signal. When given a real input signal, the system saves a nonlinear transformation of the input into the network memory or the internal state. This process description classifies the RNNs to be a mathematically dynamical system, in contrast to the function-like behavior of feed-forward networks. This way of describing RNNs gives some clues regarding the effectiveness of RNNs in sequential learning problems or time-series predictions [Terpstra (2016)].

2.3.2 Echo state networks and time series prediction

Around 2001, two new independently published works introduced a foundationally new approach to the Recurrent Neural Network paradigm, "Liquid State Machines" by Wolfgang Maass and "Echo State Networks" by Herbert Jaeger, which then collectively came to be referred to as Reservoir Computing (RC) [Lukoševičius (2012)]. In a simplistic description, while working with RNNs, the last layers were observed to be the most important and it was also observed that RNNs often seemed to perform well even with just the final network weights. This led to an attempt to bypass training with multiple layers

and instead have a fixed randomly generated reservoir and train only this readout. This paradigm was eventually realized into the field of 'Reservoir Computing'. Eventually, it became fairly common to produce and train the reservoir separately, unlike the original conception, which is the case in this project as well[Lukoševičius and Jaeger (2009a)].

Within the vast plethora of RNN models, this writing is concerned with dynamical systems. In particular, nonlinear dynamical systems. The RC paradigm particularly belongs to the class of RNNs with a deterministic update, directed connections, supervised training, and a nonlinear filter implementation which can transform an input time series signal into an output time series signal. Though the RC paradigm belongs within the categorization of RNN it overcame a major shortcoming of the regular RNNs by avoiding, training on, multiple giant network layers altogether. This discovery, which was made before the advent of Graphical processing units (GPUs) and much faster parallel processing methods for matrix computations, was a giant performance leap for the available hardware in the early 21st century. Nevertheless, it has become apparent in the present day that large RNNs are still largely useful, and are commonly run on various powerful parallel processing modern hardware.

The RC paradigm tries to optimize on the RNN training in the following ways. A random recurrent neural network is generated and remains unchanged through the training phase. This particular RNN is the so-called "Reservoir". This reservoir is excited with the input during its creation and maintains in it a form of nonlinear transformation of the input history. The output signal or rather, the prediction, is obtained through training the input excited reservoir with the input itself as the aim using linear regression. One could also train the network to map any 2 kinds of data. The trained weights out of this process are then used to make the predictions further into the signal. The following sections elaborate more thoroughly on the workings of the RC paradigm for temporal tasks.

2.3.3 Activation State/Kernel : x(n)

The mapping between the input data $\mu(n)$, where 'n' is the time step, and the desired output data $y(n)$ for generating predictions, involves multiple processes. The first and the most important of which is the making of a state $x(n)$ onto a collection of state space χ. The predictive mapping between the input $\mu(n)$ and the output $y(n) = \mu(n+1)$, in most cases, cannot simply be linear throughout the data. This most certainly is not the case for nonlinear signals by definition, which are studied in this project and will be discussed in later sections. Hence, the ESN networks perform a nonlinear expansion of the input onto some high-dimensional feature space which can then be reduced to a simple linear formation on this space based on the choice of nonlinearity. This nonlinear high dimensional projection is performed through the state $x(n)$ or $x(\mu(n))$. This simple linear formation in the high-dimensional space can then be used to train W_{out} through linear regression, which is then used to make prediction outputs $y(n) = W_{out} \cdot x(\mu(n))$. This method is often referred to as the "Kernel trick" where $x(n)$ acts as the kernel. This is elaborated more in Section 3.1.1 [Lukoševičius and Jaeger (2009a)].

The state $x(n)$ is a vector expansion of a single input $\mu(n)$, and serves as a nonlinear input expansion. Additionally, it also serves as a temporal memory, since the input of interest is a temporal signal i.e. every signal position in the present is dependent on the past positions. Hence the state can also be represented as a function of the past input points as $x(n) = x(\mu(n), \mu(n-1), \mu(n-2)...)$ or simply $x(n) = x(\mu(n), x(n-1))$ but this is usually clear from the Equations 24, 25 and 26 which will be discussed later.

This inter-state temporal relation is made more obvious by plotting one state against another state. This produces an 'attractor' chasing itself, which is seen in continuous temporal signals, solidifying the existence of some temporal memory in the states $x(n)$ belonging to the state collection matrix χ [Doyne Farmer (1982)]. Figure 8 shows this for Sine signal as the input using the simple ESN code formulated from Equations 24, 25 and 26 discussed later.

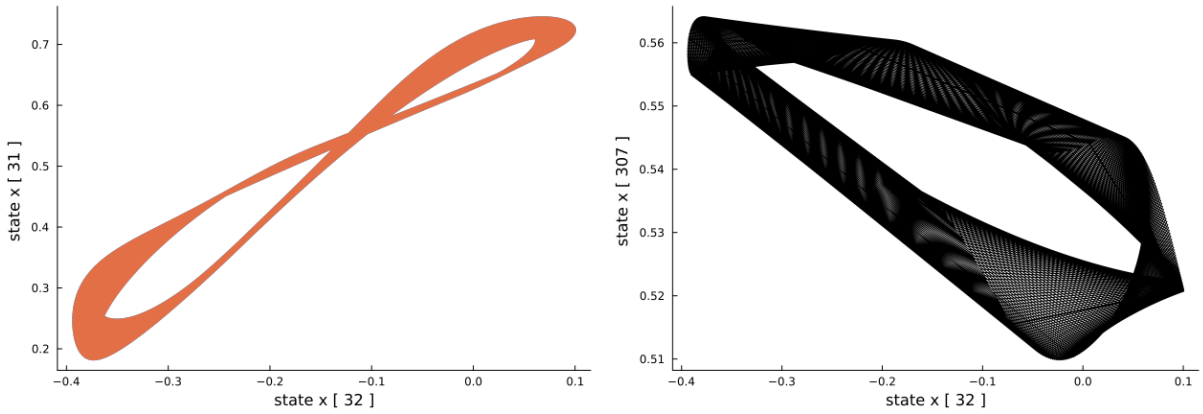

Figure 8: Dynamic Attractors for state [left] x(32) vs x(31), and [right] x(32) vs x(307)

Formulating the basic model for ESN

The update equation for the Echo-state-network can be written as follows,

$$x'(n) = tanh\left(W_{in} \cdot [1;\ \mu(n)]\right) + W \cdot x(n-1) \tag{24}$$

$$x(n) = (1-\alpha) \cdot x(n-1) + \alpha \cdot x'(n) \tag{25}$$

Where the 'n' is the time step, α is the leaking rate, and x' is the update [Lukoševičius (2012)]. Importantly W_{in} is the input scaling matrix that projects every input to the dimensions of the reservoir with additional accommodation for a bias term, W is the reservoir matrix or the input coupling matrix. The reservoir matrix is essentially a random matrix that couples the current state $x(n)$ with the past state $x(n-1)$. This can be seen as such in the Equations 24 and 25, and since these are discrete-time continuous the state $x(n-1)$ will further be related to $x(n-2)$. Hence the reservoir matrix introduces a coupling between the states, and as such can be tweaked with various parameters to

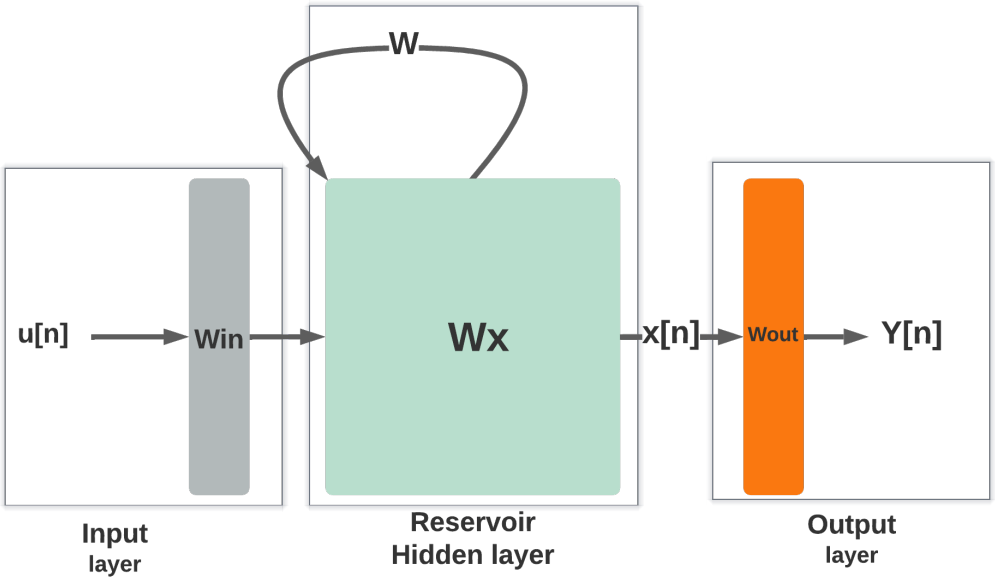

Figure 9: Visual summary of an ESN

affect different kinds of coupling between the current and past states, such as the spectral radius or the sparsity of the matrix.
One of the important parameters in Equation 25 is the leaking rate $\alpha \in (0, 1]$. The leaking rate, sometimes also called the decaying rate, determines the speed of the reservoir update dynamics. Additionally, the leaking rate is restricted $\alpha \in (0, 1]$, such that the state $x(n)$ does not retain or leak more activation than that caused by the input $\mu(n)$.

2.3.4 Readouts out of the reservoirs

Linear regression is the most commonly used readout method in reservoir computing. Since the method is born out of the methods of solving the least squares in linear algebra, the process can be described analytically. This is in contrast to the usual numerical descriptions of readout methods such as the well-known gradient descent. This also speaks to the similarity of the RC methods to that of the Kernel methods, since the final steps for a nonlinear signal prediction utilize the linear regression method.

Ridge regression, also known as the **Tikhonov regularization** [Lenzen and Scherzer (2004)], is a subset of linear regression with an additional regularization coefficient term. The method is commonly used when system parameters are highly correlated. This method is helpful in tilting an overfit curve with limited training data to better fit the full data, including the test set.
Figure 10 illustrates the visual difference in linear versus ridge regression. Unlike the linear curve fit, the reg regression fit produces a slightly tilted line as a consequence of the reg coefficient, and as can be seen in the Figure 10, this fits better with the entirety of the data set than the linear fit which is limited to the training data in its accuracy.

