SYDE 556/750
Simulating Neurobiological Systems
Lecture 6: Recurrent Dynamics

Andreas Stöckel

Based on lecture notes by
Chris Eliasmith and Terrence C. Stewart

February 4 & 6 & 11, 2020

Accompanying Readings: Chapter 8 of Neural Engineering

Contents

1 Introduction

Note: We have now discussed the first two principles of the Neural Engineering Framework: Representation and Transformation. These two principles allow us to build feed-forward networks, i.e., networks that can – mathematically speaking – be described as acyclic directed graphs. In this lecture we will address the final principle: *dynamics*.

Biological neural networks are dynamical systems – that is, they perform computation over time; the past state of the network influences present computations. So far, we discussed two sources of dynamics: neuron models and synaptic filters. The types of dynamics exhibited by these two sub-systems are usually relatively short-lived; the system "forgets" its previous state quite quickly. In turn, this means that each population in a pure feed-forward network will eventually "forget" events that happened more than a fraction of a second ago.

Example: *Short-term dynamics of neurons and synaptic filters.* LIF neurons "forget" their history whenever they emit a spike – causing them to be reset to their initial state. Furthermore, synaptic time constants are usually in a range between 1 ms to 100 ms.

This is of course not what we observe in biological systems. Most animals appear to have memories of events in the past. These "memories" span time-frames of seconds to years. In theory, there are three ways to implement such "memory":

- **Complex neuron models.** Biological neurons and realistic neuron models typically posses more complex dynamics, such as *firing rate adaptation*. However, the effects of firing rate adaptation typically last for a few seconds only and are thus not sufficient to explain "memory" in biological systems.
- **Changing connection weights over time.** Another form of "memory" can be implemented by changing synaptic weights over time. In the context of the NEF we refer to this process as "learning".
- **Recurrent connections.** Introducing "backwards" (or "recurrent") connections into a network could allow us to "remind" the network about it's previous state, preventing the network from "forgetting".

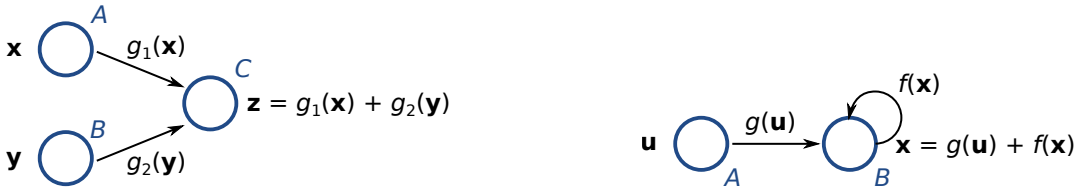
Both learning and recurrent connections are valid solutions to the problem of neural networks "forgetting" the history of their input. Typically, synaptic weight changes are regarded as evoking longer term changes (minutes to years), whereas recurrent connections are a model of shorter term changes in dynamics (seconds to minutes). In this lecture we focus on recurrence, giving rise to the third principle of the Neural Engineering Framework.

NEF Principle 3 – Dynamics
Neural dynamics are characterized by considering neural representations as control theoretic state variables. We can use control theory (and dynamical systems theory) to analyse and construct these systems.

2 Exploring Recurrent Connections

Before we take a deeper dive into the mathematics of neural dynamics, we first discuss how we can implement recurrent connections in the context of an NEF network, followed by some experiments in which we explore what happens for certain functions that are being computed in the feedback function.

2.1 Implementing Recurrent Connections



(a) Feed forward network **(b)** Recurrent network

Figure 1: Comparing a feed-forward and a recurrent neural network.

Note: When we talk about recurrent connections in the context of the Neural Engineering Framework, we typically refer to a single neuron population that is connected back to itself. While this might seem a little strange, this section is meant to point out that recurrent connections change virtually nothing in terms of the equations that we have used so far.

Revisiting a neural population receiving input from two pre-populations As we discussed in the last lecture, fig. 1a depicts an ensemble C receiving input from two pre-ensembles A and B. Mathematically, each neuron i in the post-population C receives an input current $J_i^C(t)$

$$J_i^C(t) = \langle \mathbf{w}_i^A, (\boldsymbol{\alpha}_{\text{pre}}^A * h)(t) \rangle + \langle \mathbf{w}_i^B, (\boldsymbol{\alpha}_{\text{pre}}^B * h)(t) \rangle + J_i^{\text{bias}}, \quad \text{where } \mathbf{w}_i^A = (\mathbf{E}\mathbf{D}^{g_1})_i \text{ and } \mathbf{w}_i^B = (\mathbf{E}\mathbf{D}^{g_2})_i,$$

where $\boldsymbol{\alpha}_{\text{pre}}^A$ and $\boldsymbol{\alpha}_{\text{pre}}^B$ are the activities of the pre-populations A, B, respectively. The function h is the synaptic filter, $\mathbf{E}$ is a matrix of post-population encoders, $\mathbf{D}^{g_1}$ is decoding the function $g_1(\mathbf{x})$ from population A, $\mathbf{D}^{g_2}$ is decoding the function $g_2(\mathbf{y})$ from the pre-population population B. Then, the value $\mathbf{z}$ represented by the population is approximately $\mathbf{z} = f(\mathbf{x}) + f(\mathbf{y})$.

Recurrent populations We can use the same setup as above to implement the recurrent connection depicted in figure fig. 1b. Here, a population B represents a value $\mathbf{x}$ and receives input from both a pre-population A representing a value $\mathbf{u}$, as well as itself – representing the value $\mathbf{x}$. That is, each neuron i in B receives the current

$$J_i^B(t) = \langle \mathbf{w}_i^A, (\boldsymbol{\alpha}_{\text{pre}}^A * h)(t) \rangle + \langle \mathbf{w}_i^B, (\boldsymbol{\alpha}_{\text{pre}}^B * h)(t) \rangle + J_i^{\text{bias}}, \quad \text{where } \mathbf{w}_i^A = (\mathbf{E}\mathbf{D}^{g_1})_i \text{ and } \mathbf{w}_i^B = (\mathbf{E}\mathbf{D}^{g_2})_i.$$

Notice that apart from names, there is virtually no difference between the previous two equations. The population B approximately represents the value $\mathbf{x} = g(\mathbf{u}) + f(\mathbf{x})$. In terms of control

theory, the variable $\mathbf{u}$ is the *input* to our system, whereas the variable $\mathbf{x}$ is the represented value. The function $g(\mathbf{x})$ is a function transforming our input, the function $f(\mathbf{x})$ is the *feedback function*.

