SYDE 556/750
Simulating Neurobiological Systems
Lecture 7: Temporal Basis Functions

Andreas Stöckel

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

February 13, 2020

Accompanying Readings: See References

Contents

1 Introduction

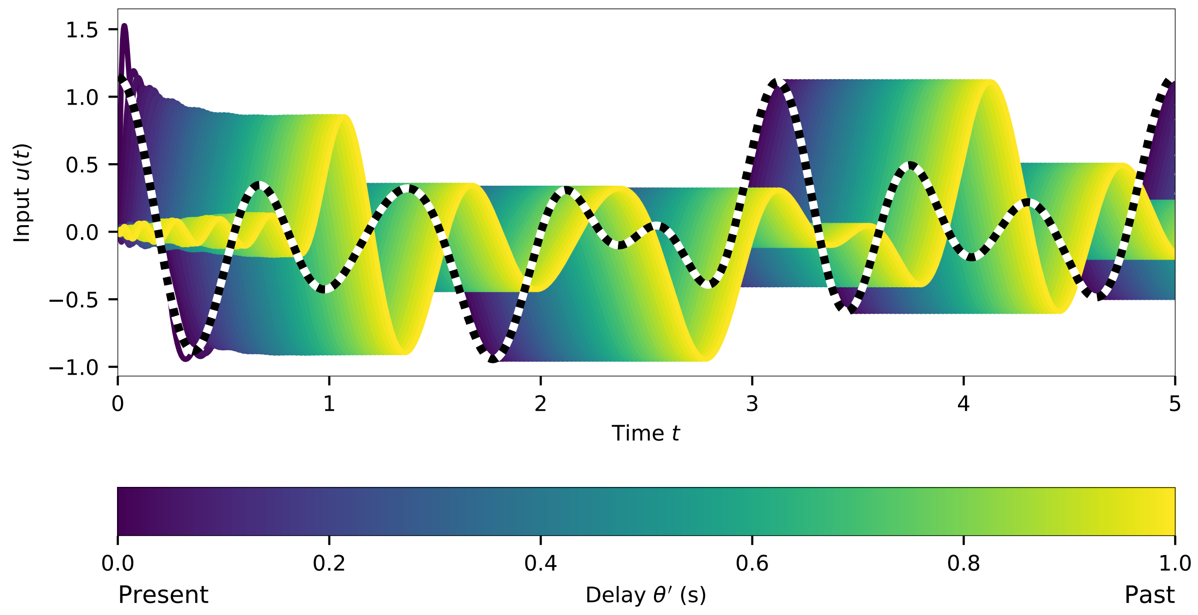


Figure 1: Example of the "delay network" we are going to discuss in the lecture. The dotted line is the input $u(t)$. The coloured lines correspond to delayed versions of the input signal. These delayed versions are all decoded from the same function representation $f_{[t-\theta,t]}(\theta')$. This diagram has been generated by decoding from a mathematically perfect implementation of the delay network. *Code*

Note: In this lecture we are going to discuss a recent addition to the Neural Engineering Framework, the so called Delay Network developed by Aaron Voelker, first presented in [1], and discussed in more detail in his PhD thesis [2]. The Delay Network allows us to represent a function over time in a neural population. A more general version of the Delay Network has been presented at NeurIPS 2019 as the "Legendre Memory Unit" (LMU) [3]. The LMU can be used as a component within artificial Deep Neural Networks and outperforms other recurrent architectures such as LSTMs in a variety of tasks.

As humans, we often feel as if we have a good recollection of events that happened in the immediate past, i.e. where "immediate" refers to events happening within the last few seconds. As events move into the more distant past, it becomes increasingly harder to recall details. In this lecture, we are going to discuss a system that similarly memorizes stimuli, remembers them for a certain period, and then gradually "forgets" them.

Mathematically speaking, what we would like to have a function $f_{[t-\theta,t]}(\theta')$ which allows us to access stimuli $u(t)$ in a time-interval from $[t-\theta, t]$, that is

$$f_{[t-\theta,t]}(\theta') = u\left(t-\theta'\right), \text{where } 0 \leq \theta' \leq \theta .$$

Put differently, $f_{[t-\theta,t]}(0)$ will return the present stimulus $u(t)$, whereas $f_{[t-\theta,t]}(\theta)$ will return the input θ seconds from the past.