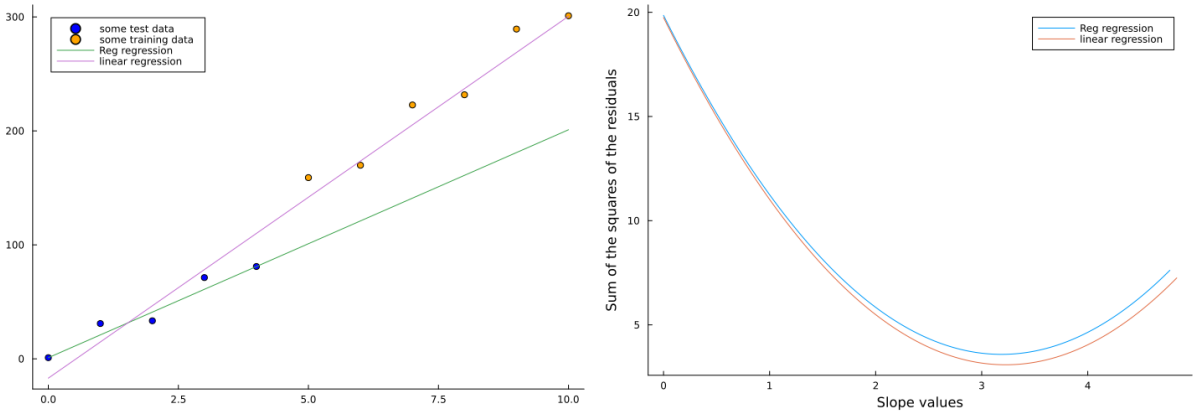

Figure 10: Visual difference between Liner and Reg coefficient

This can be further formalized with the equation for the ridge estimator as,

$$\beta_r = \left(X^T \cdot X + \lambda \cdot I\right)^{-1} \cdot X^T \cdot y$$

which is obtained through the constrained optimization of

$$min_\beta \left\{(y - X \cdot \beta)^T (y - X \cdot \beta) + \lambda \left(\beta^T \cdot \beta - c\right)\right\}$$

Where λ is the ridge coefficient or rather the Lagrange multiplier in minimization formulation, y is the statistical regressand, X is the statistical model matrix, and I is the identity matrix. The reg coefficient is generally calculated as the squared summation of the system parameter sparing the intercept value. The coefficient can also simply be used as a control parameter that can be chosen based on the specific problem to solve.

The linear readout layer is the last algorithmic step toward obtaining the final output from the network. This can be formulated as the product of the optimal weight vector obtained through reg regression W_{out}, with the state vector $x(n)$ which gives a single point output written as,

$$y(n) = W_{out} \cdot x(n) \tag{26}$$

where the variable 'n' in Equation 26 sequentially enters and is bounded within the 'training time' regime. Furthermore, the vector W_{out} can be elaborated through the Tikhonov regularization discussed earlier as shown in Equation 27 and 28.

$$W_{out} \cdot X = Y_{target} \tag{27}$$

$$W_{out} = Y^{target} \cdot X^T \left(X \cdot X^T + \beta \cdot I\right)^{-1} \tag{28}$$

Here the variable Y_{target} is a vector constructed through the collection of single output value $y(n)$ in the training. The generation of the reservoir-state matrix and the W_{out} are done separately in contrast to the traditional RC computing method that does not discriminate much between the two generations. Since the reservoir state collection matrix and the W_{out} perform different functions, nonlinear expansion vs linear fitting, it has become common to separate the two generation procedures. The latter algorithmic procedure of readout can then be further classified into predictive and generative readouts.

Predictive mode is a readout method in which the network only predicts 1 step ahead of the given time series signal. When run for the entire testing length of the data, this effectively allows the system to course-correct any error in the previous prediction since the next step prediction is made based on the original data at the current time step. This keeps the prediction from accumulating error over training lengths. Since unlike most RNN methods the computation to be performed for the next-step prediction is significantly cheap, this allows the system to make next-step predictions fairly fast after waiting for the current time data, making this method useful for applications by building some quick functions response on the prediction.

Generative mode is a readout method in which the network takes its own prediction from the past time step to make a prediction for the next time step. This makes an accurate signal prediction over time much harder since the error accumulates over time. However, generative predictions are a more accurate representation of the learned signal within the network. When successfully implemented, implementing this method correctly can lead to a powerful prediction tool.

3 Experimental analogous computational methods

3.1 The reservoir

The reservoir matrix is simply a coupling matrix, as introduced in the ESN Equations 24 and 25. However, the reservoir itself plays a more important role in this project in providing neuron activations in time. The reservoir performs a **nonlinear expansion of the input** in addition to **coupling past input activation states** to the current state. These nonlinear expansions and memory functions of the reservoir are naturally fulfilled in a **nonlinear dynamical system**, making it an ideal reservoir, and the choice of this nonlinear dynamical system in this project is the 'time delayed active particle oscillation system'. This section delves deeper into these aspects.

3.1.1 Nonlinear input expansion

A reservoir can be seen as an operation that expands a given input $\mu(n)$ into a higher dimensional space using a nonlinear function to produce the points in new dimensions. This input expansion is similar to the known 'Kernel methods' which use the kernel trick, explained in this section below [*Approximation of dynamical systems by continuous time recurrent neural networks* (1993)]. However, the reservoir computing method allows for the holding of past temporal memory of the state $x(n) = x(\mu(n),\, \mu(n-1)...\,\mu(1))$ making it a recurrent state network, elaborated in Section 3.1.2.

The kernel trick is a common method, often used in support vector machines, where input data 'μ_i' is projected into a higher dimensional space with a nonlinear function $\Phi(\mu_i) = [\phi_1(\mu_i)\ \phi_2(\mu_i)\ ...\ \phi_n(\mu_i)]$, where the kernel is simply

$$K\,(a,\ b) = \sum_{i=I} \lambda_i \cdot \phi_i\,(a) \cdot \phi_i\,(b) = \Phi\,(a) \cdot \Phi\ (b)^T$$

and the output can be simply read of as $y = w^T \cdot \phi\,(\mu_i)$, where '$w$' is the trained weight through linear regression [Shalizi (2020)]. A nonlinear input data then becomes linearly separable in this higher dimensional space. The choice of the nonlinear expansion can be made using the domain knowledge of the data, and the choice of this nonlinear function has to satisfy Mercer's condition from Mercer's theorem. However, since the dimensional expansion is through a nonlinear function this can be done to make an infinite dimensional matrix. Since this clearly is computationally impractical, the size of the dimension has to be made as a parameter choice. A parameter for the number of dimensions at which the projection of the given nonlinear input signal is linearly separable. Equations 20, 21, and 2 act as kernel expansions in this project, and W_{out} from Equation 27 can be seen as a vector with higher dimensional curve fitting parameters in its components.

Figure 4 from the theoretical section shows an initial transient and the later driven steady-state response of the oscillator angle θ. It is this input-driven steady-state response that is used as the nonlinear expansion of input, while the transient response is removed from the network.

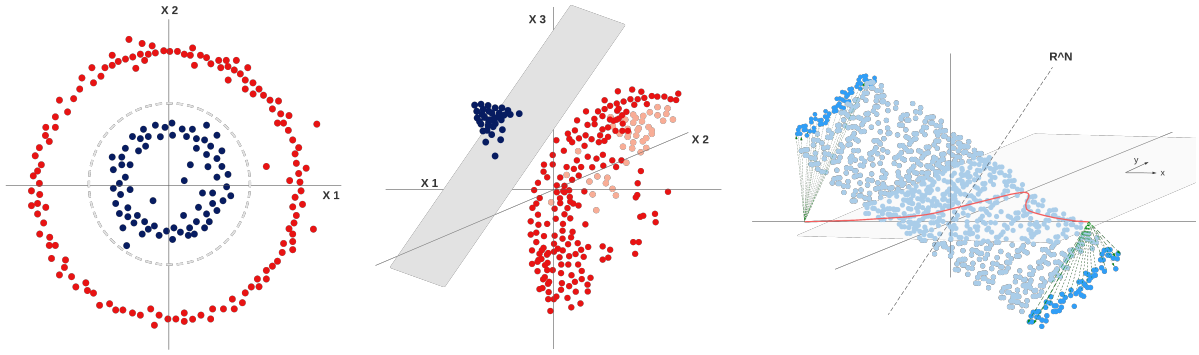

Figure 11: Visualizing linearly separable feature map of a Gaussian [iii], Visualizing classification feature map[ii] & the resulting boundary line on data [i]

The graphics on the left [i] and [ii] in Figure 11 are the visualization of the well-known Polynomial kernel in a classification task. The graphic on the right [iii] in Figure 11 is a visualization of a potential linearly separable manifold in the feature space of a simple Gaussian curve.

3.1.2 Temporal memory

What makes the RC method distinct from a kernel method, is the ability to hold temporal memory of the past activation states and couple them to the current state. These temporal dynamics make the Reservoir Computing method a Recurrent Neural Network. Coupling between states $x(n)$ mainly exists through two processes: first through the natural dynamical property of the time-delayed active particle oscillations, and second, through the method of multiplexing explained in Section 3.2.3 [Tanaka et al. (2019)].

Time delayed oscillations from dynamics presented in the theoretical Chapter 2, especially in Equations 2 which then embedded into Equations 20, 21, show that there is a time delayed coupling variable dependency from $\phi(t)$ and $\phi(t-\delta t)$ translating further through the equations describing the dynamics. The discrete-time interval of delayed coupling is simply *Delay step size* $= Z\left\{\frac{\delta\, t}{dt}\right\}$. This coupling through delayed feedback forms a natural recursive coupling in the states $x(n)$ through the dynamical evolution [Lukoševičius (2012)].

$$x\left(n\right) \quad = \quad x\left(\mu\left(n\right),\, x\left(n-1\right),\, x\left(n-Z\left\{\frac{\delta\ t}{dt}\right\}\right)\right)$$

Hence the dynamical system of t*Time-delayed active particle oscillations* forms a great reservoir in this project through its natural dynamic evolution. Additionally, since the reservoir has a mathematical description, it makes for an *Interpretable machine learning method* which is later explained in Section 4.5.

3.2 Active particle reservoir computing

This section delves into the construction of reservoir computing using the time-delayed active particle oscillation system, introduced in the theoretical foundations of Chapter 2. In a nutshell, this is done through the simulation of Equations 20, 21 and 2 as neuron activation functions for the input, and then through prediction plus readouts as described in Section 2.3.4. However, in comparison to a vanilla Echo State Networks, discussed in Section 2.3.2, there are major differences here through the choice of the reservoir discussed earlier. These important details are discussed in the section below.