Note: When implementing a computer simulation that contains recurrent connections, each occurrence of a population pre-activity $\mathbf{a}_{\text{pre}}$ should be interpreted as the pre-population activity from the last timestep.

Note that this will introduce a delay of one timestep each time a new set of populations is connected. While not much of an issue in general, this delay is not part of the continuous equations that are being simulated. The implementation of the Neural Engineering Framework in *Nengo* is thus a little more clever and will only introduce delays if it has to; it will not introduce a delay on pure feed-forward connections.

2.2 Experimenting With Recurrent Connections

The above equations illustrate how recurrent connections are implemented and what the represented value of the recurrently connection is. To get some intuition for the dynamics of such a system, we can test different feedback functions $f(\mathbf{x})$. Figure 2 depicts the behaviour of a population of LIF neurons representing a one-dimensional value along with various feedback functions and various synaptic filter time constants.

Note: The synaptic filter $h(t)$ we are using here is the exponential low pass

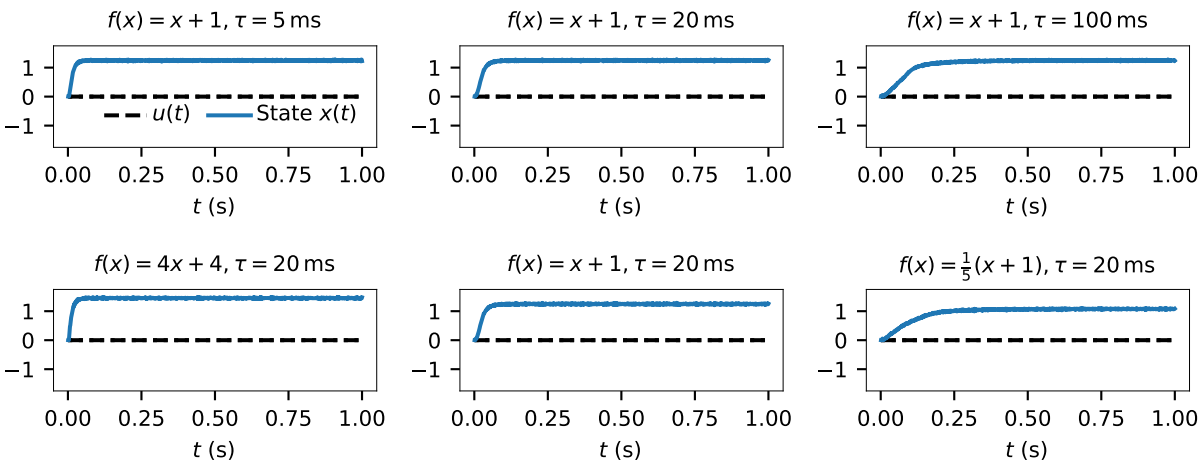
$$h(t) = \begin{cases} \frac{1}{\tau} e^{-t/\tau} & \text{if } t \geq 0\,, \\ 0 & \text{if } t < 0\,. \end{cases} \tag{1}$$

This a version of the general synaptic filter discussed in lecture four (in particular $n = 0$). For this specific case, we can compute the normalisation factor c in closed form. It holds $c = \frac{1}{\tau}$. To see that this factor is correct, we can compute the integral

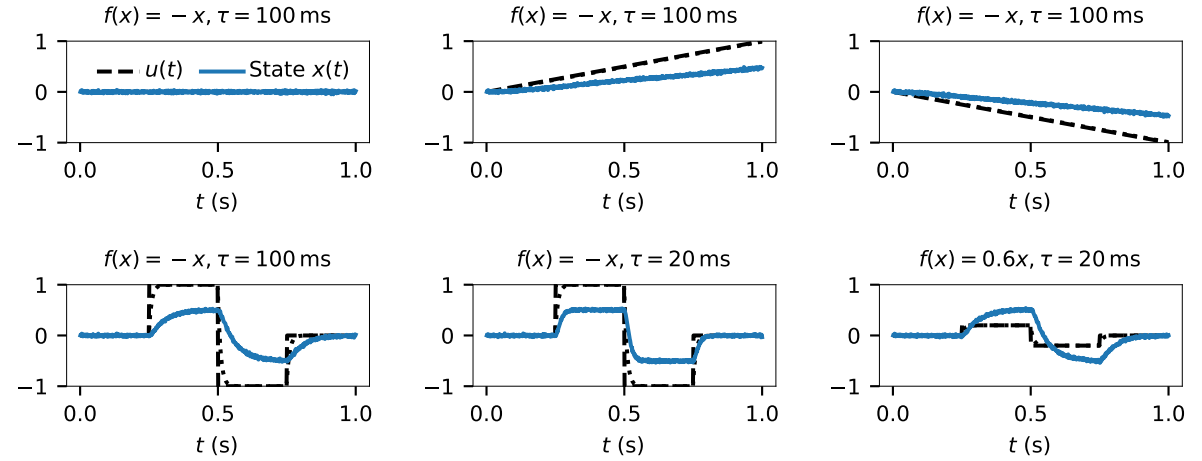
$$\int_0^\infty h(t)\,\mathrm{d}t = \int_0^\infty \frac{1}{\tau} e^{-t/\tau}\,\mathrm{d}t = \frac{1}{\tau}\big[-\tau e^{-t/\tau}\big]_0^\infty = \frac{1}{\tau}\big[0 + \tau\big] = 1\,.$$

Feedback $f(x) = x + 1$ The dynamics of the system for a feedback function $f(x) = x + 1$ are depicted in fig. 2a. The resulting system converges to a value slightly above one. The rate of convergence is determined by the time constant τ and the scaling of the feedback function; an increase of the time constant by a factor c seems to result in similar dynamics as dividing the feedback function f by c.

