EEE2047S 2024 Lab 4

July 4, 2024

1 Lab 4: Reconstruction

1.1 Introduction

It's not obvious how to represent a continuous-time signal $x(t)$ in a way that can be manipulated by a computer. The time index is real-valued, so even if we only care about the signal value over a finite interval there are infinitely many function values to consider and store.

One approach is to assume that $x(t)$ varies slowly, and hope that sampling $x(t)$ at $t = nT$ for n integer and T small is sufficient to characterise the signal. This leads to the discrete-time signal $x[n] = x(nT)$. When this assumption is formalised and holds then the method works, and we use it a lot. The first part of this workbook explores how a discrete signal $x[n]$ can be used to represent or parameterise a continuous-time $x(t)$. Understanding this link and its limitations lets us process "analog" signals $x(t)$ using digital processing of the corresponding $x[n]$. See https://en.wikipedia.org/wiki/Digital_signal_processing.

A more general option is to represent the continuous-time signal $x(t)$ as a linear combination of a fixed set of known basis functions $b_n(t)$:

$$x(t) = \sum_{n=0}^{N-1} c_n b_n(t).$$

Here the signal $x(t)$ is defined for all $t \in \mathbb{R}$, but as written it is completely specified by the finite and discrete set of values $c_n, n = 0, \ldots, N-1$. Not every signal $x(t)$ can be written in this way, and those that can depends on the choice of functions $b_n(t)$.

One simple example uses a polynomial basis with $b_n(t) = t^n$. The signals parameters c_n are then just the coefficients for each polynomial order. Another is the cosine basis $b_n(t) = \cos(n\omega_0 t)$, which can represent all periodic even functions $x(t)$ if N is large enough. Instead of summing from 0 to $N-1$ in the representation we might also have infinite bounds like 0 to ∞, or $-\infty$ to ∞. Even then, the coefficients c_n are still a countable set and can be carefully used to represent or process the continuous-time $x(t)$ that they represent. The second part of this workbook explores this concept.

1.2 Reconstruction from discrete samples

Suppose that $b_0(t)$ is an even function centered on the origin $t = 0$, and for each n the basis function $b_n(t)$ in the representation equation above is $b_0(t)$ shifted until it is centered on some point $t = nT$:

$$x(t) = \sum_{n=0}^{N-1} c_n b_0(t - nT).$$

This representation is particularly useful if we add the requirement

$$b_0(t) = \begin{cases} 1 & t = 0 \\ 0 & t = nT \quad \text{for integer } n. \end{cases}$$

Suppose now that we have access to a discrete set of regular samples $x[n] = x(nT)$ from $x(t)$, and want to determine the corresponding coefficients c_n. Note that:

$$x(kT) = \sum_{n=0}^{N-1} c_n b_0(kT - nT) = \sum_{n=0}^{N-1} c_n b_0((k-n)T) = c_k.$$

The last step above follows because $b_0((k-n)T) = 1$ only if $k - n = 0$, so all the terms in the sum are zero except one. You could also note that $b_0((k-n)T) = \delta(k-n)$ and use the sifting property. Either way $x[k] = x(kT) = c_k$, and the expansion takes the form:

$$x(t) = \sum_{n=0}^{N-1} x[n] b_0(t - nT).$$

We can view this as a reconstruction formula. It takes as input the sample values in the form of a discrete signal $x[n]$ for integer n, and produces the interpolated or reconstructed continuous-time signal $x(t)$. The nature of the reconstruction depends on the prototype basis function $b_0(t)$, and different choices lead to interpolation with different properties.

1.3 Tasks

These tasks involve writing code, or modifying existing code, to meet the objectives described.

1. Zero-order hold reconstruction:

Suppose we have samples $x[n] = x(nT)$ of the signal

$$x(t) = \cos\left(\frac{t-5}{5}\right) - \cos\left(\frac{t-5}{5}\right)^3$$

for $n = 0, \ldots, N-1$, with $N = 10$ and $T = 1.2$. Consider the reconstruction equation

$$x_r(t) = \sum_{n=0}^{N-1} x[n] b_0(t - nT).$$

On the same set of axes plot both $x(t)$ and $x_r(t)$ over the range $t = 0$ to $t = (N-1)T$ for various different interpolants. Do this by generating a vector `xvs` containing the sample values $x[0], \ldots, x[N]$ for the values of N and T specified above; a vector `tvs` should then contain the corresponding time instants for the samples.

[ ]:
```
# Plot signal and sample values
...
plt.plot(tv,xv,'b-');
```

```python
plt.stem(tvs,xvs,'r.');
```

In the code block below, define the required centered interpolation function $b_0(t)$ to use in the reconstruction formula. It should take a vector of time values `tv` and return the corresponding interpolation values `b0v`. In this instance we require $b_0(t) = p_T(t) = p_1(t/T)$, where $p_T(t)$ is the unit pulse of total width T centered on the origin. For your implementation you should ensure that $b_0(-T/2) = b_0(T/2) = 1$.

[ ]:
```python
def b0v_values(tv,T):
    ...
    return b0v
```

[ ]:
```python
# Plot reconstruction basis
tvb = np.linspace(-T,T,1000)
b0vb = b0v_values(tvb,T)
plt.plot(tvb,b0vb,'r-');
```

The code block below should generate the reconstructed signal values at the given set of time instants `tv`, using the sample values `xvs` corresponding to sampling times `tvs`, and placing the result in the vector `xvr` of the same dimension as `tv`. You should assume that the sampled signal values are zero outside of the range over which the samples were taken.

[ ]:
```python
tv = np.linspace(0,(N-1)*T,4000);
xvr = np.zeros(tv.shape)
for i in range(0,len(xvs)):
    xvr = xvr + xvs[i]*b0v_values(tv - tvs[i],T)
```

[ ]:
```python
plt.plot(tv,xv,'b-',label='Actual signal');
plt.stem(tvs,xvs,'r.',label='Sampled signal');
plt.plot(tv,xvr,'g-',label='Reconstructed signal');
plt.legend();
```

---

2. First-order hold reconstruction:

Repeat the reconstruction process developed in the previous section, but with the reconstruction kernel

$$b_1(t) = (1 - |t|/T)p_T(t).$$

The code block below should exit with `xv1r` containing the values of the reconstructed signal at time instants `tv`.

[ ]:
```python
tv = np.linspace(0,(N-1)*T,4000);
def b1v_values(tv,T):
    ...
    return b1v
```

```python
xv1r = np.zeros(tv.shape)
for i in range(0,len(xvs)):
    xv1r = ...

# Plot reconstruction basis
...
```

[ ]:
```python
# Plot signals
...
```