3.2.1 State space collection matrix

One of the most important aspects during training is the construction of the state space matrix. The reservoir neuron activations are collected in a vector for each single discrete time step, the collection of which, through the training time, gives the state space matrix. This state space matrix, in the general 'Batch mode', forms a fixed state space matrix used unchanged through the prediction process. This can, nevertheless, also be tailored as is done in the 'Online mode', where the reservoir is updated during the prediction.

In this project, the time-delayed active particle is simulated as described in Chapter 2, i.e the project uses a simulated reservoir. However, this simulation is a computational reflection of a real physical time-delayed active particle oscillation system, a physical reservoir, that was experimentally set up in the lab. Please note that the details of this experiment are not dealt with in this thesis.

Each state vector is constructed out of the input response from the reservoir system. There are 'training-length' numbers of state vectors i.e single training data point is used to produce one state vector. From the simulation of the system described in Chapter 2 there are 3 simulation variables that can be used as the reservoir activation state in the network, radius $\rho(t)$, angle $\phi(t)$, and angle $\theta(t)$. This project uses angle $\theta(t)$ as the reservoir activation state akin to the ESN state $x(n)$ introduced in Equations 24 and 25. Hence the update equation of the network for activation state can be written using Equations 2, 4 and 5, as:

$$\Theta_x(n) := \theta\ (n) = \phi\ (n) - \phi\ (n - \Delta\, t)$$

$$\phi\ (n+1) = \phi\ (n) + dt \cdot \omega\ (n) \cdot sin\left(\theta\,(n) + W_{in} \cdot \begin{pmatrix} 1 \\ \mu\ (n) \end{pmatrix}\right) \tag{29}$$

$$\rho\,(n+1) = \rho\,(n) - dt \cdot\ v \cdot cos\,(\theta)$$

Equations 29 are the complete set of equations for the description of the activation state $\Theta_x(n)$, where n is the discrete simulation time step of update resolution $dt = 1$, the subscript 'x' refers to the place order of Θ in the state space matrix where $x \in [1, Training\,length]$, W_{in} input scaling matrix as described in Section 2.3.3, and $\Delta\, t = Z\left\{\frac{\delta\, t}{dt}\right\}$ is the discretized time delay. Again, a complete single-line description of

the kernel expansion through activation state Θ, can be written using Equations 2, 4 and 5 as

$$\theta\,(n+1) = \theta\,(n) + \frac{v_0}{\rho\,(n)} \cdot dt \cdot \Bigg[sin\left(\theta\,(n) + W_{in} \cdot \begin{pmatrix} 1 \\ \mu\ (n) \end{pmatrix}\right) - sin\left(\theta\ (n-\Delta t) + W_{in} \cdot \begin{pmatrix} 1 \\ \mu\ \ (n-\Delta t) \end{pmatrix}\right)\Bigg] \quad (30)$$

The activation function is generally a nonlinearity introduced in a network that can help with the kernel projection as explained in Section 3.1.1. The choice of nonlinearity in the activation state is the sinusoidal function 'Sine', which transforms the input $\mu(n)$. Hence the activation function in this reservoir computing network is the Sine function.

3.2.2 Noisy reservoir

The equations in 29 are an idealized simulation of the physical time-delayed active particle system. In the physical experimental setup, the oscillating particle experiences Brownian noise, the model of which is explained in Section 2.2. Hence the physical reservoir is intrinsically a noisy system. Similar to Equation 29, the set of equations describing a noisy reservoir can be formulated using Equations 20, 21 and 2 as

$$\Theta_x(n) := \theta\ (n) = \phi\ (n) - \phi\ (n - \Delta\, t)$$

$$\phi\ (n+1) = \phi\ (n) + dt \cdot \left[\omega\,(n) \cdot\ \ sin\left(\theta\ (n) + W_{in} \cdot \begin{pmatrix} 1 \\ \mu\ \ (n) \end{pmatrix}\right) + \frac{N\,(0,\ d\sigma)}{\rho\,(n)}\right] \quad (31)$$

$$\rho\ (n+1) = \rho\ (n) - dt \cdot\ \ v \cdot\ cos\,(\theta\ ) + N\,(0, d\sigma)$$

where the variance for the normal distribution function, added recursively for every time step, is $d\sigma = \sqrt{2 \cdot D \cdot t}$ from Equation 23. Equations 31 are the complete set of equations for the description of the activation state $\Theta_x(n)$, from a *Noisy-Reservoir*. The prediction comparisons with the ideal reservoir and analysis of noise on prediction performance are discussed in the later chapter.

3.2.3 Coupling through multiplexing

In addition to the natural time-delayed input-state coupling through the natural dynamics in the reservoir, this project uses an additional method of activation state coupling through the method of input multiplexing [Bianchi, F., Livi, L., Alippi, C. (2017)]. This section discusses the two multiplexing methods used in this project, simple input multiplexing, and time multiplexing [Röhm and Lüdge (2018)].

Simple input multiplexing is a method by which multiple input values are extracted from a single input out of the input signal. This is done by simply multiplying the single input with the random numbers of desired multiplexing quantity. This is effectively a multiplicative increase in the size of the input. For example, an input signal of size 2, with a multiplex of 3 would be an effective input of size 6. This is demonstrated in Figure 12, where $u[1]$ and $u[2]$ are the real input from the signal, and $m[1], m[2].., m[6]$ are the multiplexed input.

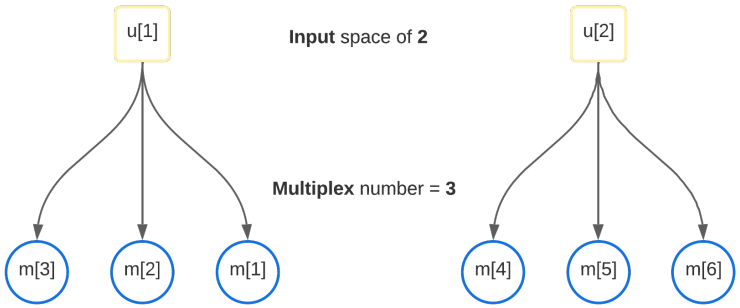

Figure 12: Simple multiplexing

This method of increasing the input in a multiplicative manner is similar to the W_{in} input scaling matrix. The code implementation of multiplexing in this project is done through functional modifications of the W_{in} matrix. Additionally, such input size scaling is very useful in training a larger state space matrix in a short duration, since the laser-trapped particles tend to escape control in a relatively short time, in the experimental setup of the physical reservoir.

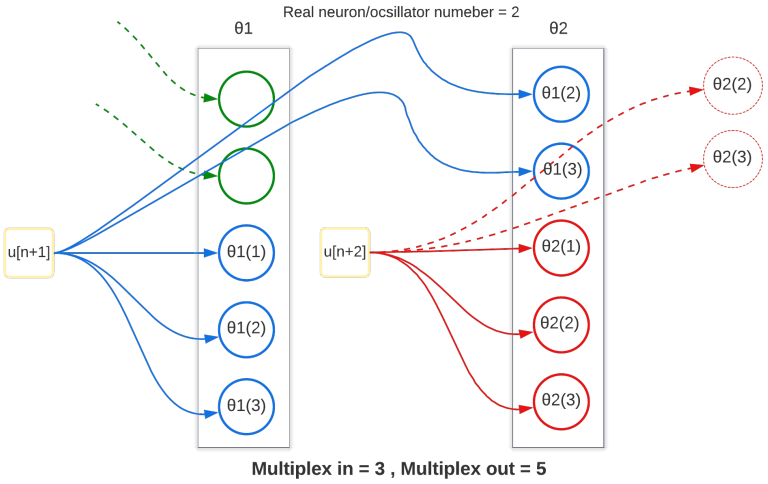

Figure 13: Time multiplex using $\mathbf{M_{out} \geq M_{in}}$

Time multiplexing through $[\mathbf{M_{out} \geq M_{in}}]$ is a method by which past state evolution memory that was multiplexed can be stacked into the current state. This is an artificial

method of affecting the temporal memory of the network. M_{out} and M_{in} are the Multiplex out and Multiplex in variables respectively. Where M_{in} performs the simple input multiplexing, and when $M_{out} = M_{in}$ is equivalent to the simple multiplexing with M. During $M_{out} > M_{in}$ the input first undergoes simple multiplexing as described earlier of the multiplicative multiplex number M_{in}, after which the last "$M_{out} - M_{in}$" number of multiplexed input activation state responses are stacked on the next state vector, as visually demonstrated in Figure 13.

