lab_convolution

September 3, 2017

1 Convolution

Convolution is the mathematical procedure that defines or represents the action of a linear and time-invariant (LTI) system on an input signal. Specifically, if a system with an impulse response $h(t)$ is driven by an input signal $x(t)$, then the output is

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\lambda)x(t-\lambda)d\lambda.$$

Every LTI system has an impulse response $h(t)$, which is the output when the input is the signal $x(t) = \delta(t)$.

The expression above is useful if you have expressions for the signals such as $h(t) = e^{-t}u(t)$ and $x(t) = u(t)$, and a symbolic math package can help in this case. However, in reality we can't specify the signals we see and care about using simple analytical expressions, particularly for input and output signals. Yet we still want to work with them.

We saw in a previous lab that if signals are bandlimited then they can be reconstructed from discrete samples. In other words, the discrete signal $x[n]$ obtained by sampling a continuous-time bandlimited $x(t)$ according to $x[n] = x(nT)$ can be used to represent or reconstruct $x(t)$. If we have a symbolic expression for the impulse response, which is often the case, then we can use the sampled input to generate samples from the corresponding output. If all signals involved are represented using samples, then it turns out that we can effectively consider convolution in continuous-time using discrete-time convolution on the samples of the signals involved. This lab explores all of these concepts and their application.

1.1 Representing a continuous-time signal from samples

A previous lab investigated the topic of reconstructing a continuous-time bandlimited signal $x(t)$ from a sampled representation $x[n]$, with $x[n] = x(nT)$. The process involves using the discrete signal $x[n]$ to construct a modulated impulse train

$$x_s(t) = \sum_{n=-\infty}^{\infty} x[n]\delta(t-nT).$$

This is an interesting signal that has both continuous-time and discrete-time characteristics: it is formulated on the continuous-time axis, but contains exactly and only the information contained in the samples $x[n]$. As such it forms the link between the two domains. The signal $x_s(t)$ is then filtered with a reconstruction filter $b_T(t)$ to interpolate between the sample points:

$$x(t) = x_s(t) * b_T(t).$$

The ideal reconstruction filter is the *sinc* function that corresponds to perfect bandlimiting.

The cell below plots the two example signals mentioned above, which will be used for demonstration. Also shown is the discrete set of values obtained by sampling the signals with a spacing T, starting at $n = 0$. It should be noted that neither of these signals are in fact bandlimited, so there will be errors in the approximation, but these errors will be reduced as T becomes smaller.

```
In [178]: import numpy as np
          import matplotlib.pyplot as plt
          %matplotlib notebook

          # Dense set of values for plotting underlying functions
          tvp = np.linspace(-1, 5, 2000);
          hvp = np.where(tvp<0, 0, np.exp(-tvp));
          xvp = np.where(tvp<0, 0, 1);
          yvp = np.where(tvp<0, 0, 1-np.exp(-tvp));

          # Discrete samples starting from n=0
          T = 0.5;  nv = np.arange(0,np.floor(np.max(tvp)/T),1,np.int32);
          tv = nv*T;
          hv = np.where(tv<0, 0, np.exp(-tv));
          xv = np.where(tv<=0, 0, 1);

          # Plots
          fh = plt.figure;
          plt.subplot(211);  plt.plot(tvp,hvp,'r-');  plt.stem(tv,hv);  plt.ylabel('h(t)');
          plt.subplot(212);  plt.plot(tvp,xvp,'g-');  plt.stem(tv,xv);  plt.ylabel('x(t)');  p
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.2 Fully discrete convolution

Suppose now that we want to numerically approximate the convolution

$$y(t) = h(t) * x(t).$$