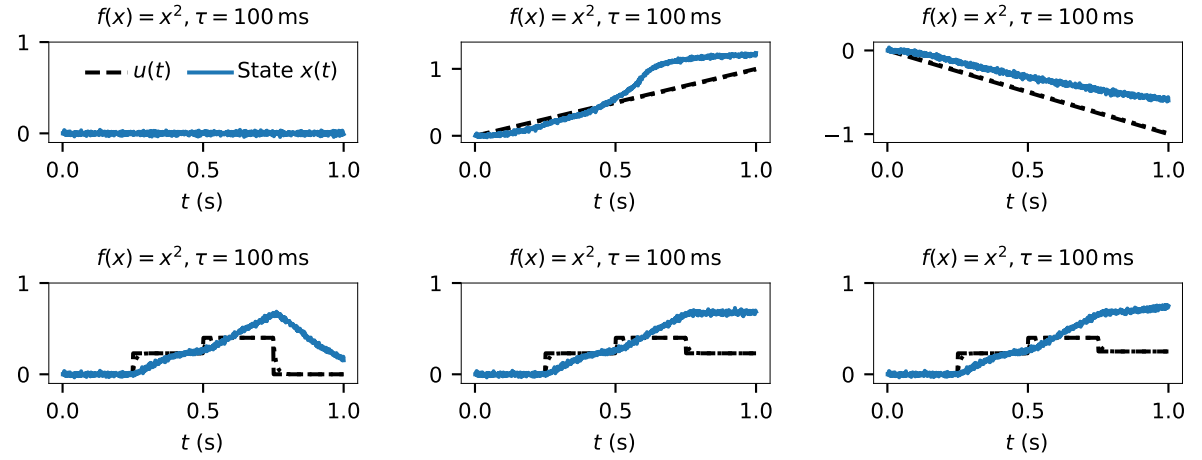
Note: This system only converges to a fixed value due to saturation of the neural responses. For example, if we replace the LIF neurons with rectified linear units, the state x will diverge monotonically towards infinity.



(a) Exploring the feedback function $f(\mathbf{x}) = x + 1$



(b) Exploring the feedback function $f(\mathbf{x}) = -x$



(c) Exploring the feedback function $f(\mathbf{x}) = x^2 s$

Figure 2: Exploring the effect of different feedback functions $f(\mathbf{x})$, input signals $u(t)$, and synaptic time constants τ on the dynamics of a single-population network. 100 LIF neurons, decoded data are filtered with a 10 ms time-constant exponential low-pass. ⌨ *Code*

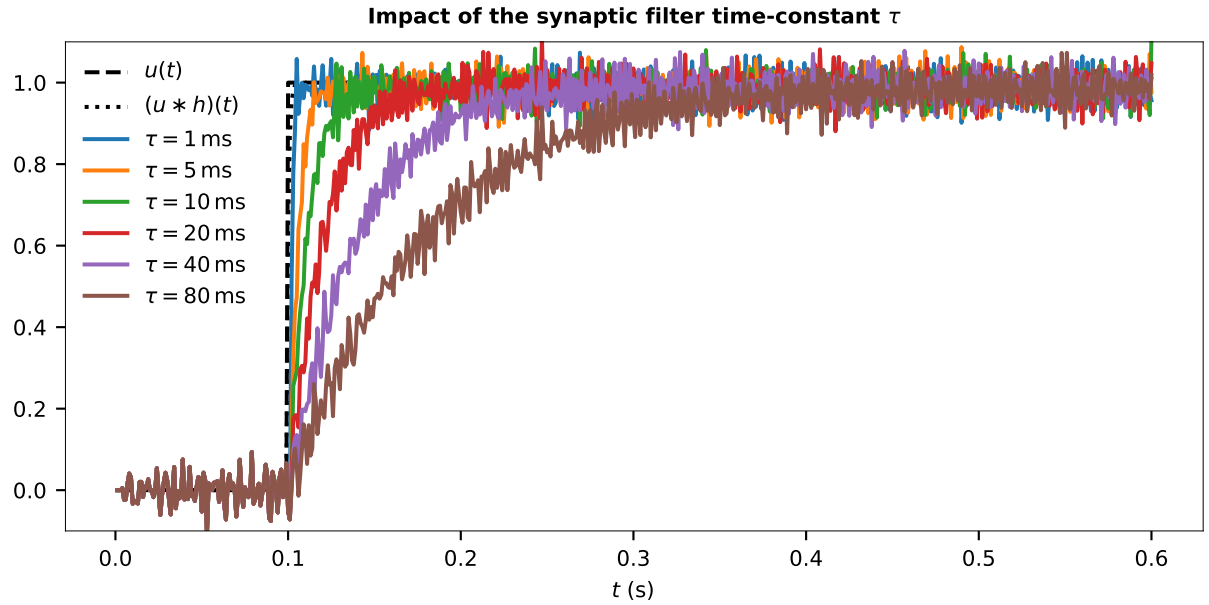


Figure 3: Dynamics of a feed-forward system for varying synaptic time constants τ. 1000 LIF neurons, decoded data are filtered with a 1 ms time-constant exponential low-pass. ⌨ *Code*

Feedback $f(x) = -x$ The dynamics of a system with the feedback function $f(x) = -x$ are depicted in fig. 2b. The system converges to exactly one half of the input $u(t)$. Rearranging the function describing the represented value x yields

$$x = g(u) + f(x) = u - x \Leftrightarrow x = \frac{u}{2}.$$

This explains why the function converges to a steady state of $\frac{u}{2}$; however, this analysis does not help us to understand the dynamics of the system (e.g., the bottom half of fig. 2b).

Feedback $f(x) = x^2$ The dynamics for a feedback function $f(x) = x^2$ are depicted in fig. 2c. The system converges to $u(t)$ for inputs $u(t) \lessapprox 0.4$ but diverges for larger values. Interestingly, once the system is in a state with $x > 0.4$, it will not converge back to $u(t)$ for values of $u(t)$ between approximately 0.2 and 0.4. This means that the system has some kind of memory. For negative inputs $u(t)$ the system approximately converges to $\frac{1}{2}u(t)$.