We would like to build a biologically plausible version of such a system. That is, we would like to represent information about the immediate past in the current activity of a population of neurons $f_{[t-\theta,t]}$. In order to do this, we have to solve two problems that we discuss separately

1. **Function representation.** So far we have seen how we can represent vectorial quantities in a neural population. But how can we represent an entire *function* of the form $f_{[t-\theta,t]}(\theta')$ in a neural ensemble?
2. **Updating the function representation.** If we were somehow able to represent functions in our neural population we still need to know how exactly to update this representation over time.

Note: Function *representation* is different from the "transformation" principle we already discussed. In short, we are not interested in computing a function $\mathbf{y} = f(\mathbf{x})$, but instead we would like to *represent* the function f itself in a neural population, i.e. have some way of storing a mapping from a value $\mathbf{x}$ onto values $\mathbf{y}$.

2 Representing Functions

For simplicity, and because this is related to the problem we are trying to solve, let's focus on scalar functions over time $f(t)$, i.e., $f : \mathbb{R} \longrightarrow \mathbb{R}$. How can we represent an interval $[0, T)$ of such a function in a neural population? We already know how a neural population can represent a vector, so let's rephrase the question. How can we represent a function as a vector?

First of all, if we have a parametrised function family $f(t; \mathbf{x})$ we want to represent (i.e., all linear functions, all affine functions, all Normal distributions with a mean μ and standard-deviation σ), then the function parameters can be described as a vector $\mathbf{x}$ that we could use:

$$
\begin{aligned}
f(t;\mathbf{x}) &= mt\,, && \text{Linear function } \Rightarrow \mathbf{x} = \big(m\big)\,,\\
f(t;\mathbf{x}) &= mt + b\,, && \text{Affine function } \Rightarrow \mathbf{x} = \big(m, b\big)\,,\\
f(t;\mathbf{x}) &= \exp\left(-\frac{(t-\mu)^2}{\sigma^2}\right)\,, && \text{Gaussian } \Rightarrow \mathbf{x} = \big(\mu, \sigma\big)\,.
\end{aligned}
$$

If we represent $\mathbf{x}$ and t in the same neural population, we can evaluate $f(t; \mathbf{x})$ in the connection from the pre- to the post-population according to NEF principle two.

In general, if we do not have any more information about the kind of function we would like to represent, there are two (closely related) approaches to representing functions as vectors: sampling/discretisation and basis functions. We discuss these two approaches in the following.

Note: We can generalise what we discuss to multi-variate functions with vectorial output, i.e. $f : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}$. We can treat multi-dimensional *output* $d' > 1$ as d' independent functions $f_1, \ldots, f_{d'}$. For multi-dimensional *inputs* $d > 1$, we need to sample on a higher-dimensional grid. When using basis functions, we need an appropriate basis $\varphi_i : \mathbb{R}^d \longrightarrow \mathbb{R}$.

2.1 Sampling

The idea of sampling is to measure the value of a function $f(t)$ at discrete, equally spaced points $\mathbf{x} = (x_0, \dots, x_{N-1})$, where

$$x_i = f(t_i) = f(\Delta t i)\,, \qquad \text{and } \Delta t \text{ is the sampling interval.}$$

The resulting vector $\mathbf{x}$ holds a representation of $f(t)$ over the interval $[0, T)$, where $T = N\Delta t$.

If we wanted the reconstruct the value of the function at a point t that was not sampled, we can in theory use one of many interpolation techniques to "guess" function values in between samples. If our function is somewhat well-behaved, we would expect the quality of these reconstructions to get better for smaller Δt. That is, the more sample points N we have over the interval $[0, T)$, the better the representation of the function (fig. 3).

Unfortunately, in general, for an arbitrary function $f(t)$, we need an infinite number of sample points to perfectly represent it. Mathematically speaking this is true for any infinitesimally small interval $T \to 0$. A function can "drastically" change its value between any two infinitesimally close points.

