Simulating Waveforms in MATLAB

By Bruce Ferguson with changes by Mark A. Yoder

Simulating waveforms is one operation for which MATLAB has been specifically designed. However, transitioning from the method we use to represent signals in class analysis and using Maple to the method used in MATLAB takes some explanation.

In your introductory courses, most of the work has been performed using continuous functions and variables, supported by the continuous mathematics you have learned. However, (digital) computers cannot represent continuous variables.

Consider a time domain function x(t). Formally, x(·) is a function of an argument. When we write x(t), we have assigned the variable "t" as the input to the function "x". In this view, x(·) is a box containing a machine. We input the argument (a specific value t_0) into the box, and a value of the function comes out ($x(t_0)$).

A plot of x(t) versus t is a convenient means of representing the function x. The variable "t" is often taken to be continuous, and roughly correlating to the thing we call time. In other words, the variable t can take on any value we would like, to an infinite precision. We typically describe t as being a variable covering a range of values, such as $0<t<10$ sec, and are then interested in the values the function x takes using the values of t as input to the function. When the input argument is continuous variable, the function x(t) becomes a continuous function as well.

However, the function x does not require a continuous variable as input. If a discrete variable t_d is applied as input to the function x, the function $x(t_d)$ becomes discrete. Examples of both situations for the function $x(\cdot) = \cos(\cdot)$ are shown in Figure 1.

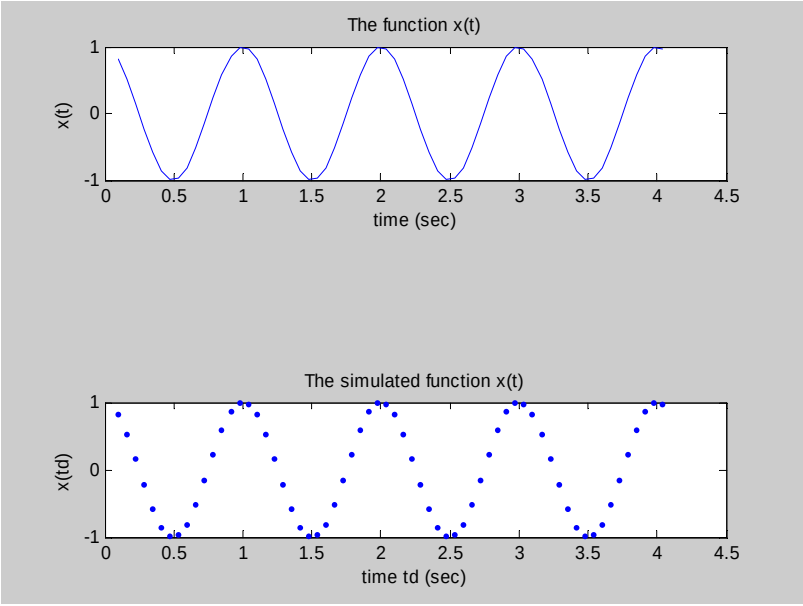

Figure 1. Continuous- and discrete-time version of a function x(t).

When working with MATLAB, we must shift our focus to discrete variables. When we describe the discrete variable t_d, we recognize that the values of t_d are limited. Most often, we describe t_d as covering the range $0< t_d <10$, using N points. When N is 100, the variable t_d takes on the values {0.1, 0.2, 0.3 … 9.9}. In MATLAB, we represent the variable t_d as a vector containing the various elements of t_d, i.e.

$$t_d = \begin{bmatrix} t_1 \; t_2 \; t_3 \cdots t_{100} \end{bmatrix}$$
$$= \begin{bmatrix} 0.1 \, 0.2 \, 0.3 \, 0.3 \cdots 9.9 \end{bmatrix}$$

The variable t_d in MATLAB is thus a 100 element row vector. The values of each element of the vector are determined from the range to be covered and the number of points used to cover that range.

There are two simple ways to create a time vector in MATLAB. The first method uses a built-in function "`linspace`" to automatically create a vector given a range and number of points. (Type "`help linspace`" at the MATLAB command prompt to get complete details.)

The syntax for using linspace is rather straightforward:

```
t=linspace(begin value, end value, number of points);
t=linspace(0.1,9.9,100)
```

It remains to determine what the range of values should be and how many points to use over the desired range. Often, we simulate periodic signals in MATLAB. The fact that the signal is periodic gives us an easy method to specify the range and the number of points:

1. Determine the period of the waveform (*period*)
2. Choose a convenient starting time (*tstart*)
3. Choose how many periods of the waveform to simulation (*numper*, at least 4)
4. The duration of the time vector is then (*duration=numper*period*)
5. Choose how many points per period to simulate (*nsper*, at least 10)
6. Calculate the total number of points *N=nsper*numper*
7. The spacing between points is *dt=duration/N*

In MATLAB syntax, this translates to:

```
period = 1;
tstart = 0.1;
numper = 4;
duration = numper * period;
nsper = 16;
N = numper * nsper;
dt = duration / N;
td = linspace(tstart, duration+tstart-dt , N);
```

We have now created a vector `td` containing N points covering the range (*tstar*t … (*tstart+duration*)). You can verify the number of points in the vector td by using a built-in function for that purpose - "`length(td)`".