Mathematical Analysis

Note: While the above examples give us a feeling for what the dynamics of a neural system with recurrent connection might look like, these systems remain quite mysterious. We need to perform some mathematical analysis in order to truly understand these networks.

In order to gain a better understanding of recurrent dynamics, let's first revisit a simple feed-forward network. Figure 3 shows a single neuron population with varying synaptic filter time-constants τ. As we can see, the dynamics of the system almost exclusively depend on the

filter time constant. Furthermore, as we have already discussed, it does not matter whether we apply the filter before or after the encoding process. Correspondingly, it is sufficient to analyse dynamics in terms of represented values and to ignore individual neurons.

Note: *Assumptions.* As so often, we will ignore the fact that we are dealing with vectorial quantities in the following subsection and instead assume that $d = 1$. That being said, all equations also hold for higher-dimensional representations.

Furthermore, we will assume that the synaptic filter is the same for both the synapses from the pre-populations, as well as the synapses filtering the recurrent connections. Often, we will find that the recurrent connections have a longer time constant than the input connections. However, the equations we derive still work reasonably well.

Analysing Neural Network Dynamics

The equation detailing the recurrence relationship $x = g(u) + f(x)$ did not not take time into account and thus does not include the synaptic filter. Assuming that both the input and the recurrent connections are filtered by the same filter (see the assumptions listed above) we get the following dynamical system:

$$x(t) = \big(h * \big(g(u) + f(x)\big)\big)(t) = \int_{-\infty}^{\infty} h(t')\big(g(u(t-t')) + f(x(t-t'))\big)\mathrm{d}t' \,.$$

Unfortunately, and as we have seen in previous lectures, the convolution operator is a little unwieldy. We could eliminate the convolution by switching to the Fourier Domain, however, there is an alternative, more powerful alternative: the Laplace transformation $\mathcal{L}\{f\} = F(s)$.

Note: The Laplace transformation is a more general version of the Fourier transformation.

$$\mathcal{L}\{f\} = F(s) = \int_0^{\infty} f(t) e^{-st}\,\mathrm{d}t\,,$$

where s is a complex-valued parameter $s = \sigma + i\omega$. Notice that the Laplace and Fourier transformation are almost the same if we let $\sigma = 0$; the major remaining difference are the integration boundaries. Since the lower integration boundary of the Laplace transform is zero (instead of $-\infty$ in the case of the Fourier transformation), the Laplace transformation is more suitable for causal systems.

The Laplace and Fourier transformation share many commonalities, including linearity, and the fact that convolution becomes multiplication. Importantly however, it also holds

$$\mathcal{L}\left\{\frac{\mathrm{d}f}{\mathrm{d}t}\right\} = \mathcal{L}\{\dot{f}\} = sF(s)\,,$$

that is, a time-differental just turns into multiplication with the variable s. This makes the Laplace transformation an important tool for dealing with differential equations.

The Laplace-transform of the first-order exponential low-pass filter in eq. (1) becomes:

$$\mathcal{L}\{h\} = \int_0^\infty h(t)e^{-st}\,\mathrm{d}t = \int_0^\infty \frac{1}{\tau}e^{-t'/\tau}e^{-st'}\,\mathrm{d}t' = \int_0^\infty \frac{1}{\tau}e^{-t'(1/\tau+s)}\mathrm{d}t'$$
$$= \left[-\frac{1}{\tau\left(\frac{1}{\tau}+s\right)}\right]_0^\infty = \frac{1}{1+s\tau} = H(s).$$

Hence, we can rewrite the dynamical system we were looking at in the Laplace domain

$$X(s) = H(s)\big(G(s)+F(s)\big) \Leftrightarrow X(s)(1+s\tau) = G(s)+F(s) \Leftrightarrow sX(s) = \frac{1}{\tau}\big(G(s)+F(s)-X(s)\big).$$

Converting back to the time domain we get the following differential equation

(Recurrent connection differential equation)

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = \frac{1}{\tau}\big(g(u(t))+f(x(t))-x(t)\big). \tag{2}$$

Note: *Canonical dynamical system, phase portraits, and equilibria.* The canonical form of time-independent a dynamical system

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \varphi(\mathbf{u}(t), \mathbf{x}(t))$$

where $\varphi(u,x)$ is an arbitrary function mapping an input u and the state x onto a state differential. We can draw the function $\varphi(u,x)$ as a so called *phase portraits* by drawing an arrow corresponding to the state update in an x-u coordinate system (fig. 4).

The points where $\varphi(u,x) = 0$ are called *equilibria* (singular: equilibrium). An equilibrium can be either *stable*, or *unstable*. Intuitively, an equilibrium is *stable* if a small perturbance causes the system to converge back to the equilibrium. An equilibrium is *unstable*, if small perturbances cause the system to drift away from the equilibrium point.

Mathematically, one can compute whether a point $(\mathbf{x}, \mathbf{u})$ is an unstable equilibrium by computing the Jacobi J matrix of φ at that point and determining whether any of the real components of the eigenvalues of J are positive.

2.3 Analysis of the Previous Experiments

Given the differential equation in eq. (2) we can now analyse the dynamics of the systems we explored above. Note that in all cases our input translation function $g(u) = u$.

Analysing $f(x) = x+1$ For this particular feedback function, our dynamical system becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = \frac{\mathrm{d}}{\mathrm{d}t}\varphi(x(t), u(t)) = \frac{1}{\tau}\big(u(t)+x(t)+1-x(t)\big) = \frac{1}{\tau}\big(u(t)+1\big).$$

Looking at the phase portrait for $f(u,x) = u-2x$ in fig. 4, we can see that this function will diverge for inputs $u(t) \neq -1$.