Example: These restrictions even apply to continuous functions! The earliest example of a "weird" continuous function that is not differentiable (i.e., not "smooth") at any point is the Weierstrass function published in 1872, defined as

$$f(x) = \sum_{n=0}^{\infty} a^n \cos(\pi b^n x)\,, \quad \text{where } 0 < a < 1, b \text{ is a positive odd integer, and } ab > 1 + \frac{3}{2}\pi\,.$$

Intuitively, this function is continuous at any point—after all, it is just a sum of cosines (which are continuous functions) and the sum of two continuous functions is also continuous. However, one can show that this function is not differentiable at any point.

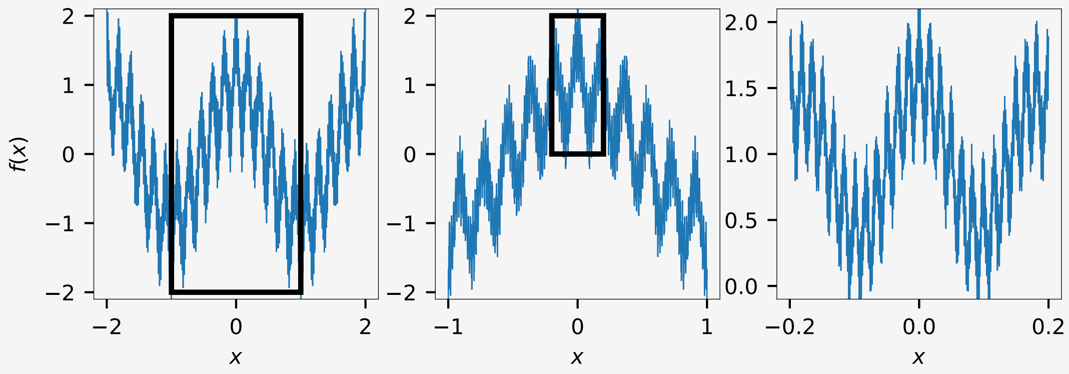
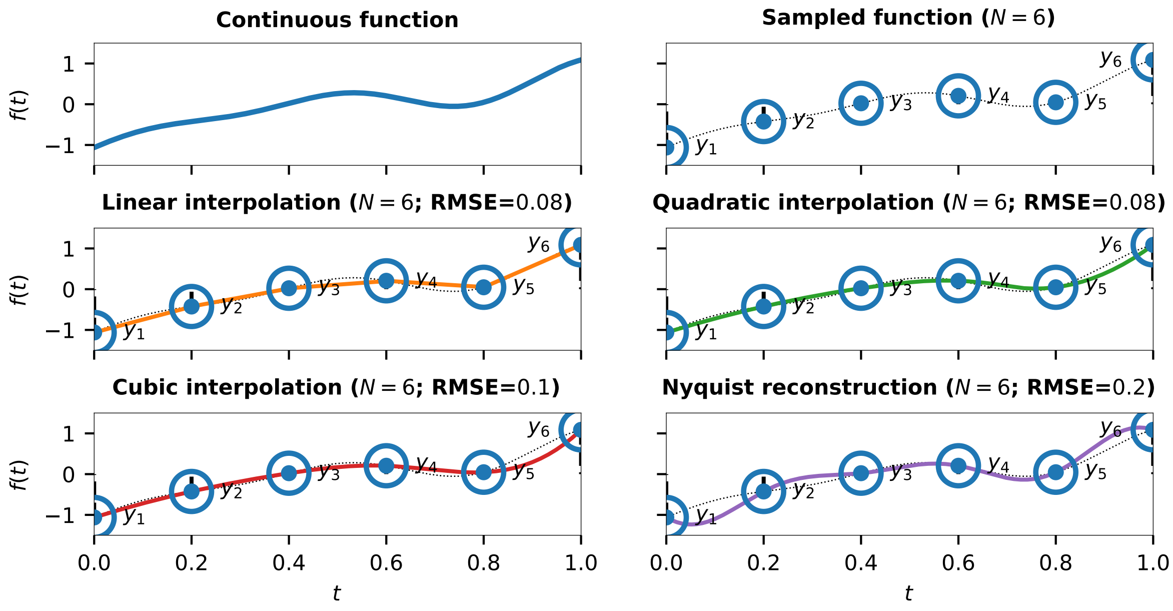
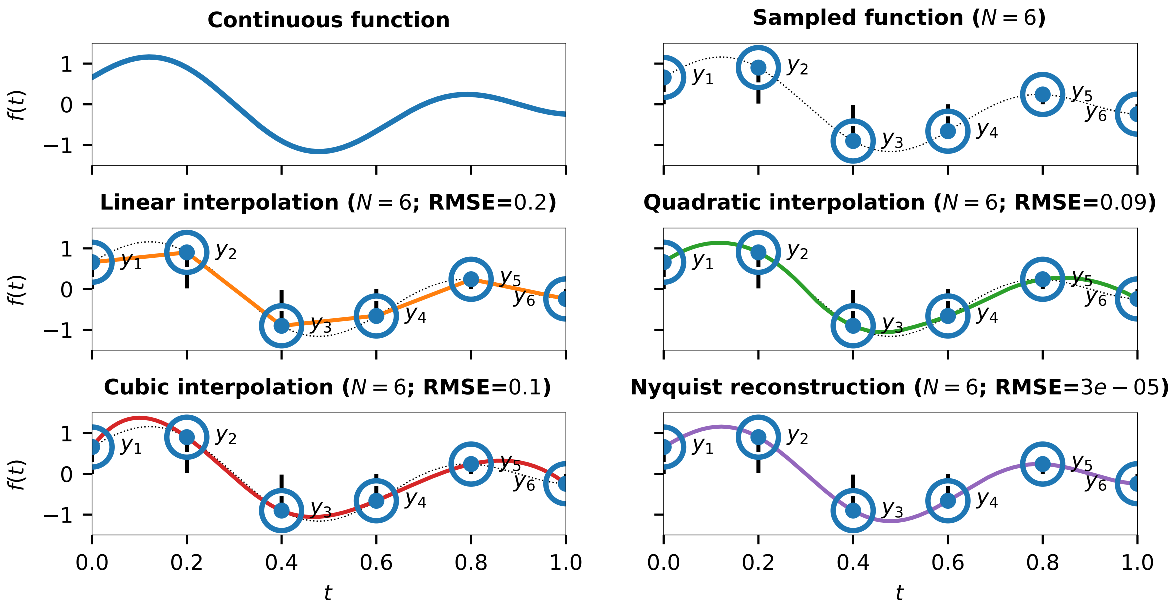