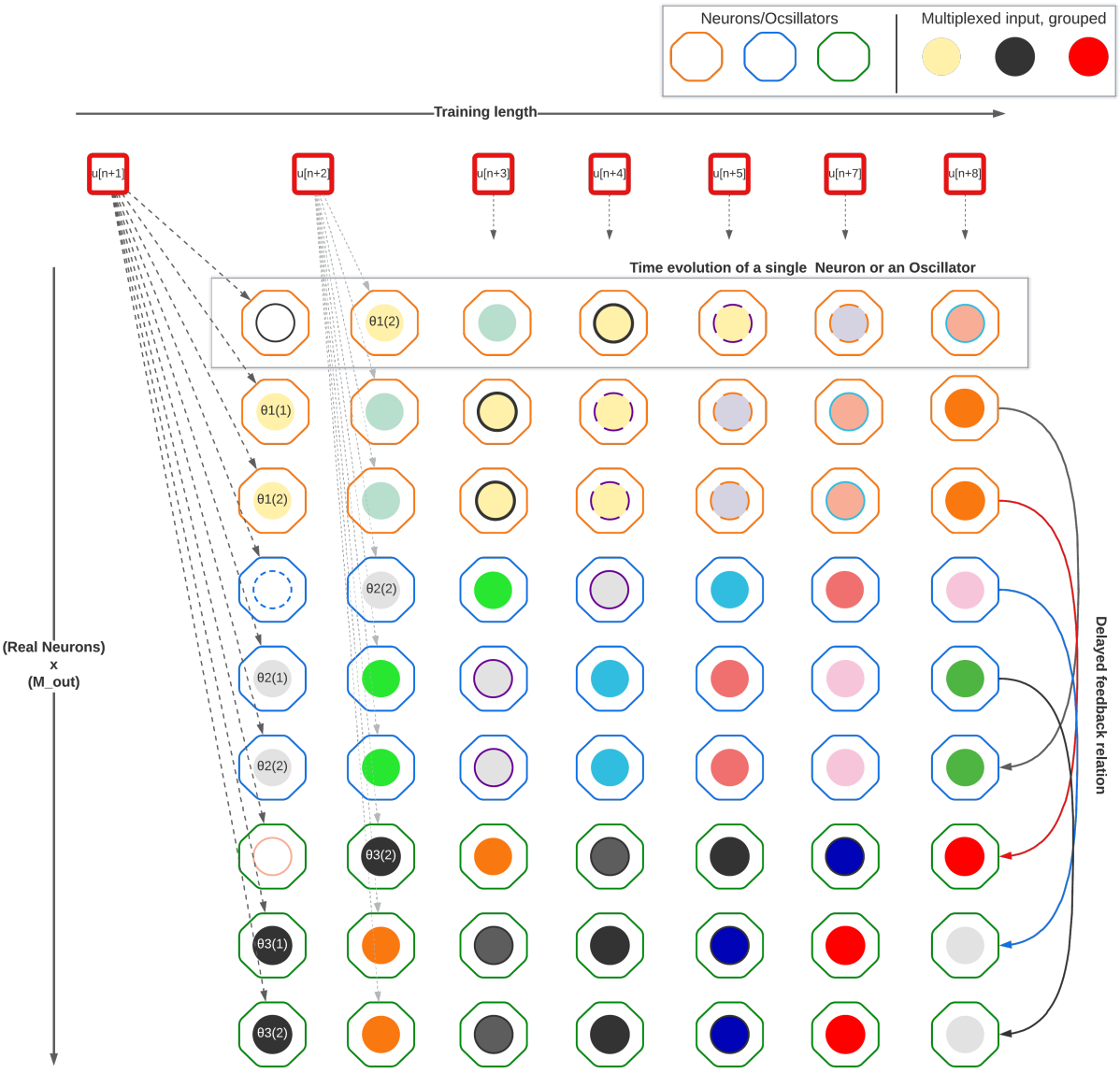

Figure 14: Visual summary of a sample state space matrix

Figure 14 is a visual representation of a sample state space matrix, where the relation between different state vectors and their multiplex relations are highlighted with different colors, with time multiplexing visual on the left, and the time delayed coupling relation on the right. The red squares on the top $\mu(n+t)$ are single input points. The row size of

the matrix is equivalent to the training length (minus the initial length) and the column length of the matrix is the multiplexed values of the real state/neurons i.e Number of states $\cdot\,(M_{in} + (M_{out} - M_{in}))$ = Number of state $\cdot\, M_{out}$. Due to the implementation of the input multiplexing, there is a distinction made between the real intended neurons/oscillators vs the total resultant multiplexed neurons/oscillators, i.e real activation states/neurons vs multiplexed states. The real states correspond to independent oscillators with their own independent input responses, similar to the real physical reservoir which can have multiple oscillating particle systems respond to the same input with different responses based on unique initial conditions, noise, and other factors. And hence in Figure 14, there are a total of 9 components in each state vector even though there are only 3 oscillators.

3.3 Code description

This section briefly highlights the code implementation of the mentioned theory so far. The programming language of choice in this project was 'Julia' and the code base of this project tries to follow some of the styles of the functional programming paradigm. Julia is a language written for scientific computing with great and fast methods for matrix operations and the ability to easily prototype on large data [Bezanson et al. (2012)]. The program implementation of this project is done with minimal usage of external libraries, the only used libraries were for importing data, Plots, and Linear Algebra. This allowed the code to be a more complete mathematical and functional description of the processes and methods undertaken. Figure 15 is a general overview of the program implementation.

Each subsection of this section describes a function in the code base and is a bare-bone functional description of the code, with the exclusion of syntax for imported libraries and initialized variables. Additionally, the Julia environment allows for extravagant usage of loops without the usual need for having to make the code run in parallel. With its fast JIT compiler, and dynamic and strong typing system, it was very well suited for the heavy computation needed in this project.

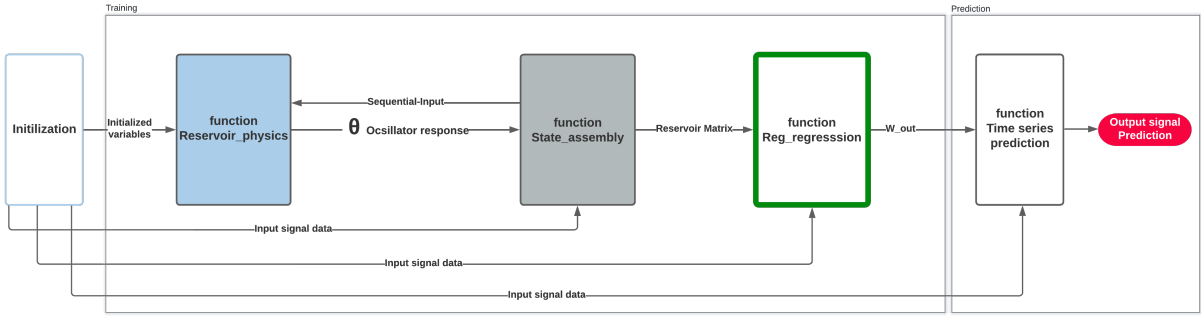

Figure 15: Functional overview of the program

3.3.1 Reservoir simulation

The following function *Reservoir physics* is responsible for simulating the time-delayed active particle oscillation.

```
function Reservoir_physics(Oscillator_number::Int64,Input::Float64)

        local dim = Oscillator_number
        time = Int64(timematrix[Oscillator_number]) #local - Oscillator time

        θ[dim, time] = ϕ[dim, time] - ϕ[dim, time - delay]
        ϕ[dim, time] = ϕ[dim, time] + dt*((v_0*sin(θ[dim, time]+ input)/ρ[dim, time])) + rand(Normal(0.0,sqrt(2*dt*diffusivity)))/ρ[dim, time]
        #markov process - diffusivity

        hold = ρ[dim, time] - (v_0 *cos(θ[dim, time]+input)*dt) + rand(Normal(0.0,sqrt(2*dt*diffusivity)))

        ρ[dim, time] = max(hold, ρ_0) #max radius limit

        #every  "+1" is a "+dt" discrete coordinate transformed
        timevector[Osc_Number] = time + 1

        return θ[dim, time].*(180 /pi)
end
```

The function is called upon every discrete time to calculate the response of an input on a particular oscillator. Since the program allows the flexibility of having multiple oscillators, similar to the lab setup, input response for different oscillators is calculated using different time memory and past variable index. Variables $\phi, \theta \& \rho$ are initialized as vectors and global time as a matrix to keep time memories of different oscillators. The equations inside the function follow the equation 31 of a noisy reservoir setup. However, the equations can also be modified to equation 29 for an ideal non-noisy reservoir setup.

3.3.2 Building the state space matrix

The following function *State assembly* calculates each state vector x and stacks them together over training length to build the state space matrix X.

```
function State_assembly()

        X = zeros((realNeurons*M_out)+inputSize+1, trainLen-initLen)

        memory = zeros(trainLen+testLen+1,realNeurons,(M_out - M_in))

        for i = 1:trainLen
                #builds one state 'x' in one loop
                x= []
                data_collect = []

                        for lop = 1:(d_collect_num+1)
                                if lop==1
                                        push!(data_collect,1)
                                else
                                        push!(data_collect,data[i]*input_multiply)
                                end
                        end

                for j = 1:realNeurons
                        #for each real state
                        for k = 1:M_in   #Multiplexing
                                if k == 1
                                        for idx = 1:(M_out - M_in)
                                                push!(x,memory[(i+1)-1,j,idx])
                                        end
                                end
                                #reservoir output
                                holdon = Reservoir_physics(j, collect(transpose(data_collect)*Win_again[j,k,:])[1])

                                push!(x, holdon)

                                if k == M_in
                                        for idx = 1:(M_out-M_in)
                                                memory[(i+1),j,idx] = x[end-(M_out-M_in)+idx-1]
                                        end
                                end
                        end
                end
                #stacking the state vectors into a matrix
```

```
                if i > initLen
                        X[:,i-initLen] = [(data_collect);x]

                        #X[:,i-initLen] =[1;data[1,i];data[2,i];data[3,i];x]
                end

        end
        return X
end
```

The function performs the time multiplexing operation as described in section 3.2.3, after which it calculates the evolution of each oscillator using the earlier function *Reservoir physics*, as per section 3.2.2, stacking them together to form a single state x. Each of these is again stacked through the length of training to make the state space matrix, similar to the visual representation in Figure 14. Additionally, the function trims some of the obtained reservoir activations states at the beginning of training with the variable;

```
        initLen     #initial length
```

This is simply to remove the initial transient response of the oscillator and just keep the input-driven steady-state response which relates to the nonlinear expansion of the input.

3.3.3 Regression

The following function *Reg regression* calculates the weight vector W_{out} through reg-regression as explained in Section 2.3.4. Since the program is written in Julia, the function can utilize Julia's backslash solver.

```
function Reg_regression()
        X = State_assembly() #state matrix
        Yt = transpose(data[initLen+2:trainLen+1]) #traing data

        reg = 1.1e-8  # regularization factor

        #Reg-regression using Julia's \ solver
        Wout = transpose((X*transpose(X) + reg*I) \ (X*transpose(Yt)))

        return Wout
end
```

The **back slash solver** uses the QR factorization method and is a numerically stable and reliable method for the speed relative to other available methods. The solver simply computes $W_{out} = \left(X\ X^T\right)^{-1}\ X^T\ Y_t$ to minimize the least square error $min_x\ ||X\ W_{out}\ - Y_t|\ |_2$.

3.3.4 Testing and prediction

The function, *Time series prediction* below finally uses the built state space matrix and trained weights W_{out} to run predictions as explained in Section 2.3.4, using the equation 26.

```
function Timeseries_prediction()

        Wout = Reg_regression()
        global u = data[trainLen+1]

        Y_out = Float64[]


        for t = 1:testLen    #test time
                x = []
                data_collect = []
                for lop = 1:(d_collect_num+1)
                        if lop==1
                                push!(data_collect,1)
                        else
                                push!(data_collect, u*input_multiply)
```

```
                end
            end


            #calculating activation states for test-data
            #similar to activation state code in State_assembly()

            for j = 1:realNeurons
                for k = 1:M_in
                    if k == 1
                        for idx = 1:(M_out - M_in)
                            push!(x,memory[(trainLen+1+t)-1,j,idx])
                        end
                    end

                    holdon = Reservoir_physics(j, collect(transpose(data_collect)*Win_again[j,k,:])[1])

                    push!(x, holdon)

                    if k == M_in
                        for idx = 1:(M_out-M_in)
                            memory[trainLen+1+t,j,idx] = x[end-(M_out-M_in)+idx-1]
                        end
                    end
                end
            end

            #final prediction calculation
            y = Wout*[(data_collect);x]
            Y_out[t] = y


            global u = y                    #generative mode
            #global u = data[trainLen+1+t] #Predictive mode

        end
        return Y_out
end
```

Additionally, an important detail is the ability to switch between the 'Generative' and the 'Predictive' modes. The following code snippet runs the Prediction() function in the generative mode.

```
        global u = y                #generative mode
```

This is achieved by simply feeding the network's own prediction $y_{out}(n)$ into the activation state calculation $x(n+1)$ in the reservoir and using $y_{out}(n+1) = W_{out} \cdot x(n+1)$. Hence, making it a time series generative network. This also makes any prediction deviation or error accumulate over prediction time and is a much harder method to get right, as explained in Section 2.3.4. And finally, the code snippet below runs the network in a predictive mode.

```
        global u = data[trainLen+1+t]               #Predictive mode
```

This is simply done by feeding the current step data into the activation state calculation, $x(n+1)$, which then translates into further prediction calculation. This method is relatively easier to get the right results since any error in prediction has no cumulative accumulation.

