Appendix C

Laboratory Exercises Solutions

C.1 Introduction to Matlab Solution

C.1.1 In-lab section

1. (a) `[1 2 3 4 5]` is 1×5, a row vector. `[1:5]` is 1×5, a row vector. `1:5` is 1×5, a row vector. `[1:1:5]` is 1×5, a row vector. `[1:-1:-5]` is 1×7, a row vector. `[1 2; 3 4]` is 2×2, a square matrix. `[1; 2; 3; 4]` is 4×1, a column vector.

   (b) A constant 2×3 matrix:

   ```
   M = [11 12 13 ; 21 22 23]
   ```

   Matrices can also be constructed piecemeal. Here is row-by-row and column-by-column construction:

   ```
   row1 = [ 11 12 13 ]; row2 = [ 21 22 23 ];
   mat = [ row1 ; row2 ]
   col1 = [ 11 ; 21 ]; col2 = [ 12 ; 22 ]; col3 = [ 13 ; 23 ];
   mat = [ col1 col2 col3 ]
   ```

   We can also do this with expressions that return matrices:

   ```
   [ rand(1,3) ; ones(1,3) ]
   [ zeros(2,1) randn(2,1) ones(2,1) ]
   ```

   Here is a matrix constructed by putting together a column vector and a square matrix:

   ```
   [ [ rand(1) ; pi ] eye(2) ]
   ```

   (c)
   ```
   >> size(1:0.3:10)

   ans =

        1    31
   ```

```
>> size(1:1:-1)

ans =

     1     0
```

The `size` function gives both the number of rows and the number of columns. Notice that in the second case, there are no columns, so this can be interpreted as an empty row vector. You can confirm this interpretation with the `length` function:

```
>> length(1:1:-1)

ans =

     0
```

Now, the array constructor pattern

```
array = start : step : stop
```

returns an array of numbers between `start` and `stop` that begin with `start` and are incremented by `step`. If `stop` is less than `start`, then a zero-length array is returned. Otherwise, the result is an array of length

```
floor((stop-start)/step) + 1
```

2. (a) Here is a for loop that computes the sum:

```
>> x = 0
for i = 0:25
x = x + i;
end


x =

     0

>> x

x =

   325
```

Notice that if we leave off the semicolon above we will see all the partial sums.

(b)
```
>> sum(0:25)

ans =

   325
```

(c)
```
>> sin(0:pi/10:pi)

ans =

  Columns 1 through 7

    0    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511

  Columns 8 through 11

    0.8090    0.5878    0.3090    0.0000
```

(d)
```
>> n = sin(0:pi/10:pi);
>> n.*n

ans =

  Columns 1 through 7

    0    0.0955    0.3455    0.6545    0.9045    1.0000    0.9045

  Columns 8 through 11

    0.6545    0.3455    0.0955    0.0000
```

Note that `n^2` will not work (it triggers an error message). A vector cannot be squared in this way.

3. (a)
```
>> x = zeros(36);
>> x(18) = 1;
>> subplot(2,1,1), plot(x)
>> subplot(2,1,2), stem(x)
```
produces the plots shown in figure C.1.

(b) The frequency is 5 Hz, or $2\pi \times 5$ radians per second. The period is 1/5 second, and there are 10 cycles in the 2-second interval $[-1, 1]$. The value at zero is 0.5.

(c) The following sequence of commands yields the plot in figure C.2.

```
>> frequency = 8000;     % sampling frequency in Hertz
>> period = 1/frequency; % sampling period in seconds
>> t = -1:period:1;      % sample times as a row vector
>> sampledSignal = sin(t,2*pi*5*t + pi/6);
>> plot(sampledSignal)
```

Notice that in the `plot` command we specify both the sample times and the values of the signal. This results in a plot with the horizontal axis correctly labeled in units of

