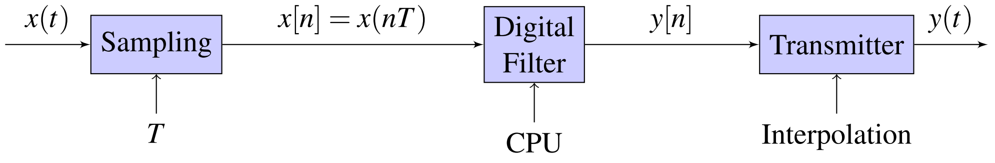
EECS 16B Designing Information Devices and Systems II
Fall 2020 UC Berkeley Note 21

1 Introduction

In the final module of the course, we will take a look at the introductory concepts behind **digital signal processing**. A digital signal processing system can get quite complex but we will show the simplest of models where we take a continuous time signal, sample it every T seconds, apply digital filter, and then interpolate it back into a continuous time signal to transmit.



Putting all of the pieces together, we were briefly introduced to the idea of sampling in the controls module through **discretization.** We saw sampling as the evaluation of a continuous function $x(t)$ at evenly spaced intervals to get a discrete signal $x[n]$.

In the circuits module, we analyzed multiple types of **filters** such as a low and high-pass filter. These filters took continuous, sinusoidal inputs and outputs and are referred to as **analog** signals. We however, will take a look at **digital** signals since computers fundamentally behave in discrete-time.

Lastly, the transmitter takes a discrete signal $y[n]$ and *interpolates* a continuous function $y(t)$. In this note, we will take a look at **interpolation** and understand how we can *reconstruct* a continuous-time signal out of a discrete-time signal. An interesting and practical question is whether or not we can ever reconstruct the signal *perfectly* from the samples; we haven't shown you the tools needed to answer that question yet, but you'll want to keep it in mind.

2 Interpolation

Let's first define the problem statement behind interpolation. Supose we took N evenly spaced apart samples every T seconds from an unknown continuous-time signal $y(t)$.

$$y[0], y[1], \ldots, y[N-2], y[N-1]$$

We would like to recover the original continuous-time signal $y(t)$ that we have taken samples from.

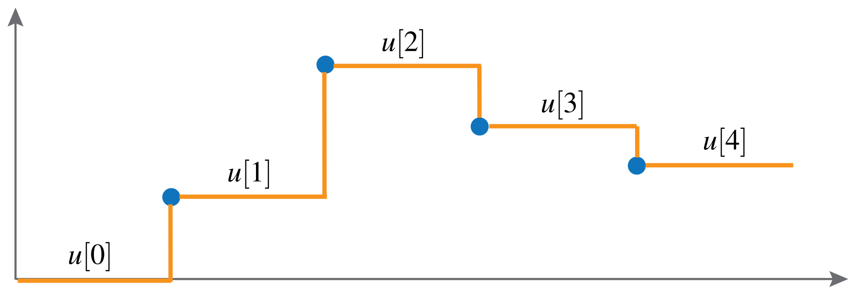
This may sound similar to a regression problem, but we now have an additional constraint that $y(t)$ must match $y[n]$ at each of its sample points. In other words, we are requiring that

$$y[n] = y(nT) \quad \text{for } n = 0, 1, \ldots, N-1.$$

In some sense, we can think of interpolation as the opposite of sampling since we are constructing a continouous-time signal out of a discrete signal.

Interpolation is an important – and indeed necessary – part of any discrete-time system that interacts with the real world, since the real world is always in continuous-time. In other words, interpolation allows for an *interface* between discrete information and the continuous world.

In its role as an interface, we have actually already met interpolation, as we used it implicitly during the control theory module. Recall that for a sampled discrete-time system, applying the discrete-time control $u[n] = y$ actually means that we apply the constant continuous-time input $u(t) = y$ over the interval $t \in [nT, (n+1)T)$ to the continuous-time system. This means that $u(t)$ is an *interpolation* of $u[n]$ – specifically a *zero-order hold* interpolation, as shown below.