4 Results and comparative analysis

The time series prediction results analyzed in this chapter were generated using the generative mode of the network, as explained in Section 3.3.4 and 2.3.4. Additionally, Mackey–Glass chaotic time series signal is used as the signal of choice for most comparative studies. This part of the chapter briefly discusses the Mackey–Glass signal, the root mean square error, and the results from the 'Predictive mode' of the network.

The Mackey-Glass signal is a chaotic time series signal, obtained from the Mackey–Glass delayed-differential equation, written as follows:

$$\frac{dy}{dt} = \beta \cdot y\left(t\right) + \frac{\alpha \cdot y\left(t-\tau\right)}{1 + x\left(t-\tau\right)}$$

Where 't' is time, 'τ' is the time delay, and 'y' is a unit-less amplitude [Mackey and Glass (1977)]. The time series has chaotic behavior when the time step delay is at least 17 steps, $\tau \geq 17$. The chaotic nature of the time series signal with its dependence on the past states makes it an interesting signal to analyze the effects of noise in reservoir computing.

Root-Mean-Square-Error (RMSE) [Wang and Lu (2018)] is the square root of the Means squared error. RMSE is a common method of analyzing errors in time series prediction and is extensively used in this chapter.

$$RMSE = \sqrt{\frac{\sum_{n=1}^{N}\left(Y_i^{\,True} - Y_i^{\,Predicted}\right)}{N}} \tag{32}$$

Predictive mode , as described in Section 3.3.4 and 2.3.4, is used to make time-series predictions of the Lorentz signals.

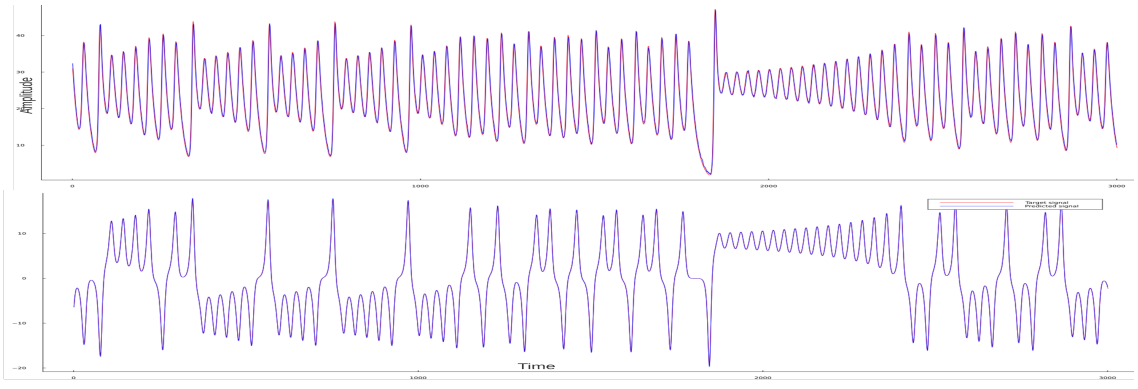

Figure 16: 3,000 step prediction of the 1st and 2nd dimensional Lorentz system in the predictive mode

Figure 16 is a plot of the prediction for the famous Lorentz system. It can be seen that the 'predictive mode' generates an almost perfect prediction. However, discrepancies in the prediction and data can be seen more clearly in the plot for the Lorentz attractor in

Figure 17. Results for the 3-dimensional signal are obtained by coupling the 3-dimensional input in the state space matrix using the reservoir responses. Figure 17 also does a comparison between prediction from a noiseless and a noisy reservoir, the added noise is along the equation 31, diffusion $D = 0.08$, and the attractors look very alike with very little difference. The predictions can be further enhanced or analyzed with various parameter adjustments. The rather successful prediction in part has to do with the minimal task undertaken by the network in 'predictive mode' since the network only generates prediction one step ahead at any time step and consequently is not a good representation of the learned input in the network or the network itself. Generative mode on the other hand forces the network to make an independent signal prediction after training without any reliance on current step data during the testing-prediction phase. This allows for a better understanding of the internal activation in the network achieved during training, further expanding opportunities for experimental methods to obtain a longer and better time series prediction generation.

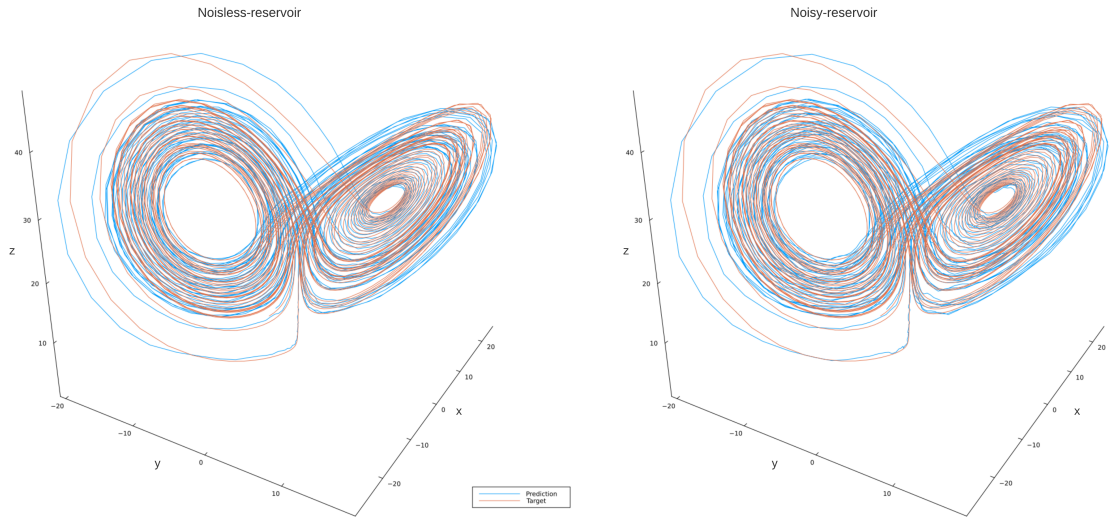

Figure 17: Comparing Lorentz attractor predictions for reservoir with and without noise, in the prediction mode

4.1 Contrasting noisy & noiseless reservoirs

4.1.1 Results from a noiseless reservoir

Figure 18 is generated using a noiseless reservoir, following the equation 29 for reservoir state activations. The parameters used to obtain the predictions are 35 real oscillators, multiplex = 10, training length = 2000, and testing length = 2000. With an RMSE value along the training length of $RMSE = 0.08967$, and the means square error (MSE) of the 1st 1000 step of $MSW[1000] = 6.77092e - 5$.

The obtained prediction in Figure 18 is a good prediction, especially in 1st 1000 steps as is evident with target-prediction difference graphs in Figure 19.

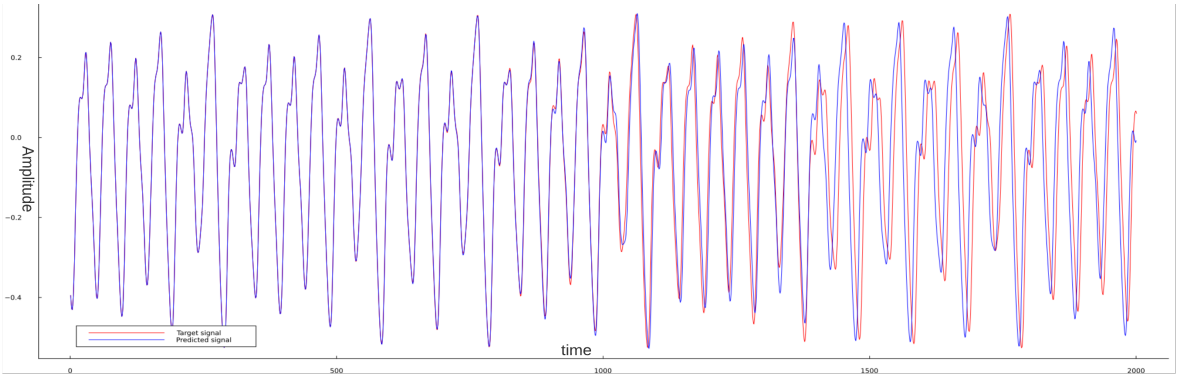

Figure 18: Noiseless reservoir, Mackey-Glass prediction

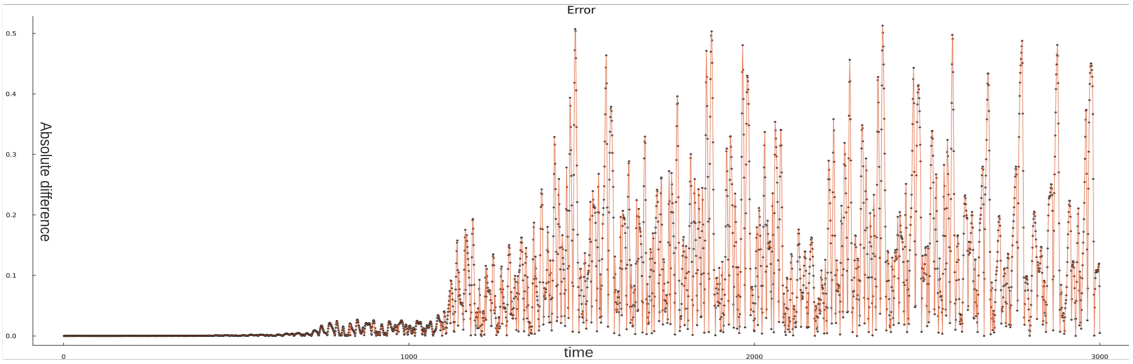

Figure 19: Absolute difference between target data and prediction, for figure 18

4.1.2 Results from a noisy reservoir

The noisy reservoir follows the Equations 31 for the reservoir activation states. With a noisy reservoir, the steady state input-driven response from the reservoir becomes noisy as seen in Figure 6 from the theoretical chapter, making the reservoir state activations noisy and ultimately adding noise to the state space matrix.