Figure 2. Visualisation of the Weierstrass function for $b = 11$, $a = \frac{1+\frac{3}{2}\pi}{10}$. Black rectangles correspond to the region shown in the neighbouring plot to the right. No matter how far we "zoom" into the function, there is an infinite amount of detail we would have to capture when sampling (this is one of the properties of a *fractal*; note that this does not necessarily imply an infinite amount of *information* in this function; the function is fully determined by two parameters a, b). ⌨ *Code*



(a) Example 1: Bandlimit above 3 Hz



(b) Example 2: Bandlimit below 3 Hz

Figure 3: Example illustrating sampling. A continuous function is sampled at $N = 6$ points (represented as a “lollipop” plot). We can use various interpolation techniques to reconstruct (“guess”) the values between individual sample points. In case the original function is bandlimited to a frequency below $N/2T = 3\,\text{Hz}$ (as in **(b)**, but not **(a)**), the sample points uniquely define the function according to the Nyquist-Shannon sampling theorem.

Guarantees for band-limited signals While the above restriction—namely, that we need infinitely many points to represent any infinitesimally small interval of a function—is true for general, mathematical functions, we have a much better guarantee regarding the required number of required for "physical", i.e., band-limited, signals.

This guarantee is the Nyquist-Shannon sampling theorem, which connects the discrete, sampled world and continuous functions.

(The Nyquist-Shannon Sampling Theorem)
If $f(t)$ contains no frequencies greater than B then it is *completely* determined by samples spaced $\Delta t = \frac{1}{2B}$ apart ($N = 2BT$ equally spaced samples for a time-slice $[0, T)$). There is a *one-to-one mapping* between the samples $\mathbf{x}$ and the function $f(t)$.