The essential strategy for interpolating a signal is to express the interpolation as a weighted sum of some chosen continuous-time functions $\phi_i(t)$; that is, to write

$$y(t) = \sum_i \alpha_i \phi_i(t), \tag{1}$$

where the α_i weights on functions $\phi_i(t)$ are determined by the specific discrete-time samples $y[n]$ in question. From this general starting point, two specific methods emerge: the method of *interpolation by polynomials* and the method of *interpolation by basis functions.*

3 Polynomial Interpolation

Given n distinct points, we can find a unique degree $n-1$ polynomial that passes through these points. Let the polynomial p be

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \tag{2}$$

Let the n points be

$$p(x_1) = y_1, p(x_2) = y_2, \cdots, p(x_n) = y_n,$$

where $x_1 \neq x_2 \neq \cdots \neq x_n$.

We can construct a matrix-vector equation as follows to recover the polynomial p.

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}}_{\vec{a}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\vec{y}} \tag{3}$$

We can solve for the a values by setting:

$$\vec{a} = A^{-1}\vec{y} \tag{4}$$

Note that the matrix A is known as a Vandermonde matrix whose determinant is given by

$$\det(A) = \prod_{1\le i<j\le n} (x_j - x_i)$$

Since $x_1 \neq x_2 \neq \cdots \neq x_n$, the determinant is non-zero and A is always invertible.

Another way to view polynomial interpolation is through basis functions. By picking the polynomial basis $\phi_i(t) = t^i$, we see that

$$y(t) = \sum_i \alpha_i t^i \tag{5}$$

The α_i coefficients should align directly with the a_i that we solved for in (4).

Note that there are more efficient methods to perform polynomial interpolation instead of inverting the Vandermonde matrix through techniques such as **Lagrange Interpolation** but we won't be showing these examples in this note.

4 Interpolation by Basis Functions

Another method to interpolate discrete samples is through basis functions. We construct the interpolation rather directly, as

$$y(t) = \sum_{k=0}^{N-1} y[k]\phi(t-kT),$$

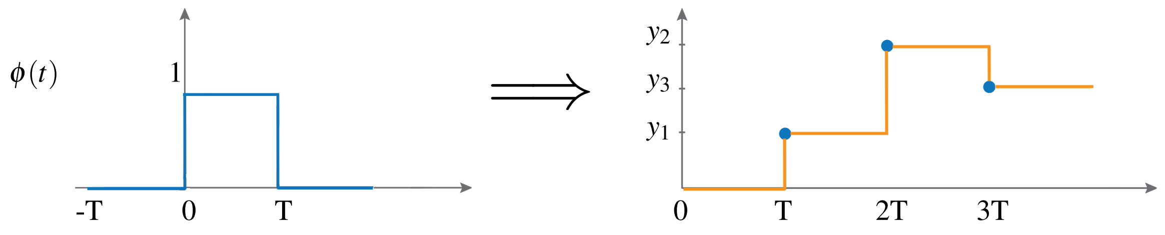
where $\phi(t)$ is a *basis function* [1] of our choosing. The basis function ϕ may be any function we like that has the following two properties:

- $\phi(0) = 1$;
- $\phi(kT) = 0$ for all $k \neq 0$.

That is, the function is 1 when $k = 0$ and 0 at all other k multiples of T. These two properties ensure that the $y(t)$ we constructed in the summation above satisfies $y(kT) = y[k]$.