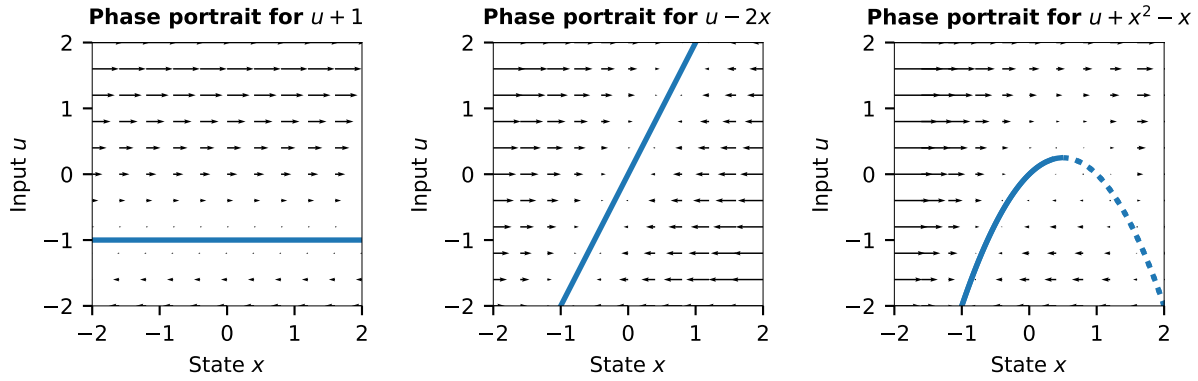


Figure 4: Phase portraits (see note) for the dynamical systems corresponding to the three feedback functions we experimented with. Blue lines correspond the equilibria of the dynamical system $f(u,x)$; dotted blue line corresponds to unstable equilibria. *Code*

Analysing $f(x)=-x$ For this feedback function, our dynamical system becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t)=\frac{\mathrm{d}}{\mathrm{d}t}\varphi(x(t),u(t))=\frac{1}{\tau}\big(u(t)-2x(t)\big),$$

which can be re-written as

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t)=\frac{\mathrm{d}}{\mathrm{d}t}\varphi(x(t),u(t))=\frac{2}{\tau}\left(\frac{u(t)}{2}-x(t)\right),$$

Looking at the phase portrait for $f(u,x)=u-2x$ in fig. 4, we can see that this system has an line-attractor at $x=\frac{u}{2}$.

Analysing $f(x)=x^2$ For this feedback function, our dynamical system becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t)=\frac{\mathrm{d}}{\mathrm{d}t}\varphi(x(t),u(t))=\frac{1}{\tau}\big(u(t)+x(t)^2-x(t)\big).$$

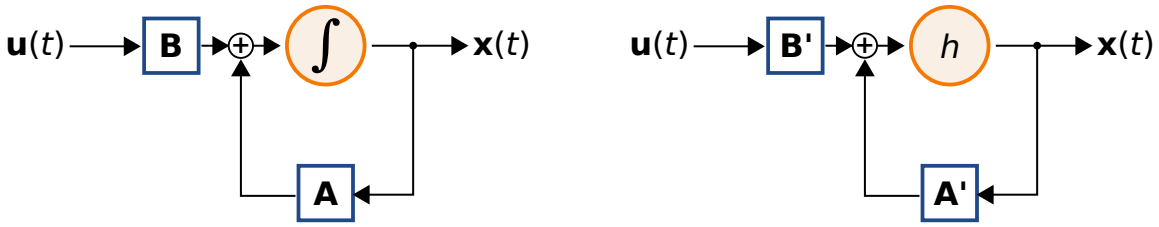
This system can be re-written as a dynamical system with two attractors

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t)=\frac{1}{\tau}\left(x(t)-1+\frac{\sqrt{1-4u(t)}}{2}\right)\left(x(t)-1-\frac{\sqrt{1-4u(t)}}{2}\right).$$

These two attractors are visible in the phase portrait for $\varphi(u,x)=u+x^2-x$ (fig. 4); however, we can see that one attractor (on the left side of the portrait) is stable, whereas the other attractor is unstable (right side).

3 Implementing Arbitrary Dynamical Systems

We have seen how we can analyse the dynamics emerging from a certain input and feedback function pair $g(\mathbf{u})$, $f(\mathbf{x})$. In other words, given a feedback function $f(\mathbf{x})$ and an input function $g(\mathbf{u})$ we can find the corresponding dynamical system $\varphi(\mathbf{u},\mathbf{x})$. This raises the question whether we can also invert this process, i.e., given a dynamical system $\varphi(\mathbf{u},\mathbf{x})$ find the feedback and input function $f(\mathbf{x})$, $g(\mathbf{u})$ that implement this dynamical system in a neural context.



(a) Canonical LTI system **(b)** Neural LTI system

Figure 5: Comparison between an LTI system being evaluated using an integrator, compared to the corresponding neural implementation. Our goal is to find matrices $\mathbf{A}'$ and $\mathbf{B}'$ such that the two systems are equivalent.

3.1 Transforming a Linear Time-Invariant (LTI) System

We will first look at a special case, a linear time-invariant dynamical system (LTI). This is a dynamical system of the form

$$\varphi(\mathbf{u}, \mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \qquad \text{where } \mathbf{u} \in \mathbb{R}^{d'}, \mathbf{x} \in \mathbb{R}^{d}, \mathbf{A} \in \mathbb{R}^{d \times d}, \mathbf{B} \in \mathbb{R}^{d' \times d},$$

and $\mathbf{A}$ is a matrix describing the feedback and $\mathbf{B}$ is a matrix describing a mapping from the input onto the state differential.

Normally, we would implement an LTI system using a perfect integrator. Integration of the differential gives us a new state, which is fed back for the computation of the next state differential (fig. 5a).