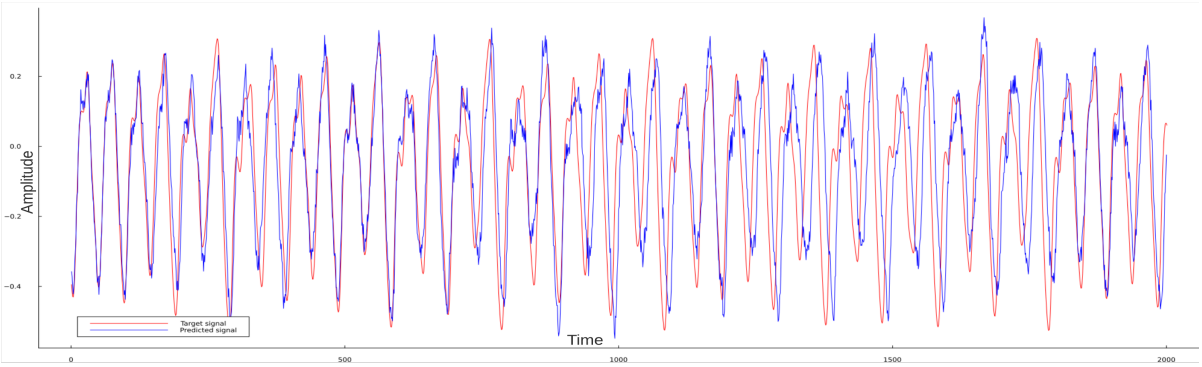

Figure 20: Generated predictions with a noisy reservoir

Additionally, the prediction results obtained in Figure 20 were obtained after extensive

simulation attempts and parameter searches, the important parameters of which are described in section 4.4. Major parameters, changed in comparison to predictions in Figure 18, used for obtaining results in Figure 20 are $M_{out} = 500$, $M_{in} = 3$, an input scaling factor of 11 and 35 real oscillators.

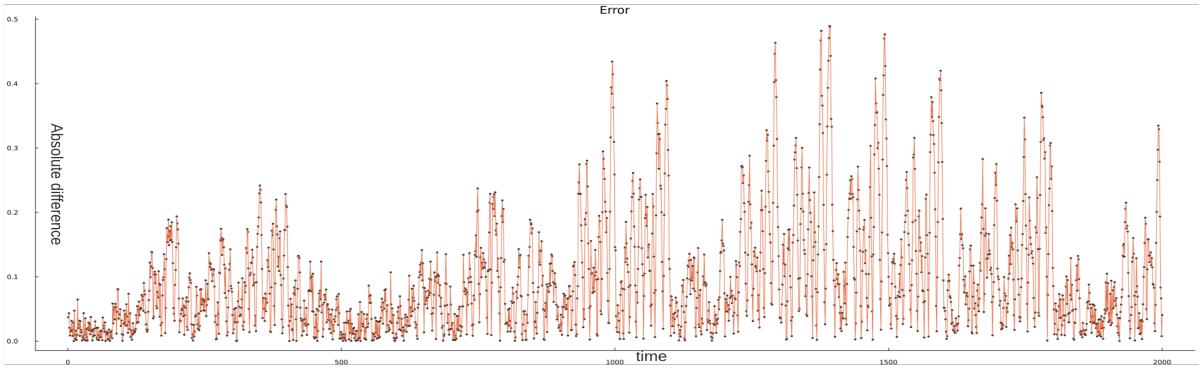

Figure 21: Absolute difference between target data and prediction, for Figure 20

When comparing Figure 21 and Figure 19 it might seem the errors produced in a noisy reservoir are rather smaller than those in the noiseless reservoir, this is not true as is clearly evident by the prediction trajectories in figure 20. This happens due to the fair amount of positive and negative noise deviations cancelling themselves out in an absolute value graph and hence the extensive use of RMSE as the more accurate picture of the error. Additionally, when comparing the graphs in Figure 21 and Figure 19 one can also notice that the prediction to target divergences does not happen in a significant way until the 1000 time step mark in Figure 19 of the noiseless reservoir, while the signal to target divergence can be noticed much earlier in Figure 21 around the 200 step mark.

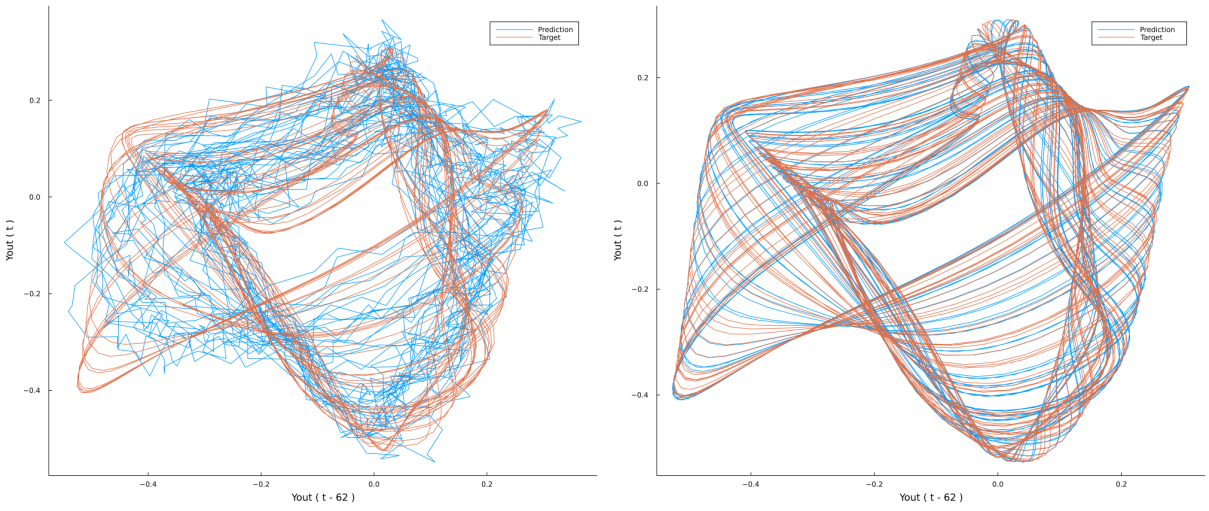

Figure 22: Comparing attractors, of the generated predictions, from Noisy and Noiseless reservoir

Figure 22, does a comparison between attractors produced by the prediction from a noisy reservoir with that of a noiseless reservoir. The attractor is produced by plotting the

predicted signal with 62 steps delayed version of the same and the same is done with the test data for comparison. Given the bounded nature of an attractor, the visual can aid greatly in understanding the long-term evolution of the time series signal without having to go through a very long time series plot [Doyne Farmer (1982)]. As can be seen in the plot on the left in Figure 22, the prediction does retain a great deal of the long-term evolution of the system, but with significant noise.

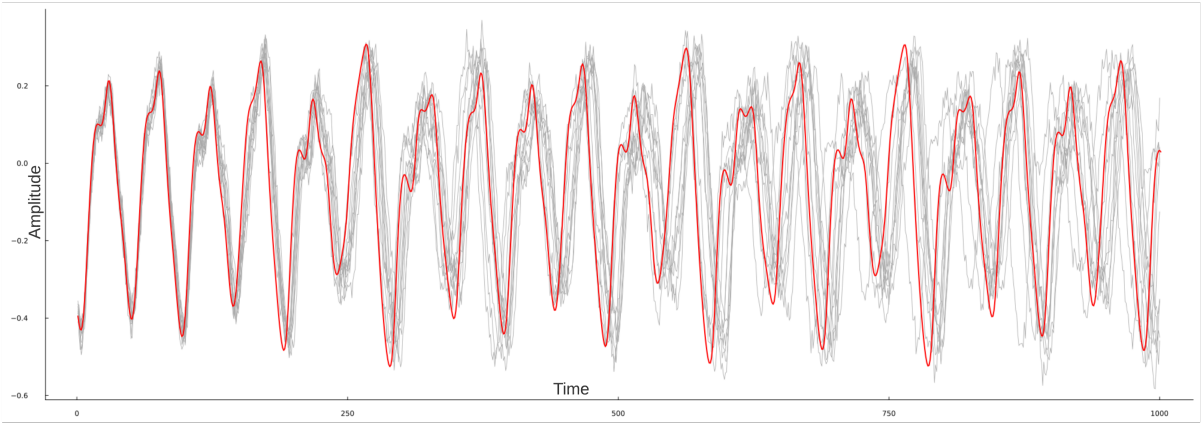

Figure 23: Multiple (11) simulation predictions [Grey], compared with target signal

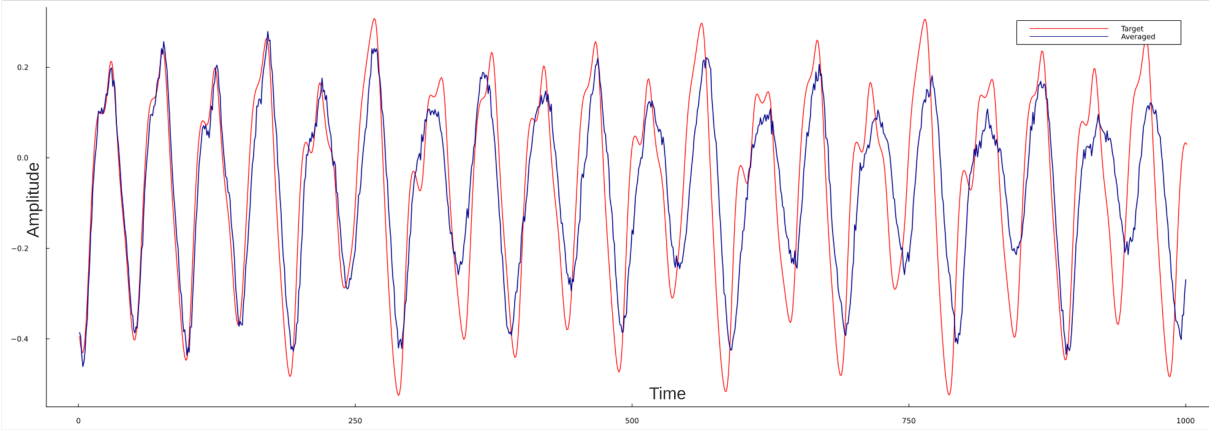

Figure 24: Average of multiple simulations compared with the target signal

Figure 23 and Figure 24 are results obtained after 11 simulation runs. The prediction divergence from the target can be seen more clearly in the averaged prediction in Figure 24. While prediction is fairly good until around time step 200, there is phasing out between the prediction and target signal. This prediction phasing out is a common observation in RC predictions as can also be seen in Figure 18. However, the prediction performed using a noisy reservoir, phases out from the target signal faster than the noiseless reservoir, and this phase difference grows over time steps.