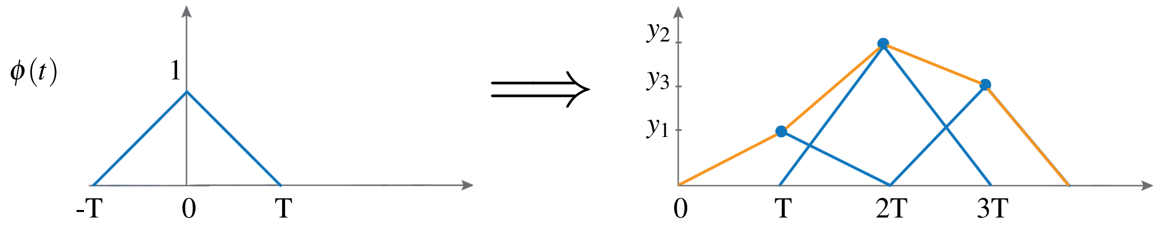
This method gives us more room for creative interpolation schemes. For every new basis function we come up with, we get a new interpolation scheme. Several common interpolation schemes can be written in terms of basis functions.

For example, zero-order hold (ZOH) interpolation can be expressed using a box-shaped basis function:

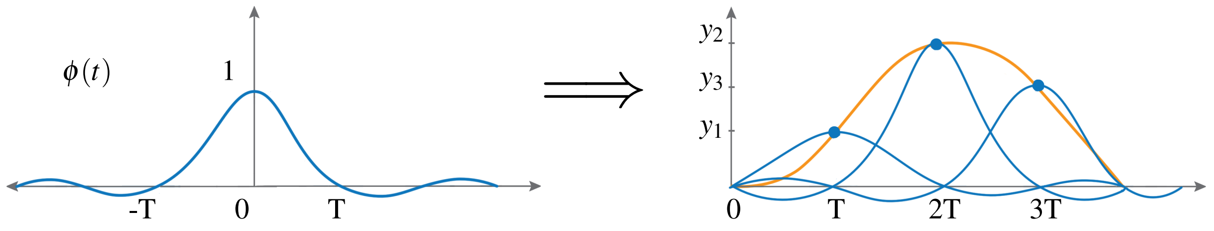


[1] Based on the similarity of the name, you probably suspect that basis functions are related to the basis of a vector space in some way. This is actually correct, in a way, but an exploration of this relationship is sadly out of scope.

Similarly, piecewise linear (PWL) interpolation can be expressed using a triangle-shaped basis function:

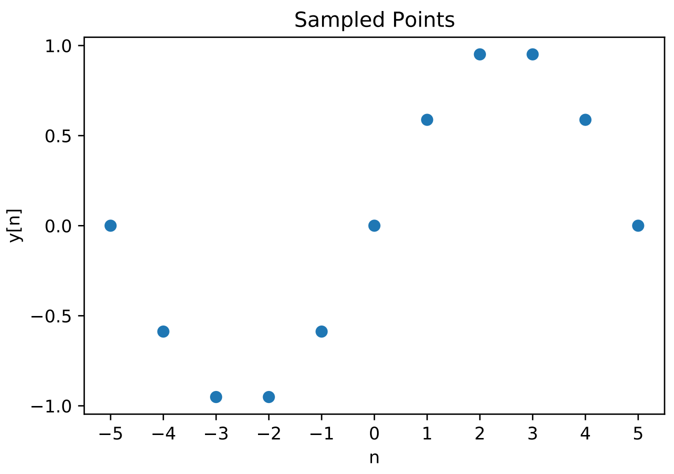


Another useful basis function is the *sinc* function $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$, where we take $\phi(t) = \text{sinc}(t/T)$. The sinc function is differentiable and *band-limited*, the importance of which will become clear in the DFT module.



5 Interpolation Examples

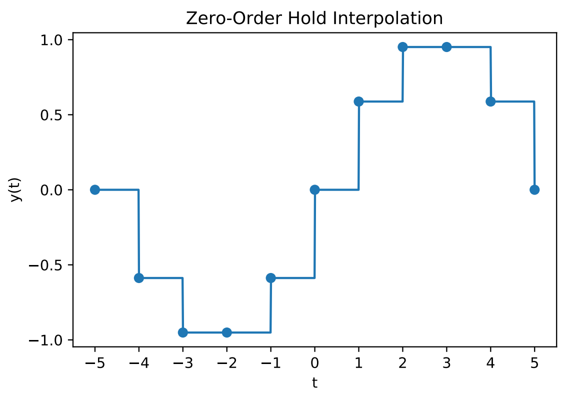
We will now take a look at all of the interpolation schemes and draw comparisons between each approach. Let's start by looking at a set of discrete samples $y[n]$ from $n = -5$ to 5 spaced $T = 1$ apart.