However, when building neural networks, we do not have access to a perfect integrator. Instead, we have the synaptic filter h, which can be seen as a "leaky integrator". Hence, we are looking for a way to rewrite the dynamical system $\varphi(\mathbf{u}, \mathbf{x})$ into a new dynamical system $\varphi'(\mathbf{u}, \mathbf{x})$, such that φ' is equivalent to φ in the context of a leaky integrator (fig. 5b). In particular, we would like to find another LTI system

$$\varphi'(\mathbf{u}, \mathbf{x}) = \mathbf{A}'\mathbf{x} + \mathbf{B}'\mathbf{u}.$$

Writing the canonical dynamical system and the neural dynamical system in the time domain we get

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \qquad \mathbf{x}(t) = \big(h * \big(\mathbf{A}'\mathbf{x} + \mathbf{B}'\mathbf{u}\big)\big)(t).$$

We would like to find $\mathbf{A}'$ and $\mathbf{B}'$ such that the two systems behave in the same way. Converting to the Laplace to eliminate the convolution we get

$$sX(s) = \mathbf{A}X(s) + \mathbf{B}U(s), \qquad X(s) = H(s)(\mathbf{A}'X(s) + \mathbf{B}'U(s)).$$

Let's expand the right equation and bring it into the same form as the left equation:

$$\begin{aligned} X(s) &= \frac{1}{1+s\tau}(\mathbf{A}'X(s) + \mathbf{B}'U(s)) \\ \Leftrightarrow X(s)(s\tau+1) &= \mathbf{A}'X(s) + \mathbf{B}'U(s) \\ \Leftrightarrow s\tau X(s) &= \mathbf{A}'X(s) + \mathbf{B}'U(s) - X(s) = (\mathbf{A}' - \mathbf{I})X(s) + \mathbf{B}' \\ \Leftrightarrow sX(s) &= \frac{1}{\tau}(\mathbf{A}' - \mathbf{I})X(s) + \frac{1}{\tau}\mathbf{B}'. \end{aligned}$$

Comparing to our target equation we get

$$\mathbf{A} = \frac{1}{\tau}\left(\mathbf{A}' - \mathbf{I}\right), \qquad \mathbf{B} = \frac{1}{\tau}\mathbf{B}'.$$

Rearranging yields

(Neural implementations of LTI dynamical systems)

$$\mathbf{A}' = \tau\mathbf{A} + \mathbf{I}, \qquad \mathbf{B}' = \tau\mathbf{B}. \tag{3}$$

3.2 Transforming an Additive Time-Invariant System

We can apply the same math to a non-linear time-invariant dynamical system φ that can be decomposed into an input translation function $g(\mathbf{u})$ and a feedback function $f(\mathbf{x})$, that is $\varphi(\mathbf{u}, \mathbf{x}) = g(\mathbf{u}) + f(\mathbf{x})$. Note that this is more general as a linear time-invariant system – since g and f can be non-linear functions – but not quite a general dynamical system.

Assuming that our target system is $\varphi'(\mathbf{u}, \mathbf{x}) = g'(\mathbf{u}) + f'(\mathbf{x})$, i.e., we would like to find the input translation function g' and the feedback function f' to use in our neural system. Using the same approach as above, we get

$$\begin{aligned} X(s) &= H(s)\big(G(U(s)) + F(X(s))\big) \\ \Leftrightarrow sX(s) &= \frac{1}{\tau}\big(F(X(s)) - X(s)\big) + \frac{1}{\tau}G(X(s)). \end{aligned}$$

Transforming back to the time domain, comparing to the target system and rearranging yields

(Neural implementation of additive time-invariant dynamical systems)

$$f'(\mathbf{x}) = \tau f(\mathbf{x}) + \mathbf{x}, \qquad g'(\mathbf{u}) = \tau\mathbf{u}. \tag{4}$$

3.3 Transforming Arbitrary Dynamical Systems

The textbook is also giving an equation that allows us to transform an arbitrary, time-variant dynamical system $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \varphi(\mathbf{u}, \mathbf{x}, t)$. In this general case we get

$$\Phi'(U(s), X(s), s) = \frac{1}{\tau}\Phi(U(s), X(s), s) + X(s).$$

As we can see, we generally have to scale our dynamical system by $\frac{1}{\tau}$ and feed back the current state in order to "remind" the system of its past.

3.4 Examples

In this example we are using the above equations in order to implement some common dynamical systems in a neural network.

Integrator An integrator is defined as the following dynamical system:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \mathbf{u},$$

that is, if we write this as a canonical LTI system (eq. (3)), we get

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \qquad \text{where } \mathbf{A} = 0, \mathbf{B} = \mathbf{I}.$$

Plugging the matrices $\mathbf{A}$ and $\mathbf{B}$ into eq. (3), we get

$$\mathbf{A}' = \mathbf{I}, \qquad \mathbf{B}' = \tau\mathbf{I}.$$

Figure 6 shows an integrator implemented in a neural ensemble. In order to verify that our neural implementation is working correctly, we can compute the ideal response.

In particular, for the step function input (left diagram) we get

$$u(t) = \begin{cases} 0 & \text{if } t < 0.1, \\ 1 & \text{if } 0.1 \leq t < 1.1, \\ 0 & \text{if } 1.1 \leq t, \end{cases} \qquad x(t) = \int_0^t u(t')\mathrm{d}t' = \begin{cases} 0 & \text{if } t < 0.1, \\ t - 0.1 & \text{if } 0.1 \leq t < 1.1, \\ 1 & \text{if } 1.1 \leq t. \end{cases}$$

We can see that the neural population seems to drift from the ideal value of one, but stays relatively constant afterwards. Such errors are inevitable. Neural implementations of dynamical systems are always approximations. As we have discussed before, errors in neural representations are caused by (at least) two kinds of error: static distortion and noise. Analysing the impact of noise on the dynamics is hard. However, we can analyse this drift by drawing the phase-portraits taking the error due to static distortion into account fig. 7.

For the sine wave input (middle diagram) we would expect

$$u(t) = \sin(2\pi t), \qquad x(t) = \int_0^t \sin(2\pi t')\mathrm{d}t' = \left[-\frac{\cos(2\pi t')}{2\pi}\right]_0^t = \frac{1 - \cos(2\pi t)}{2\pi}.$$

This is also approximately what the neural ensemble is computing.

Oscillator An oscillator can be easily written down in terms of the following LTI system

$$\mathbf{A} = \begin{pmatrix} 0 & -\omega \\ \omega & 0 \end{pmatrix}, \qquad \mathbf{B} = 0$$

where ω is the angular velocity. Applying eq. (3) we get

$$\mathbf{A}' = \begin{pmatrix} 1 & -\tau\omega \\ \tau\omega & 1 \end{pmatrix}, \qquad \mathbf{B}' = 0.$$

The dynamics of the resulting network are depicted in fig. 8. The phase portrait is shown in fig. 9. Notice how the limited dynamic range of the neurons naturally restricts the maximum amplitude of the oscillation.