Note the emphasis on "one-to-one mapping": if we have $N = 2BT$ equally spaced samples of a function with band-limit B, we can completely reconstruct the function without any losses. Of course, if we have access to the function, we can measure the sample points.

In summary, this means that there is hope that we can represent physical signals by storing just a few values. If we can guarantee that our function does not have any frequencies above B, we only need to store $N = 2BT$ samples; or, in other words we need to sample with a frequency of $f_\mathrm{s} = 2B$ (the "Nyquist Frequency"). So for a signal that is limited to 5 Hz we only need 10 samples per second to be able to perfectly reconstruct it. Conversely, there is no reason to store significantly more than N samples—doing so would just be "wasting space".

Aside: The Nyquist-Shannon Sampling Theorem and Audio Signals. There are some engineering related reasons for sampling slightly faster than the Nyquist frequency. For example, the absolute hearing threshold for (young) humans is about $B = 20\,\mathrm{kHz}$. Correspondingly, to record an audio signal meant for humans (i.e., music or speech recordings), we can do the following: (1) band-limit the original signal to 20 kHz in the capturing device, (2) sample at $f_\mathrm{s} = 40\,\mathrm{kHz}$. This allows us to perfectly reconstruct the band-limited signal according to the Nyquist-Shannon sampling theorem. Since frequencies above 20 kHz are not audible, the result will appear to humans exactly as the original signal.

However, this only works if we can trust our capturing device (i.e., the microphone and its amplifier circuit) to *sharply* cut off all frequencies above 20 kHz. Unfortunately, such perfect filters are extremely difficult (if not impossible) to implement as an analogue device. Frequencies slightly above B will still pass through, albeit being attenuated. This violates the pre-condition of the Nyquist-Shannon sampling theorem, leading to imperfect reconstructions (en effect known as *aliasing*). Hence, it is better to sample faster than Nyquist, to give room for frequencies in the signal that are (slightly) above B.

This is why common sampling rates for audio signals are 44.1 kHz (CD audio), and 48 kHz (DVD audio). Much higher sampling of audio signals for human listening make no sense, except for intermediate signals (i.e., just sidestepping the imperfect analogue filter problem by sampling at a very high rate and applying a cheap and precise digital filter).

Note: *Online resources.* For more information on the Nyquist-Shannon sampling theorem in general and its implications with respect to storing audio signals in particular, refer to this material by Chris Montgomery:

- An excellent Video on the Nyquist Shannon sampling theorem, as well as quantisation: Xiph.org Digital Show and Tell, Episode 2.
- An article on "24 bit / 192 kHz Music Downloads and why they make no sense, explaining the above in much more detail."

Note: *Algorithm for the "Nyquist Reconstruction" of Functions.* The Nyquist-Shannon sampling theorem tells us that it is possible to perfectly reconstruct a function with band limit B as long as we have $N = 2BT$ equally spaced sample points. But how do we compute this reconstruction for discrete signals in practice? I.e., how do we convert N sample points $x'_0, \ldots, x'_N$ to a densely sampled signal $x_0, \ldots, x_M$ with $M \geq N$. The answer is to use the Fourier transformation of the sample points.

First consider the reverse direction. We're given a signal with band limit B and $M \geq N$ equally spaced sample points $x_0, \ldots, x_M$. We can then compute the corresponding Fourier coefficients $\omega_{-M/2}, \ldots, \omega_0, \ldots \omega_{M/2}$. If a function is band-limited with band-width B, this means that only the $N = 2BT$ frequency coefficients $\omega_{-BT}, \ldots, \omega_0, \ldots, \omega_{BT}$ are nonzero. Discarding the zero-coefficients and converting the remaining N frequency coefficients back to the time-domain results in $2BT$ equally spaced sample points in the time domain, $x'_0, \ldots, x'_N$.

Hence, if we want to convert $x'_0, \ldots, x'_N$ to $x_0, \ldots, x_M$ with $M \geq N$, we perform the above steps in reverse: we convert $x'_0, \ldots, x'_N$ to the Fourier domain, fill with leading and trailing zeros so we have M frequency coefficients, and convert to the time-domain.

