Applications of logical vectors and matrices in MATLAB

By Gilberto E. Urroz, August 2004

Logical statements in MATLAB are statements that return a value of true (*1*) or false *(0*). The simplest logical statements are comparison statements involving the relational operators *equal (==), not equal (<>), less than (<), less than or equal (<=), greater than (>),* and *greater than or equal (>=)*. Some examples of simple comparison statements are shown next:

```
» 3 == 3
ans = 1

EDU» 5<6
ans = 1

EDU» 4<2
ans = 0

EDU» 3<=2
ans = 0
```

You can also create vectors or matrices of comparison statements, for example:

```
» [3<2 3>2 5~=6 5>=6 4==2 4~=2]

ans =  0     1     1     0     0     1
```

We could refer to this vector as a *logical vector* since its components are logical statements.

Operations with logical vectors and matrices

An interesting result is obtained when including a logical vector in arithmetic operations with numbers. Consider, for example, the product of the previous vector with the number 5. The result is:

```
» 5*[3<2 3>2 5~=6 5>=6 4==2 4~=2]

ans =  0     5     5     0     0     5
```

Notice that the product of the number 5 with a true statement produces a 5, while the product of 5 with a false statement produces a 0.

Consider next the following term-by-term product of one numerical vector and one logical vector:

```
» [1 2 3 4 5 6].*[3<2 3>2 5<=6 5>=6 4==2 4~=2]

ans =  0     2     3     0     0     6
```

Arithmetic operations involving numerical and logical values include addition, subtraction, and multiplication only. The following is an example of term-by-term multiplication of a numerical matrix and a *logical matrix*:

```
» A = [1,2;5,4] % numerical matrix
A =
     1     2
     5     4

» B = [1~=2,3>=2;4<=3,2~=1] % logical matrix
B =
     1     1
     0     1
```

```
» A.*B
ans =
     1     2
     0     4
```

Addition of the numerical matrix **A** and the logical matrix **B** produces the following numerical matrix:

```
» A+B
ans =
     2     3
     5     5
```

Applications of logical vector operations in graphics

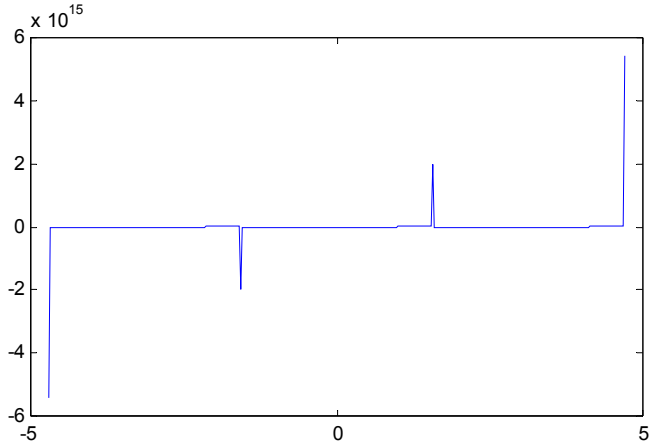
In this section we present some examples of applications of logical vectors operating with numerical vectors to overcome limitations in the production of MATLAB graphics.

Eliminating infinite values in a graph

Consider the plot of the function *tan(x)* in the range $-\pi < x < \pi$. Function *tan(x)* has singularities at $x = -3\pi/2$ and $x = 3\pi/2$. At those values of x, *tan(x)* $\rightarrow \pm\infty$. The following is an attempt to plot this function:

```
-->x = [-3/2*%pi:%pi/100:3/2*%pi];y=tan(x);
-->plot(x,y);
```

The resulting plot is shown next:



The spikes in the figure occur near the critical points $x = -\pi/2$ and $x = \pi/2$, as well as near the extremes of the interval. Those values are responsible for the extremely large range of values in the *y*-axis. Such large range hides the detail of the curve variation through the rest of the *x* domain. To eliminate the very large values in the data set we can re-define vector y by the following operation involving a numerical and a logical vector:

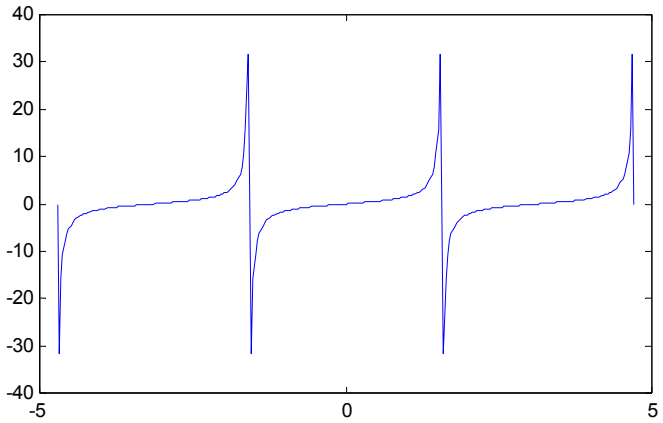
```
» y = y.*(abs(y)<1e10);
```

With this operations, those elements of vector *y* such that $|y_i| < 10^{10}$ get multiplied by 1 (thus preserving their values), while those such that $|y_i| < 10^{10}$ get multiplied by 0. It turns out that near $x = -3\pi/2$, $-\pi/2$, $\pi/2$, $3\pi/2$, the graph approaches $-\infty$ on one side and $+\infty$ on the other side. By making $y = 0$ at those critical points, we force the curve to go from a very large negative number to a very large positive number (or vice versa) through zero, thus producing a

nice transition at those points. Also, by eliminating the very large values in *y* the y-axis range gets smaller, allowing the details of the curve to show in the graph. The graph is obtained by using:

```
» plot(x,y)
```

The resulting graph is shown below.



Eliminating a division by zero

Another application of operations involving logical and numerical vectors in graphics is illustrated in the following example. We intent to plot the graph of the function $y = sin(x)/x$ in the range $-4\pi < x < 4\pi$. Obviously, the function is not defined at $x = 0$. An attempt to produce the vector $y = sin(x)/x$, as shown below, produces an error:

```
» x = [-4*pi:pi/20:4*pi];
» y = sin(x)./x;
Warning: Divide by zero.
```

One way to avoid this situation is to substitute the zero elements in *x* with MATLAB's constant *eps*, the small value used by MATLAB for rounding results down to zero. The default value of *eps* can be found by using:

```
» eps
ans = 2.2204e-016
```

The following statement will replace those zero elements in *x* with the value *%eps*:

```
» x = x + (x==0)*eps;
```

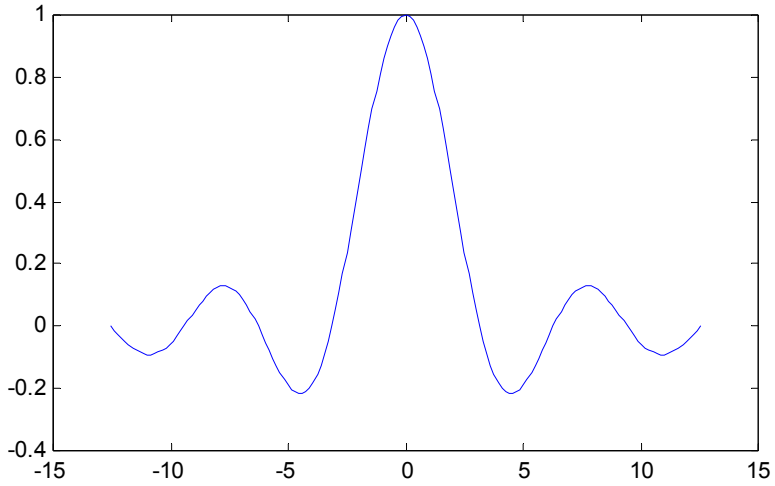
Evaluating the function *sin(x)/x* will not produce an error in this situation, since we no longer have zero elements in *x*:

```
» y = sin(x)./x;
```

A plot of the function is produced by using:

```
-->plot(x,y)
```

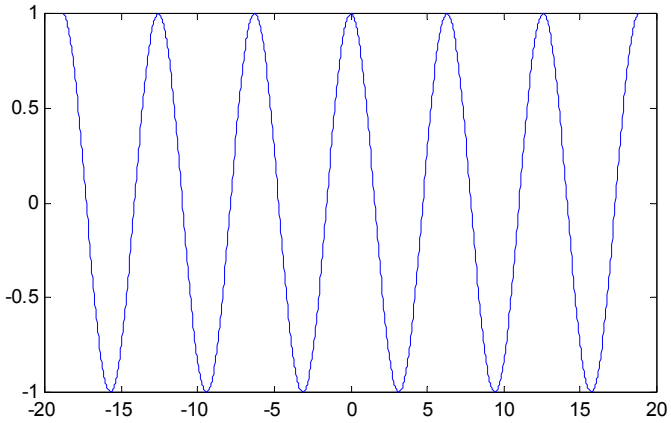
The resulting graph is shown next:



Producing a discontinuous graph by eliminating negative values

The function *cos(x)* in the range $-6\pi < x < 6\pi$ has positive and negative values. A plot of this function is shown next:

```
-->x = [-6*%pi:%pi/100:6*%pi];
-->y = cos(x);
-->plot(x,y)
```