Each interpolation method has its own tradeoffs and there is no perfect solution that works for every single use case. Often times, the specific method is chosen based on the problem we are dealing with. For example, zero-order hold will be useful when performing discretization but will perform very poorly when reconstructing audio samples. Piecewise Linear Interpolation is commonly used in Computer Graphics and Lagrange Interpolation can be used when sending messages over a channel.

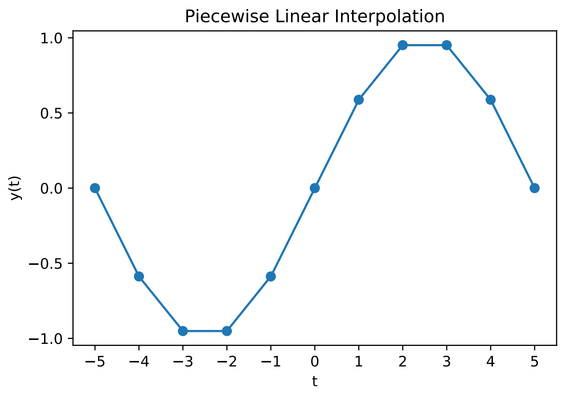
5.1 Zero-Order Hold

To perform Zero-Order Hold Interpolation (ZOH), we keep the value of $y[k]$ constant for $t \in [kT, (k+1)T]$.



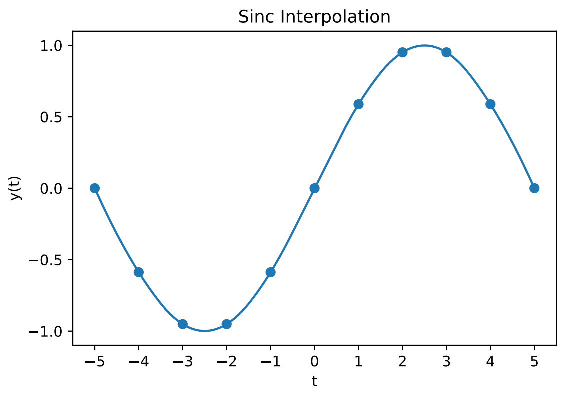
5.2 Piecewise Linear Interpolation

In linear interpolation (PWL), we draw a line between point $y[k]$ and $y[k+1]$.



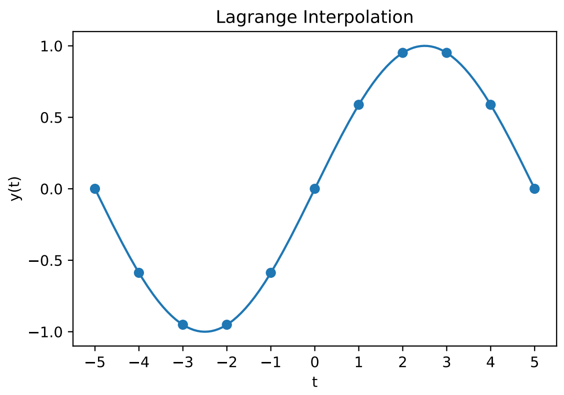
5.3 Sinc Interpolation

We compute sinc interpolation through numerical tools and the result will be a smooth function.