The growth in phase difference between the target and the predicted signal might potentially be due to a lower dependence of the time series signal over the increasing past

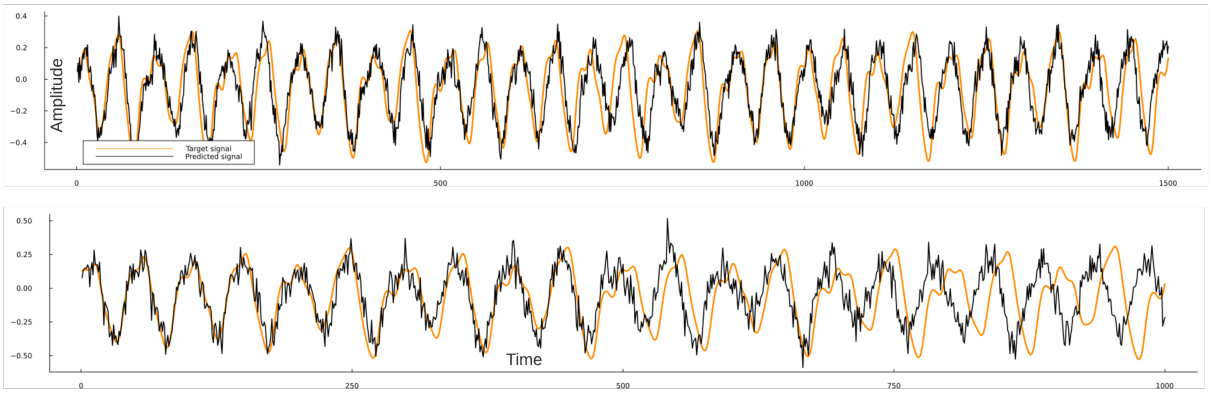

Figure 25: 3000 step prediction with a noisy reservoir on an online mode

time. Since the Mackey-Glass signal itself is a time-delayed signal the dependence of any current value in the signal might have a decreasing or fading relation to the past states of the signal after the time-delayed point in the past. This problem of the signal's fading relation to its past can be resolved for purpose of a better long-term prediction by updating the reservoir after the prediction with new past data. This is called an *online network* or the online method.

Figure 25 is an implementation of the online method with a noisy reservoir, where the state space matrix is updated every 200-time steps with the reservoir activation from the past data during the testing phase. However, the good-looking performance in the Figure is a little misleading since there are differences in performance at different parts of the Mackey-Glass signal and since the online mode does not allow for any changes in parameters during a long prediction, this leads to bad predictions in parts of the signal that are not shown in the figure.

4.2 Noise to performance relation

To better understand the effects of noise on prediction, RMSE values were collected from each simulation, a total simulation run of 200 times with varied variances of noise in the reservoir. Every other parameter was fixed as in the prediction obtained in Figure 20. The First 100 simulations were run between $d\sigma \in [0.001, 0.1]$ and the next 100 between $d\sigma \in [0.1, 1.0]$, the results of which are shown in Figure 26.

The obtained data is then fitted with a polynomial curve fit to make sense of the pattern in the data. The system has a linear growth in error with increased variance between $d\sigma \in [0.001, 0.1]$, as can be seen in the plot on the left in Figure 26. Additionally, the system has a cubic growth for a larger variance of $d\sigma \in [0.1, 1.0]$ and also seems to have a growing variability in performance with growing variance $d\sigma$ in noise, as can be seen in the plot on the right in Figure 26.

These relations can be further studied with larger computing power for various parameters, especially the ones mentioned in section 4.4. Since predictions do seem to show improvements with certain parameter adjustments, these performance observations sometimes change due to the behavior of the noise in the system adding uncertainty to the viability of the chosen parameter combination. Hence thorough parameter sweep with better computing could give a better understanding of the effect of noise in the reservoir on the prediction performance. Ultimately, reducing the noise in the reservoir itself can benefit significantly towards better performance, and the Brownian noise experienced in a real physical reservoir could be reduced by reducing the temperature of the oscillator environment. Though this is a much tougher solution to implement.

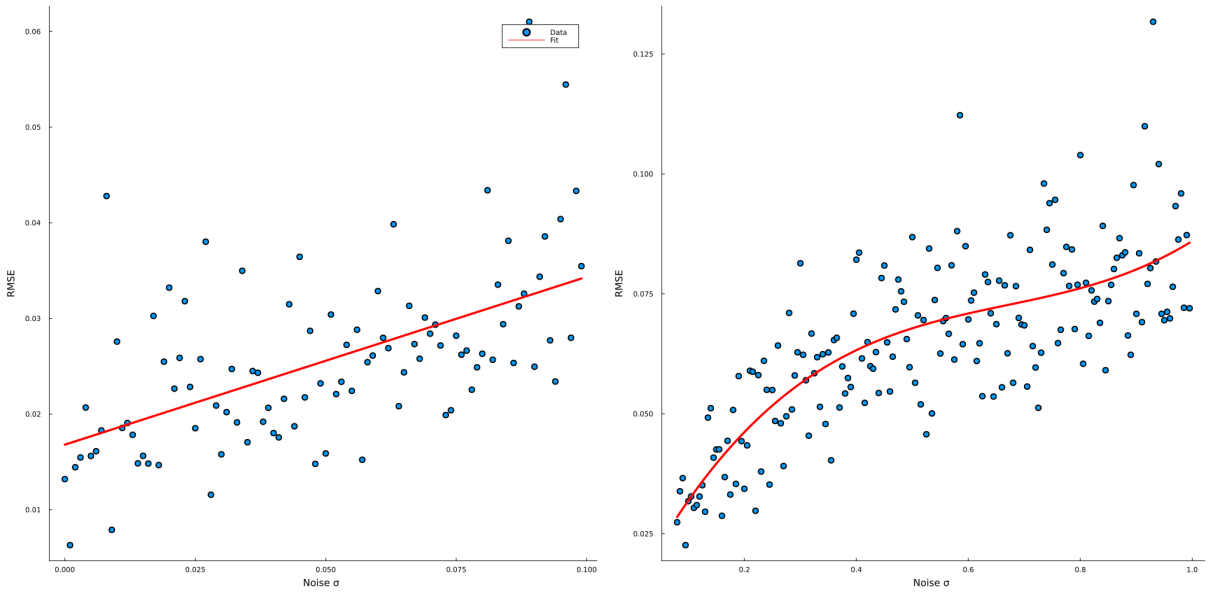

Figure 26: RMSE plot with 100 simulations between $d\sigma \in [0.001,\ 0.1]$ [left] & 100 simulations between $d\sigma \in [0.001,\ 0.1]$

4.3 Predictive performance analysis of the system with different signals

In an effort to understand the system, various signal predictions were attempted. The noiseless reservoir system generally does a great job at predicting simple signals such as the damped oscillation in Figure 27. Whereas the system does a terrible job of predicting a stochastic signal such as the one in Figure 28. This might simply be due to the stochastic signals having a past independent behavior at any time step, making it a non-ideal type of signal to predict using this system which exclusively exploits past dependence of states in a signal.

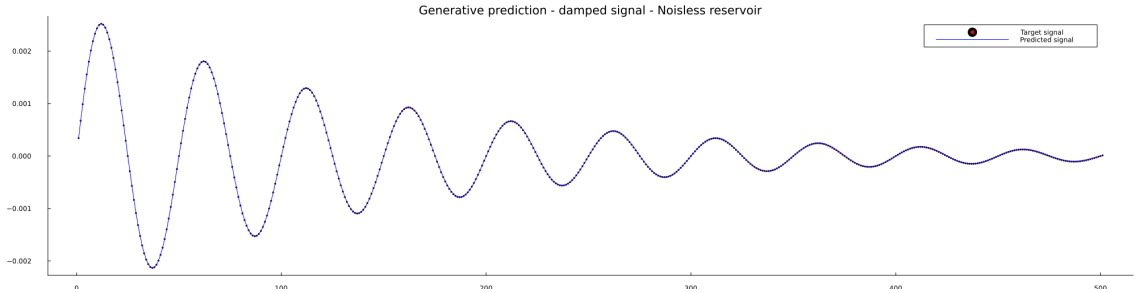

Figure 27: Damped oscillation prediction

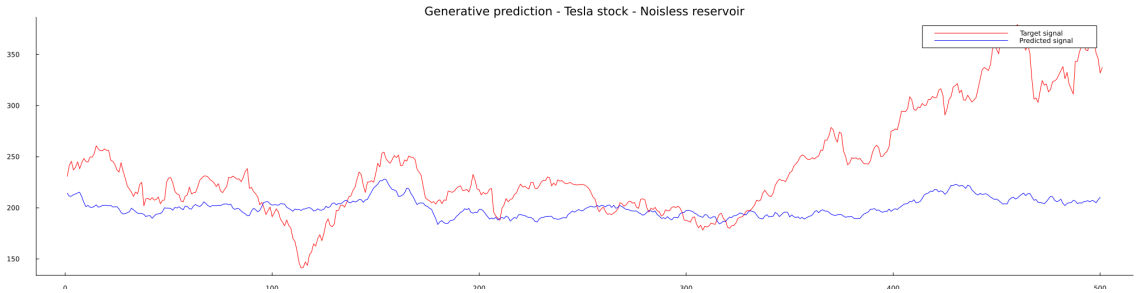

Figure 28: Unsuccessful attempt in prediction Tesla stock prices

4.4 Influential parameters for noise mitigation

This section discusses the two main parameters observed to make a significant difference in prediction performance improvements with a noisy reservoir.

4.4.1 Effect of time multiplexing on performance with a noisy reservoir

The method of Time multiplexing $M_{out} \geq M_{in}$ as discussed in section 3.2.3 makes a significant difference in the prediction performance of the network. The parameter used in generating the predictions in Figure 20 and Figure 25 was of $M_{out} = 500$ and $M_{in} = 3$. In general, a small M_{in} and a large M_{out} tended to give a good performance, as observed through plots and RMSE values.

This observed effect is very likely due to the structure multiplexing, where a large value of M_{out} stacks a large number of activations from the past oscillator states into the current state vector at any moment in testing. This might allow some dissolution of noise through the presence of a large number of past activations combined with a smaller number of current activations. Additionally, this also increases the temporal memory of the network as discussed in section 3.1.2. A larger temporal memory might be an important factor in mitigating the effect of a noisy reservoir.