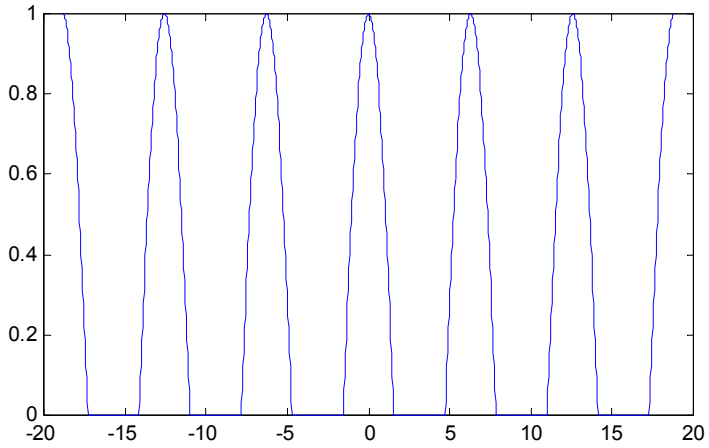


Suppose that we are interested in showing only the positive values of the graph while keeping the same range of *x*. One way to do this is by replacing all values in *y* that are negative with a zero. Here is a MATLAB statement that will produce such result:

```
» y = y.*(y>0);
```

Positive values of *y* are multiplied by one (thus preserving their values). On the other hand, negative values of y are multiplied by zero. A plot of the resulting vector *y* versus the original domain *x* is shown next:

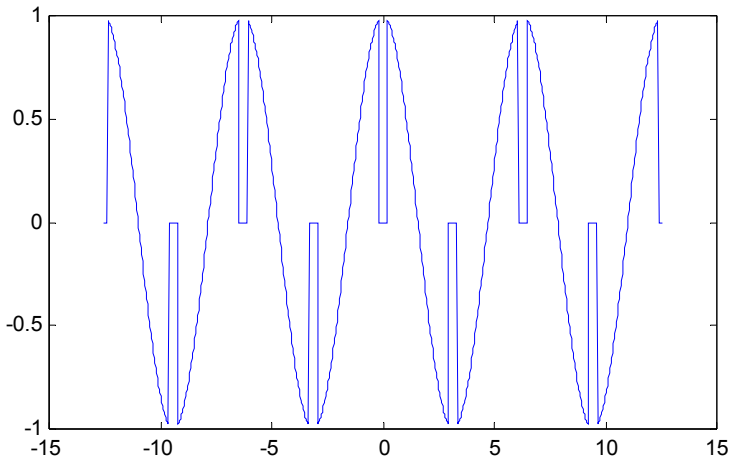
```
» plot(x,y)
```



Clipping the tips of the cosine wave

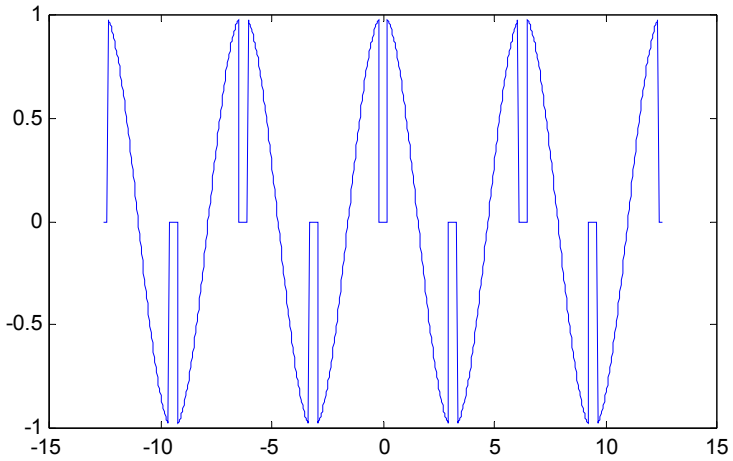
In this example we use an operation similar to the one used above to clip the tips off the cosine wave crests so that values such that $-0.98 < y < 0.98$ are kept, while those outside of that interval are converted to zero. The following statements will produce the desired result in the interval $-6\pi < x < 6\pi$:

```
» x = [-4*pi:pi/100:4*pi];
» y = cos(x);
» y = y.*(y>=-0.98 & y<=0.98);
» plot(x,y)
```



A graph that preserves the tips of the cosine wave crests while reducing the remaining values to zero is produced next:

```
-->x = [-4*%pi:%pi/100:4*%pi];
-->y = cos(x);
-->y = y.*(y<=-0.98 | y>=0.98);
-->plot(x,y)
```


1
0.5
0
-0.5
-1
-15
-10
-5
0
5
10
15