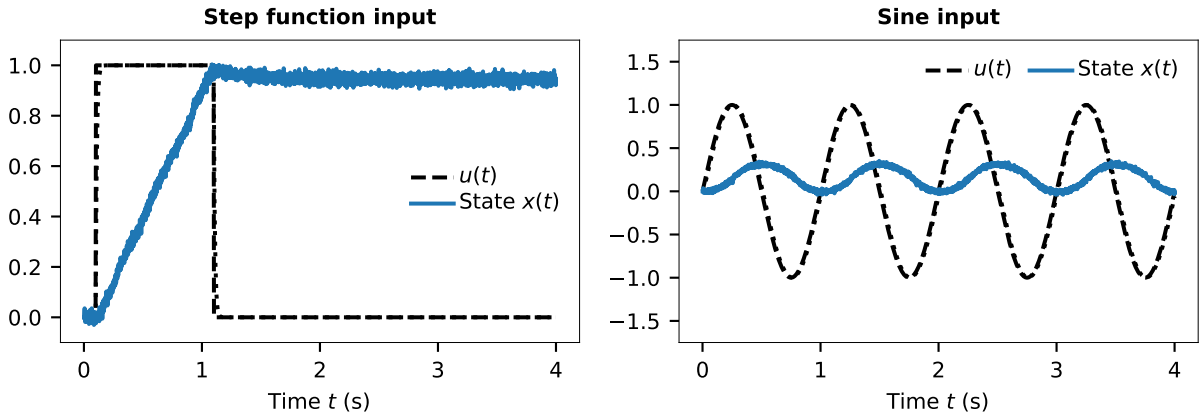


Figure 6: Dynamics of an integrator implemented using a neural ensemble. Decoded values filtered with a 100 ms first-order exponential low pass. The right plot shows the phase portrait of the system implemented by the neural population. ▩ *Code*

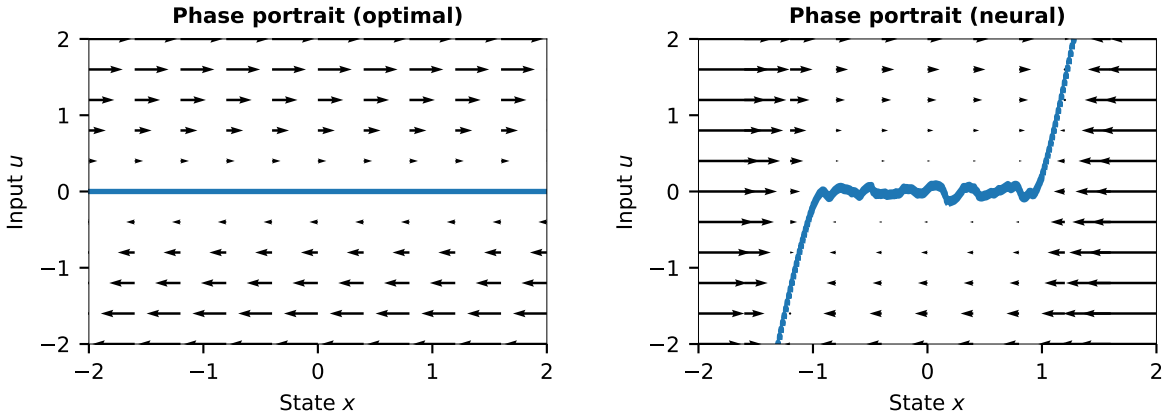


Figure 7: Phase portraits of the optimal (left) and neural (right) implementation of an integrator. ▩ *Code*

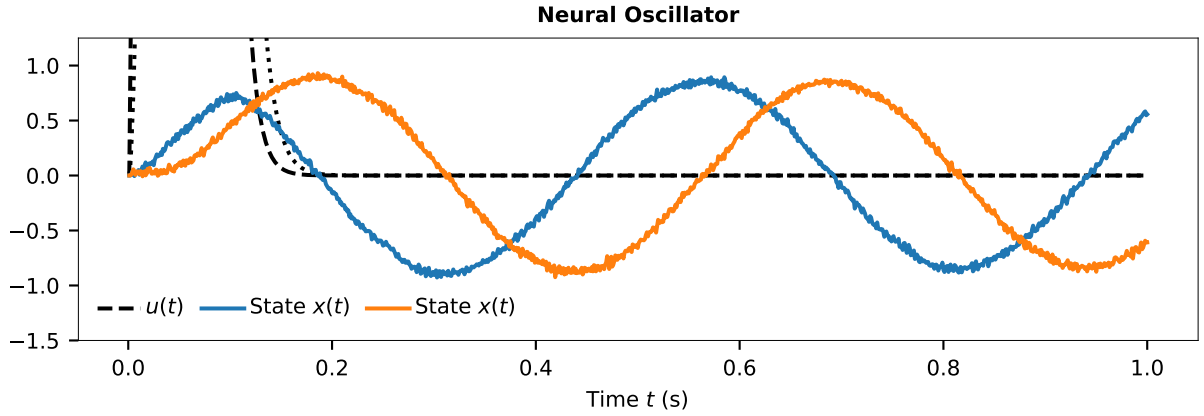


Figure 8: Oscillator with $\omega = 4\pi$ (i.e., a frequency of 2 Hz). The input in the beginning is required in order to kickstart the oscillator. ▩ *Code*