Another method for creating a time vector uses the colon notation. This method is best if you know the range of time to cover and resolution (the spacing between time values). The syntax is straightforward:

```
td = [tstart, dt, tfinish];
td = [tstart, dt, duration+tstart-dt];
td = [0.1, 0.1, 9.9];
```

The square brackets are used to define the elements of an array. The resolution (here called dt) will be discussed further below.

The function $x(t_d)$ can easily be produced in MATLAB. Since `td` is a set of discrete variables, we expect `x` to become a set of discrete variables. Specifically, in MATLAB, since td is a vector, we would expect x(td) to also be a vector. The assignment statement "`x=cos(2*pi/T*td);`" is the MATLAB syntax for calculating a vector x containing values of the function x evaluated at the element values of the vector td, i.e.

$$\begin{aligned} x &= \cos\left(\frac{2\pi}{T}t_d\right); \\ &= \left[x(t_1)\ x(t_2)\ x(t_3)\cdots x(t_{N-1})\right] \\ &= \left[\cos(0.1)\ \ \cos(0.2)\ \ \cos(0.3)\ \cdots\ \cos(9.9)\right] \end{aligned}$$

Note that the argument of the cos function in this case is actually $2\pi ftd$. In this case, MATLAB automatically creates `x` as a vector based on the fact that the argument to the function x, which is `td`, is a vector. This is called "implicit type assignment".

We can now use the vectors td and x in all sorts of manners, for example to plot the function x(t). The plot function can either display the elements of a vector variable versus array index ("`plot(x)`"), or displayed versus some other vector ("`plot(td,x)`"). The latter is the form we normally use to display waveforms.

At this point an important difference in the way students perceive the operation of Maple and MATLAB has been revealed. In Maple, when we wish to plot cos(t), we use the plot command "`plot cos(t), t=1..10`". This automatically creates a vector t and computes a vector cos(t) and then plots the vector cos(t) versus the vector t. We just don't see the vector operations. In MATLAB, we explicitly perform the vector operations by creating the vector `td` and then computing the vector `cos(td)`.

We might also wish to create a new function $y=x^2$ (using "`y=x.*x`"). Since x is a vector, which just a row matrix, we must be mindful of which type of operation we wish to perform when using MATLAB. The assignment "`y=x*x`" attempts to matrix-multiple the vector x by itself, which results in an error (why?). The "`.*`" operation, on the other hand, performs an element-by-element multiplication:

$$\begin{aligned} y &= x.*x; \\ &= \left[x(t_1)\ x(t_2)\ \cdots x(t_N)\right].*\left[x(t_1)\ x(t_2)\ \cdots x(t_N)\right] \\ &= \left[x(t_1)*x(t_1)\ \ x(t_2)*x(t_2)\ \ \cdots x(t_{100})*x(t_{100})\right] \end{aligned}$$

We could then plot the vector y versus td using the plot command using ("`plot(td,y)`").

The time resolution was mentioned above as an important parameter of the simulation. By *time resolution* is meant the time spacing between samples of the functions being simulated. If the time resolution is too large, important details of the waveforms might not be represented. If the time resolution is too small, too many points might be used and the simulation might become computationally unwieldy. So, how should the time resolution be chosen?

There are guidelines for choosing the time resolution which you will learn in a later course. For now, a simple guideline will be presented. In our first courses in Signals and Systems, we use periodic waveforms as models of signals. Since the signal is periodic, all of the information contained in the waveform is contained within one period. So it makes sense that we should choose an appropriate number of samples per period to represent that single period. Certainly ten to twenty points should adequately represent one period of the periodic waveform. So, the suggestion for starting out is 10-20 points per period. For example, if the waveform has period T0=0.001 seconds, and we use 20 sample points per periods, a time resolution of $dt=T_0/20=50$ μsec results.

Code used to create example plot:

```
period = 1;
tstart = 0.1;
numper = 4;
duration = numper*period;
nsper = 16;
N = numper*nsper;
dt = duration/N;
td = linspace(tstart, duration+tstart-dt, N);
% or
td = [tstart : dt : duration+tstart-dt];

x = cos(2*pi/period*td);

figure
orient tall
% To create a plot which looks continuous...
subplot (3,1,1); plot(td,x);   %default plot setting creates
                            %a continuous line plot
title('The function x(t)')
xlabel('time (sec)')
ylabel('x(t)')

%To create a plot which emphasizes the discrete nature of %the
variables, and that they are vectors
subplot (3,1,3); plot(td,x,'.');
title('The simulated function x(t)')
xlabel('time td (sec)')
ylabel('x(td)')
```