4.4.2 Effects of scaling input on noise

One of the interesting effects observed while working with the system was with input scaling i.e., by increasing the amplitude of the input, there was an increase in prediction performance with the noisy reservoir. This idea of input scaling was also mentioned in the original Echo-state-network paper [Jaeger (2001)] and additionally in [Lukoševičius and Jaeger (2009b)]. However, this amplification has a limit in this system, since the input-induced angle in the time-delayed active particle oscillation system is physically constrained to repetition, with an angle of at most 90-180 degrees.

A simple mathematical description of this can be written as,

$$W_{in} \cdot \begin{pmatrix} 1 \\ \Lambda \cdot \mu\,(n) \end{pmatrix}$$

where Λ is the amplitude or rather the added positive multiplication factor. Conversely, this could also be done by increasing the variance of random number prediction within W_{in}. This effect could be better implemented by the method of input signal normalization. Every signal, no matter the amplitude, can be normalized with an amplitude that performs the best within the given physical reservoir. Otherwise, the amplitude itself can be used as an additional parameter to achieve the best performance by searching the optimal point of activation.

The effectiveness of input scaling is very likely due to its effect on the nonlinear introduced by the reservoir. Different input scaling can affect the nonlinearity of the reservoir response. The input response of a scaled signal in a time-delayed active particle oscillation can produce a richer nonlinear response than otherwise. This response can then be tailored based on the type of time series signal the system tries to predict: for some signals a more nonlinear response might not be good for training predictions. However, with a noisy reservoir, a large input scaling between 9-21 was effective in reducing some noise and improving predictions. Additionally, as the Brownian noise remains relatively intrinsic to the reservoir setup, the amplification of the input signal can be optimized for better nonlinear activation response, which might otherwise be drowned out by the Brownian noise.

There is also an observed interaction between the input scaling and the multiplexing method. Previously in the case of a noisy reservoir, increasing the size of the state vector or the state matrix using much larger values for 'Multiplex-in' and 'Multiplex-out' did

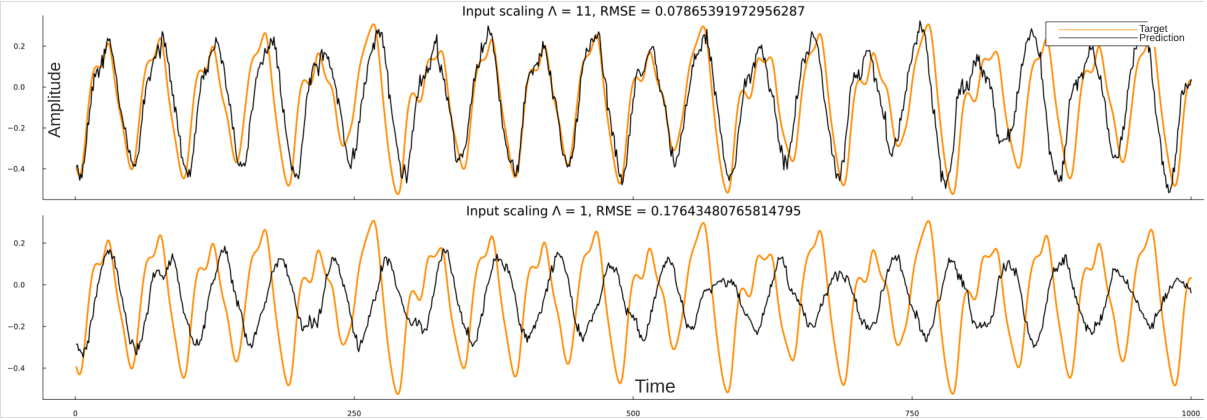

Figure 29: Prediction comparison with input scaling of 11 and just 1, with first 100 step RMSE

not yield any significant performance benefits. This was contrary to the case of a normal Echo state network, with which a larger state matrix generally led to increased performance. However, with the use of λ as an additional parameter, there was a noticeable increase in performance with the increase in state space matrix size through large values of 'Multiplex out'.

4.5 Interpretable machine learning

Machine Learning methods and Recurrent Neural Networks are often not explainable. The success of the network predictions is often analytically extremely hard to derive or sometimes just plain indecipherable. This hinders an educated exploration or adds uncertainty to the safety of real-world deployments of these methods, regardless of their amazing success [Molnar et al. (2020)].

The flavor of non-linearity chosen for input expansion makes a huge difference in the effectiveness of the predictions as described in section 3.1.1 [Shalizi (2020)]. Since the normal echo-state network uses a random state coupling matrix as its reservoir, this allows more flexibility in flavors of nonlinearity, which can be effective for various signal predictions. It must also be noted though, that this also makes it much harder to backtrack or reason a successful prediction.

The RC computing system is one the simplest methods of Recurrent Neural Network, with only one matrix (state collection matrix) to optimize the training data set. The ability RNNs to couple past state activations with current state activation makes it a great method for training time series signals that are dependent on their past values. The time-delayed oscillation active particle reservoir system in this project uses a time-delayed oscillator to project a time-delayed signal (The Mackey-Glass signal) into a multidimensional feature space. Exploiting this emergent linearity, weights for predictions are trained using linear reg regression to predict a nonlinear signal. Making this system a much more

interpretable method of machine learning, where the dynamical system behavior of the reservoir can be used to make reasonable conclusions on the effective predictions and errors.

4.6 Conclusion

This project simulates a collection of active particle time-delayed oscillators to produce a collection of nonlinear activations, which are then trained with a linear reg regression method to produce weights or trained coefficients to predict a time series signal. Which is then contrasted with the expected test output time series signal. Further the same is repeated with a simulation of active particle time-delayed oscillation but this time with induced Brownian motion affecting such oscillations.

The dynamics of the reservoir follow a form of the delayed differential equation through the input-driven active particle oscillations. The choice of nonlinearity through time-delayed *Sine* activation, for the reservoir, might make this system uniquely positioned to make good predictions on a *delayed time series signal* such as the Mackey-Glass signal or signals from the delayed differentials such as the signal out of Ikeda delay differential equation. And, given the intrinsic nature of Reservoir computing and RNNS in general, the system is only well positioned to predict signals which have some dependence on the past states and would perform rather poorly on signals resembling stochastic processes. Additionally, for signals with very large time-delayed dependencies, Echo-state networks would need large training and memory parameters, i.e very large state space collection matrix to gain enough information to identify motifs for predictions.

The predictions performed by a reservoir network activation, with Brownian motion, are generally more erroneous than the predictions from the ideal noiseless reservoir activation. This might be due to the effect of noise in hampering the quality of network memory and reducing the effectiveness of the chosen nonlinearity for input expansion. And hence measures should be taken to try and reduce the effect of noise or reduce the noise itself. In this project, two methods of effective noise mitigation were found and explored. The method of time multiplexing through $M_{out} \geq M_{in}$, a large value of multiplex-out and a small value of multiplex-in was effective in producing better prediction results from a noisy reservoir. Additionally, maximum input scaling to get the best non-linear activation out of the reservoir was also effective in yielding better predictions out of a noisy reservoir.

A Appendix

References

Approximation of dynamical systems by continuous time recurrent neural networks (1993). *Neural Netw.* **6**(6): 801–806.

Bezanson, J., Karpinski, S., Shah, V. B. and Edelman, A. (2012). Julia: A fast dynamic language for technical computing.

Bianchi, F., Livi, L., Alippi, C. (2017). Multiplex visibility graphs to investigate recurrent neural network dynamics.

Doyne Farmer, J. (1982). Chaotic attractors of an infinite-dimensional dynamical system, *Physica D* **4**(3): 366–393.

Einstein, A. and Cowper, A. D. (n.d.). ALBERT EINSTEIN, PH.D, `https://www.maths.usyd.edu.au/u/UG/SM/MATH3075/r/Einstein_1905.pdf`. Accessed: 2022-11-23.

Jaeger, H. (2001). The" echo state" approach to analysing and training recurrent neural networks-with an erratum note', **148**.

Kokot, G., Faizi, H. A., Pradillo, G. E., Snezhko, A. and Vlahovska, P. M. (2022). Spontaneous self-propulsion and nonequilibrium shape fluctuations of a droplet enclosing active particles, *Communications Physics* **5**(1): 1–7.

Lenzen, F. and Scherzer, O. (2004). Tikhonov type regularization methods: History and recent progress, *Proceeding Eccomas* **2004**.

Lukoševičius, M. (2012). A practical guide to applying echo state networks, *Lecture Notes in Computer Science*, Lecture notes in computer science, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 659–686.

Lukoševičius, M. and Jaeger, H. (2009a). Reservoir computing approaches to recurrent neural network training, *Computer Science Review* **3**(3): 127–149.

Lukoševičius, M. and Jaeger, H. (2009b). Reservoir computing approaches to recurrent neural network training, *Computer Science Review* **3**(3): 127–149.

Mackey, M. C. and Glass, L. (1977). Oscillation and chaos in physiological control systems, *Science* **197**(4300): 287–289.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.* **5**(4): 115–133.

Molnar, C., Casalicchio, G. and Bischl, B. (2020). Interpretable machine learning – a brief history, State-of-the-Art and challenges.

Pavliotis, G. A. (2014). The langevin equation, *in* G. A. Pavliotis (ed.), *Stochastic Processes and Applications: Diffusion Processes, the Fokker-Planck and Langevin Equations*, Springer New York, New York, NY, pp. 181–233.

Röhm, A. and Lüdge, K. (2018). Multiplexed networks: reservoir computing with virtual and real nodes, *J. Phys. Commun.* **2**(8): 085007.

Shalizi, C. (2020). Extending linear classifiers with nonlinear features, 36-462/662, Spring.

Sherstinsky, A. (2018). Fundamentals of recurrent neural network (RNN) and long Short-Term memory (LSTM) network.

Sjögren, L. (n.d.). Stochastic processes lecture notes ch. 6: Brownian motion: Langevin equation.

Szabados, T. (2010). An elementary introduction to the wiener process and stochastic integrals.

Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D. and Hirose, A. (2019). Recent advances in physical reservoir computing: A review, *Neural Networks* **115**: 100–123.
URL: *https://www.sciencedirect.com/science/article/pii/S0893608019300784*

Terpstra, C. R. (2016). Delay differential equations, *University of Groningen* .

The Editors of Encyclopedia Britannica (2022). *Brownian motion*.

Wang, W. and Lu, Y. (2018). Analysis of the mean absolute error (MAE) and the root mean square error (RMSE) in assessing rounding model, *IOP Conf. Ser.: Mater. Sci. Eng.* **324**(1): 012049.

Wang, X., Chen, P.-C., Kroy, K., Holubec, V. and Cichos, F. (2022). Spontaneous vortex formation by microswimmers with retarded attractions.