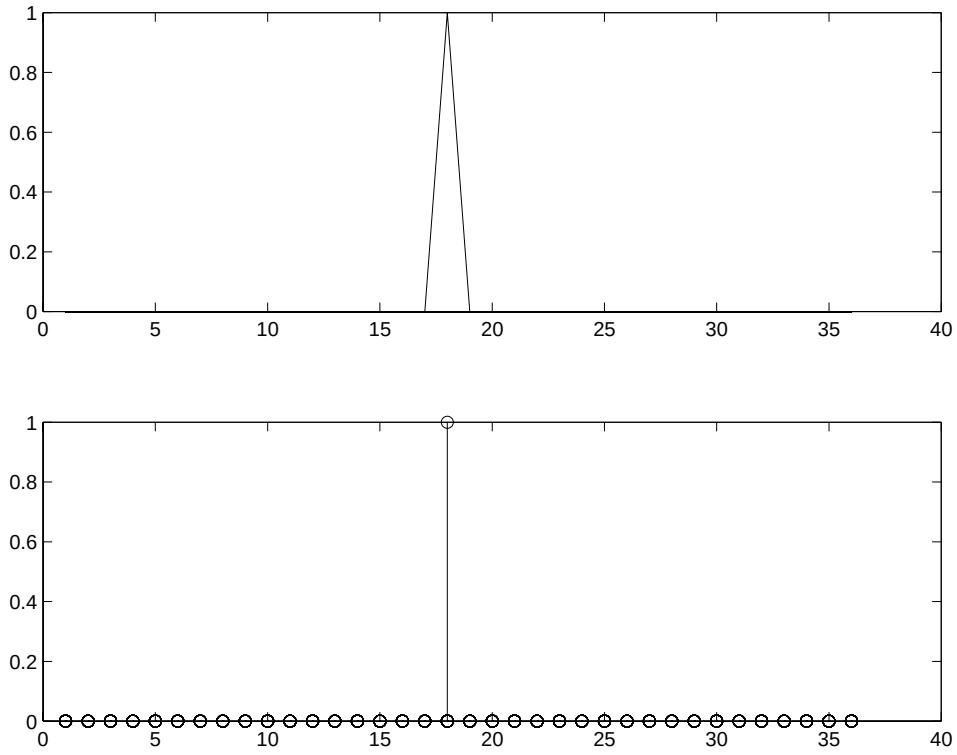

Figure C.1: Two plots of an impulse, generated by `plot` and `stem`.

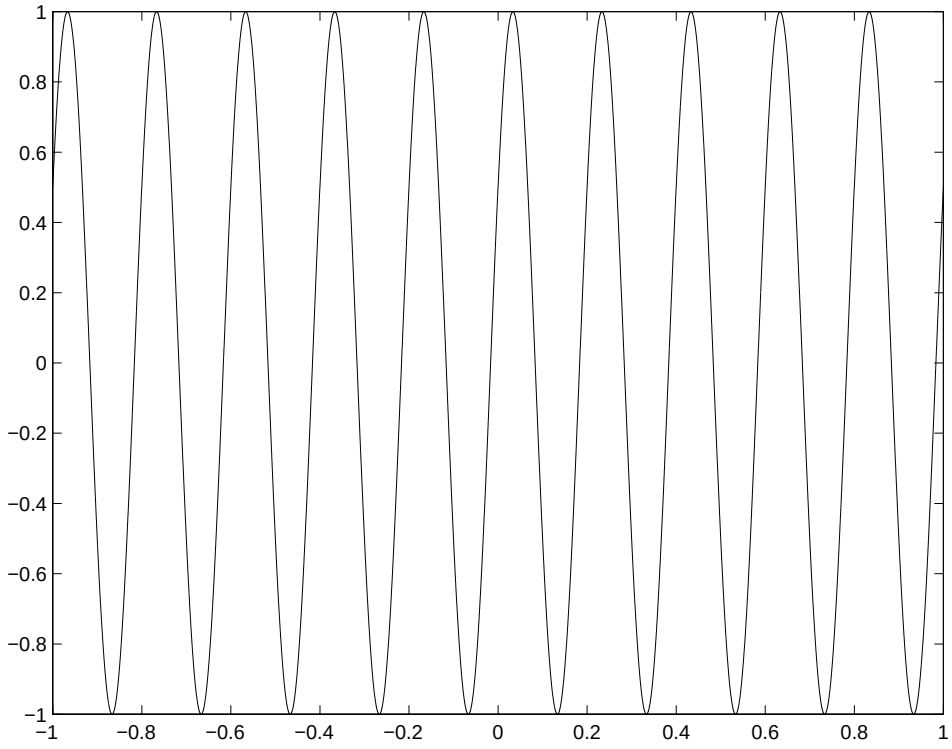

Figure C.2: A sinewave at 5 Hz sampled at 8 KHz.