In practice, there are more efficient ways to do this. This process is also known as *sampling rate conversion*.

2.2 Basis Functions

- **Idea:** $\hat{f}(t) = \sum_{i=1}^{q} x_i \varphi_i(t)$, where the $\varphi_i(t)$, are a set of *basis functions*.
- The coefficients $\mathbf{x} = (x_1, \ldots, x_q)$ form a vector that could be represented in a neural ensemble.
- We can use a linear transformation to compute $f(t)$ at a single point t

$$f(t) = \langle \boldsymbol{\alpha}, \mathbf{x} \rangle \,, \qquad \text{where } \alpha_i = \varphi_i(t) \,.$$

- We can compute the $\mathbf{x}$ approximating a certain $f(t)$ by discretizing $f(t)$, $\varphi_1(t), \ldots, \varphi_q(t)$ and using least squares.
- Example Function Spaces

  - Discrete Fourier Basis
  - Discrete Cosine Basis
  - Legendre Polynomials

$$\varphi_i(t) = \tilde{P}_i(t) = (-1)^{i-1} \sum_{k=0}^{i-1} \binom{n}{k} \binom{n+k}{k} (-t)^k$$

  - Neural Tuning Curves (this is exactly what we have done so far)

- **Remark:** Sampling can be seen as a special case where $\varphi_i(t) = \delta(t - \Delta t i)$.

3 Representing the Past: The Delay Network

- **Motivation:** Implement a perfect delay of a time θ in the Neural Engineering Framework.
- This can be seen as a dynamical system. In the Laplace Domain, a perfect delay is $e^{-s\theta}$.
- Use Padé approximants up to a degree q to compute an LTI system approximating $e^{-s\theta}$. This system has a one-dimensional input $u(t)$ and an internal state $\mathbf{x}(t)$ of dimension q:

$$\begin{aligned}
&\mathbf{x}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) && \\
&\theta\mathbf{A} = a_{ij} \in \mathbb{R}^{q \times q}\,, && a_{ij} = \begin{cases} (2i+1)(-1) & i < j\,, \\ (2i+1)(-1)^{i-j+1} & i \geq j\,, \end{cases} \\
&\theta\mathbf{B} = b_i \in \mathbb{R}^q\,, && b_i = (2i+1)(-1)^i\,.
\end{aligned}$$

- We can implement this LTI system as a neural ensemble using the transformation

$$\begin{aligned}
\mathbf{A}' &= \tau\mathbf{A} + \mathbf{I}\,, \\
\mathbf{B}' &= \tau\mathbf{B}\,.
\end{aligned}$$

- The state $\mathbf{x} \in \mathbb{R}^q$ represents more information than just the input θ seconds ago. It represents the state at every point in time up to θ seconds.
- In fact, $\mathbf{x}$ represents the function $f_{[t-\theta,t]}\left(\frac{\theta'}{\theta}\right) \approx u(t' - \theta')$ in the Legendre basis

$$u(t' - \theta') \approx f_{[t-\theta,t]}\left(\frac{\theta'}{\theta}\right) = \sum_{i=1}^{q} \tilde{P}_i\left(\frac{\theta'}{\theta}\right) x_i(t)\,.$$

- When representing $\mathbf{x}$ in neurons, we can not only decode delays but any function using information from the past θ seconds.
- This network implements an optimal recurrent neural network remembering a slice of the past, as so called “reservoir”.

References

[1] Aaron R. Voelker and Chris Eliasmith. "Improving Spiking Dynamical Networks: Accurate Delays, Higher-Order Synapses, and Time Cells". In: *Neural Computation* 30.3 (Mar. 2018), pp. 569–609. DOI: `10.1162/neco_a_01046`. URL: `https://www.mitpressjournals.org/doi/abs/10.1162/neco_a_01046`.

[2] Aaron R. Voelker. "Dynamical Systems in Spiking Neuromorphic Hardware". PhD thesis. Waterloo, ON: University of Waterloo, 2019. URL: `http://hdl.handle.net/10012/14625`.

[3] Aaron R. Voelker, Ivana Kaji, and Chris Eliasmith. "Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems*. 2019.