Discrete Approximation of a Convolution

This note discusses how to approximate a continuous convolution with a discrete convolution, and how Matlab can easily be used to compute this approximation. Matlab works with vectors and arrays of numbers, not continuous functions, so it is essential to develop a familiarity for moving between continuous and discrete methods to apply Matlab to simulating physical systems and solving problems.

Introductory comments on Discrete Time Series vs. Continuous Functions

Our first issue is how to use a vector of numbers to represent a continuous function $f(t)$. We start by selecting a sampling interval- a period of time which is short relative to the phenomenon that we're interested in (we will make this concept much more quantitative once we discuss Fourier theory and the Nyquist theorem in Chapter 4). For example, if we're working with a function that varies over a period of several seconds, then a sampling interval of 0.01 s will probably provide adequately dense sampling.

For each sampling interval, we select an "average" value to assign to the associated sample. This might be the true average over the interval, or the function value at the midpoint of the interval, or the value at some other reasonable point in the interval. We then collect the values into a vector. Notice that since we cannot store vectors of infinite length, we cannot approximate functions which are nonzero everywhere. Typically, it is possible to use enough sampling intervals to cover the portion of $f(t)$ that we're interested in.

For example, the following Matlab code represents some arbitrary function, $f(t)$, between $t = 1$ and $t = 2$ using 100 sampling intervals and stores the result as a sequence stored in the vector x. The function $f(t)$ is evaluated at the midpoint of each sampling interval, at t equal to 1.005, 1.015, 1.025, and so forth.

```
for i=1:100,
  t=1.0+0.005+0.01*(i-1);
  x(i)=f(t);
end;
```

Integrating a function represented in sampled fashion is straightforward by using rectangular strips to approximate the area under the function during each sampling interval. To do this we approximate the function on each sampling interval as constant, and having the value associated with that interval. We then integrate $f(t)$ by adding up the area under $f(t)$ across all sampling intervals. The following bit of Matlab code integrates $f(t)$ from 0 to 1 using the x vector that we just generated.

```
s=0.0;
for i=1:100,
  s=s+x(i)*0.01;
end;
```

Notice that the length of the sampling interval 0.01 plays an important part in this formula. If we chose a different sampling interval, we must scale the sum appropriately to get approximately the same result.

Discrete Convolution

Infinite sequences can be convolved in a fashion very similar to convolution of functions in time. The convolution of the sequence x_i with the sequence y_j is given by

$$z_n = \sum_{k=-\infty}^{\infty} x_k y_{n-k}. \tag{1}$$

In practice, we don't store vectors of infinite length, but rather treat all of the entries outside of our finite length vectors as if they were 0.

The Matlab command "conv" implements a discrete convolution of two finite length vectors. Given a vector x of length r, and a vector y of length s, the convolution operation produces a vector z of length $r+s-1$. Since x has entries 1 through r, and y has entries 1 through s, the nonzero entries in the convolution should be in positions 2 through $r+s$. However, since Matlab arrays always start with index 1 (sorry, c programmers), the entries are shifted one place to the left.

For example, suppose that $x_1 = 1$ and $x_2 = 2$ and all other entries of x are zero. Also suppose that $y_1 = 3$, and $y_2 = 4$, and all other entries of y are 0. In the convolution, using (1), we get that $z_2 = 3$, $z_3 = 10$, $z_4 = 8$, and all other entries in z are 0. In Matlab, we would have $x = [12]$, $y = [34]$, and $z = [3108]$.

Now, how can we compute the discrete approximation to a continuous convolution? To approximate $z(t) = x(t) * y(t)$, start by representing the nonzero parts of $x(t)$ and $y(t)$ by vectors x and y with some suitable sampling interval. Next, compute $z = \mathrm{conv}(x, y)$. Finally, scale z by the sampling interval. The scaling is required because each of the numbers in the vectors x and y represents the average value of $x(t)$ or $y(t)$ over a sampling interval. When we multiply these together and add them up in the convolution, we're effectively integrating, but we need to take into account the width of the sampling intervals.