5.4 Polynomial Interpolation

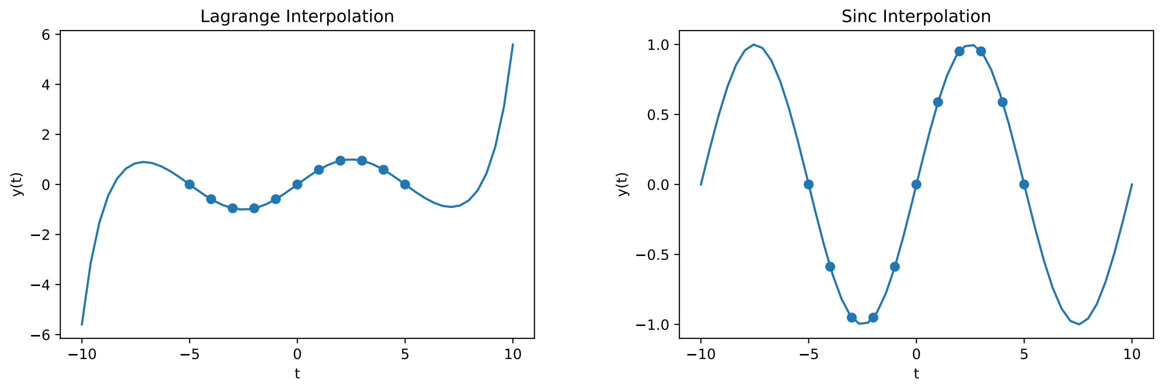
Lagrange Interpolation will try to create a polynomial of degree at most 10 out of the data. The result will be a smooth function similar to sinc interpolation.



5.5 Tradeoffs

Now that we have seen the results from each interpolation, let's discuss the tradeoffs between each method. Each interpolation method will be exact at its sampled point $y[k]$ but may show some error at points that are not multiples of kT. Sinc and Polynomial Interpolation are advantageous if we are trying to reconstruct a smooth function. ZOH and PWL will show larger errors but are advantageous in that they are much easier to implement in practice.

When looking at our example above, Sinc and Polynomial Interpolation have near identical reconstructions. So what makes the two methods different? Remember that interpolation creates a function $y(t)$ for all values of t. Therefore, let's look at values of $y(t)$ from $[-10, 10]$ which are outside of the sampled window $[-5, 5]$.



Polynomial interpolation creates an good reconstruction for the points we have sampled from -5 to 5 but becomes unstable as t grows large. Given n sampled points, polynomial interpolation creates a polynomial of degree at most $n-1$. However, each polynomial t^k becomes unstable when $k > 1$. In addition, polynomial interpolation is very susceptible to *overfitting* the sampled points.

Sinc Interpolation on the other hand, generalizes the sampled pattern and is able to reconstruct a sinusoid. This is because the basis function $\phi(t) = \text{sinc}(t/T)$ searches for periodic patterns. It is also more *stable* than polynomial interpolation since it does not grow to ∞ as $t \to \infty$.

6 Conclusion

In this note, we saw how to recover a continuous-time signal from a set of discrete samples through a technique called interpolation. To do this, we considered a set of discrete samples $x[n]$ spaced T seconds apart. Contrary to regression, interpolation had the additional constraint that each sample $x[n]$ had to match the continuous-time function at $x(nT)$.

In order to perform interpolation, we introduced two technqiues: Polynomial Interpolation and Basis Function Inteprolation. Using Polynomial Interpolation we were able to create a polynomial of degree at most $N-1$ using the sampled points. Basis Functions on the other hand, gave us more creativity on how we could shape our interpolation whether it was the rigid box-shaped basis or the smooth sinc basis.

Depending on the context, each method had its own unique advantages and tradeoffs. Methods such as Zero-Order Hold and Piecewise Linear Interpolation were easy to implement but came at the cost of larger approximation error. Polynomial gave us nice smooth function but we saw that it was susceptible to overfitting and was "unstable" at the endpoints. Moving forward, Sinc Interpolation will be considered the *golden standard* for interpolation and in the next note, we will take a look at how we can reconstruct our signal perfectly given a specific set of conditions.

Contributors:

- Taejin Hwang.
- Murat Arcak.
- Alex Devonport.
- Siddharth Iyer.
- Jane Liang.