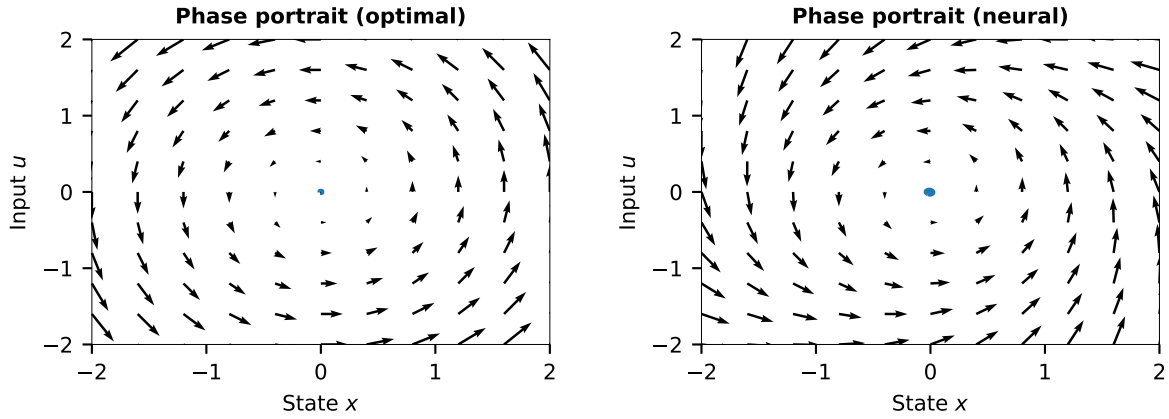


Figure 9: Phase portraits of the optimal (left) and neural (right) oscillator dynamics. ⌨ *Code*

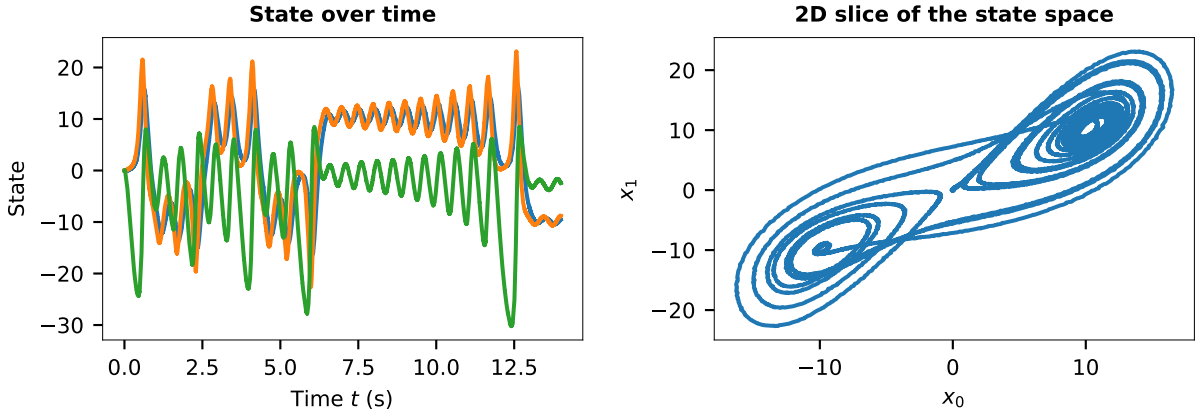


Figure 10: Neural implementation of the Lorentz Attractor. ⌨ *Code*

Lorentz Attractor A variant of the Lorentz Attractor is given as

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \begin{pmatrix} 10x_2(t) - 10x_1(t) \\ -x_1(t)x_3(t) - x_2(t) \\ x_1(t)x_2(t) - \frac{8}{3}(x_3(t) + 28) - 28 \end{pmatrix}.$$

we can use eq. (4) to implement this system as a neural ensemble. An example showing this attractor is given in fig. 10.

4 Dynamics in Biological Systems: Eye Control

As an example of a dynamical system in biology, we can have a look at horizontal eye control. We know that horizontal eye position is controlled within a part of the brainstem called “Nuclei Prepositus Hypoglossi”. In particular, we know that this part of the brain receives an eye velocity v as an input $u(t)$ and outputs an eye position $x(t)$, which in turn is translated into corresponding muscle tensions.

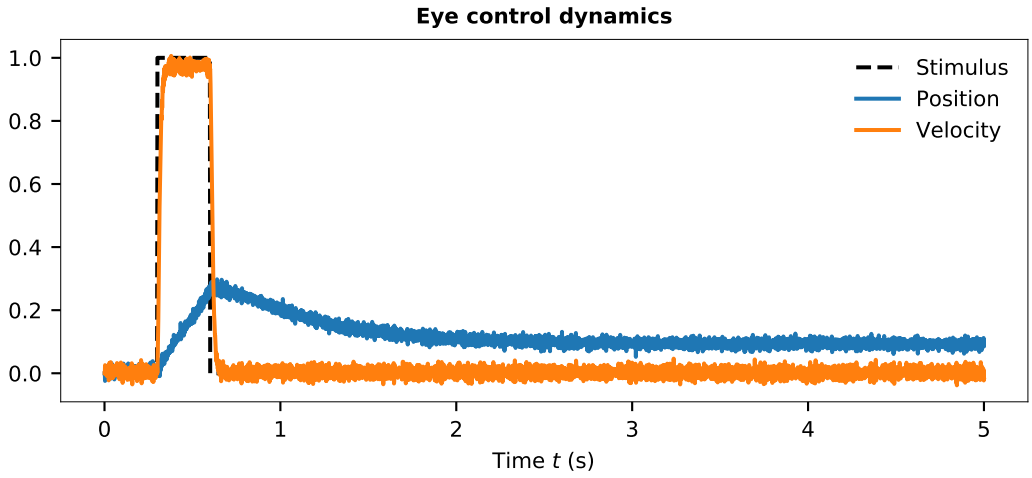


Figure 11: Simple horizontal eye control model

The optimal dynamical system that turns a velocity signal into a position is an integrator. We know from above how to implement a perfect integrator. However, further experiments reveal that the neural circuit in the Nuclei Prepositus Hypoglossi is not a perfect integrator. If a person is told to fixate a point, and we afterwards turn the light off while telling the person to still look into the same direction, their eye will drift back to the centre. This is not due to some mechanical effect in the motor system, but due to the representation produced by the Nuclei Prepositus Hypoglossi converging back to zero. This "drift to the centre" can be modelled as a leaky integrator with an exponential decay with a time constant τ_{eye} of approximately 70 s.

Correspondingly, we get the following differential equation describing the dynamics of the horizontal eye control system

$$\dot{x}(t) = -\frac{1}{\tau_{\text{eye}}} x + \nu\,.$$

This is an LTI system. The corresponding NEF model would thus be

$$\mathbf{A}' = \tau - \frac{1}{\tau_{\text{eye}}} + 1\,, \qquad \mathbf{B}' = \tau\,.$$