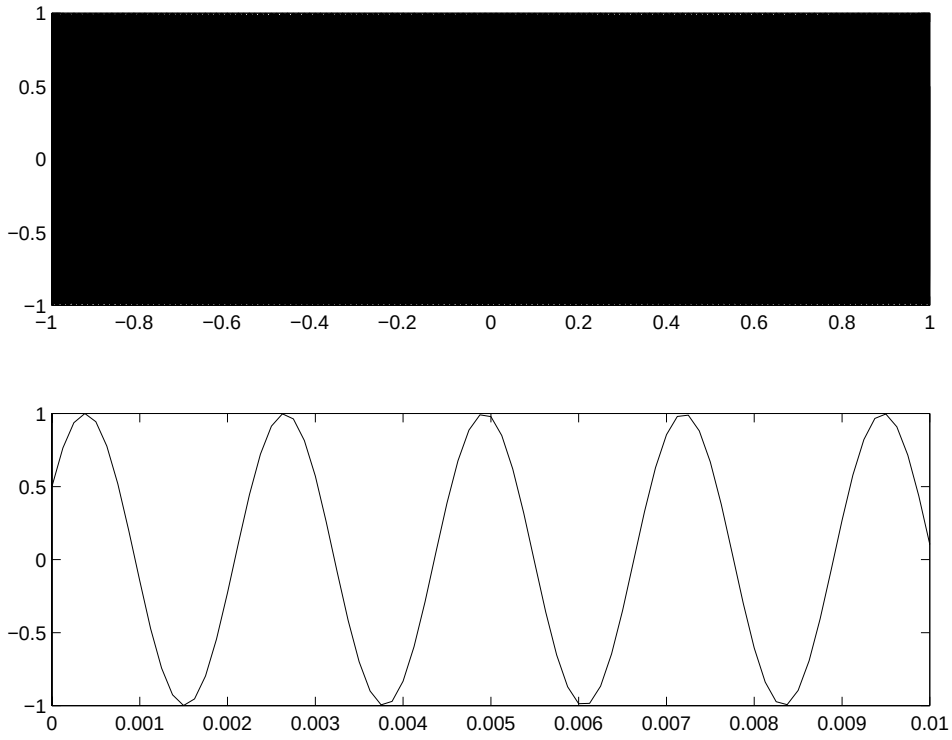

Figure C.3: A sinewave at 440 Hz sampled at 8 KHz.

seconds. There are a total of 16,000 samples. Also notice that all commands end in semicolons to prevent Matlab from displaying the 16,000 samples.

(d) Assuming we have executed the commands above, the following commands result in the plots shown in figure C.3.

```
>> sampledSignal = sin(2*pi*440*t + pi/6);
>> subplot(2,1,1), plot(t, sampledSignal);
>> t = 0:period:0.01;
>> sampledSignal = sin(2*pi*440*t + pi/6);
>> subplot(2,1,2), plot(t, sampledSignal);
```

The upper plot is very hard to read because there are so many cycles that they all run together. The lower plot shows far fewer cycles.

(e) For the 10 msec sound segment, you hear just a click. This interval is very short. For the 2 second interval, you hear a tone with frequency 440 Hz, middle A in the western musical scale.

(f) The following command:

```
sound(0.5*sampledSignal,frequency)
```

results in a sound that is quieter, while

```
sound(2*sampledSignal,frequency)
```

is louder and distorted. It is no longer a pure tone. The sound produced by

```
sound(sampledSignal,frequency*2)
```

has a higher pitch, while

```
sound(sampledSignal,frequency/2)
```

has a lower pitch.