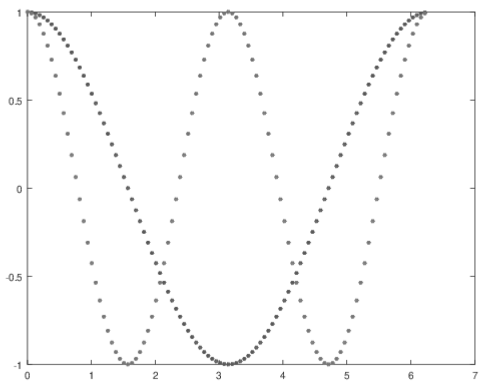
Analyzing frequencies in time data

Frequency is defined as number of cycles per unit time. Surely different input data vectors can come from different frequencies:

```
> t = linspace(0, 2*pi, 101)(1:end-1)';     # partition of [0, 2 pi), 100 points
> length(t)           # we kept only 100 out of 101 data points
> t(end)              # prints last value in vector t
> y1 = cos(t);
> y2 = cos(2*t);
> plot(t, [y1 y2], '.', "markersize", 12)
```

The resulting plot is at right. Not surprisingly, `y1` has a frequency half that of `y2`. It makes sense to say the two vectors have frequencies 1 and 2 respectively, but to do so inherently means we are taking the width of the interval $[0, 2\pi)$ as a single unit of time. If the number 1 on the t-axis corresponded to a single unit of time, then `y1` would have frequency $1/(2\pi) \doteq 0.159$ and the corresponding frequency for `y2` would be double that, at $2/(2\pi) \doteq 0.318$. This reflects the fact that `y2` has completed just 31.8% of a cycle by time $t = 1$.



The problem is that if sampled data comes to us only as a vector, all we may glean from that is the actual values, and how many values are provided. Take, for instance the data representing a sound recording of a touch tone phone call as the call is being placed.

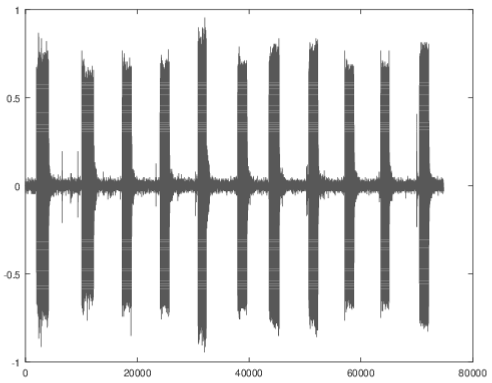
```
> load touchtone.dat
> whos
Variables in the current scope:

   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        D           201x151                  30351  uint8
        y             1x1                    74808  struct
```

When loaded, this file provided two saved pieces of data. Notice that `y` is of class `struct`. To see what `y` contains, we may type

```
> fieldnames(y)
ans =
{
  [1,1] = sig
  [2,1] = fs
}
```

and we may access either of these pieces as `y.sig` or `y.fs`. The former is our data vector which, before plotting, I convert from type `uint8` (8-bit integer) to `double` and rescale it.

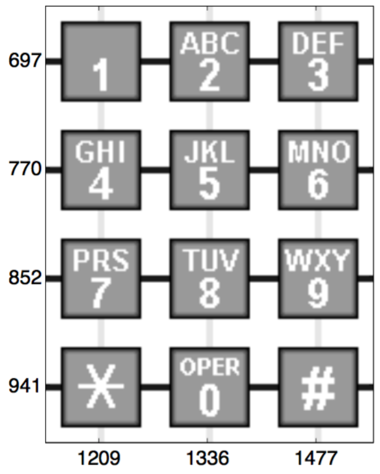
```
> fs  = y.fs;
> y = double(y.sig)/128;
> plot(y)
> length(y)
ans =   74800
```



This data vector, now rescaled, has 74,800 values all between (-1) and 1. That, in of itself, does not tell us what a single unit of time was taken to be (in this case it was 1 second), nor how many measurements were made during each time unit, the latter being the real crux of the matter. In this case, we have been given the sampling rate, too, the number I saved as `fs`, revealing that the first 8192 components of the vector were measured during the first time unit (second), and the whole recording lasts $74800/8192 \doteq 9.13$ s.

The plot itself looks like it consists of 11 different *pulses*, suggesting a phone number 1-xxx-xxx-xxxx was dialed. But if the first pulse represents the number 1, we would be hard-pressed to pick out visually what numbers the other pulses represent, or even if another pulse is a 1. To analyze pulses, we will

- isolate them, then
- take discrete (or finite) Fourier transforms.



The number `fs` will be used in determining which frequencies are present, and the graphic at right will help us decide which digit was pressed. Note that each button has a two-frequency signature. The number 4, for instance, employs sounds at 770 Hz (cycles per second) and 1209 Hz. (I wonder how those were chosen?)

First, the entire vector can be split into 11 subvectors, which I make the rows of a matrix `pulses` below.

```
> n = 74800/11
n =   6800
> pulses = reshape(y,n,11)';
```

For our purposes, rather than view separate parts (real vs. imaginary) of the components of the

DFT, we can simply view the modulus (absolute value) of each component. For the first pulse, that would be

```
> plot( abs(fft(pulses (1,:) )) )
```

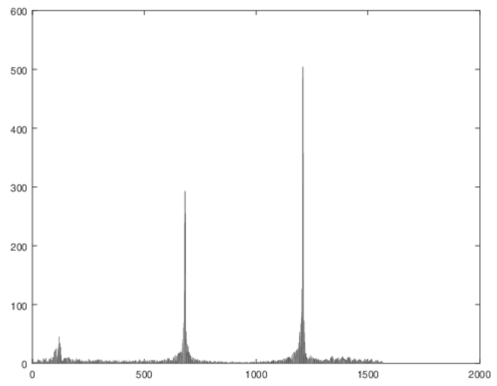
There are some clear spikes in the plot. But the frequencies are being identified as running from 0 to 6800 (or 0 to 3400, if we stop at the half-way/Nyquist point), and these are incorrect. We must rescale those frequencies by the factor `fs/n`. In what follows, we create a vector `f` of these rescaled frequencies, and plot the absolute values of the DFT only up to the Nyquist point.

```
> Y = fft( pulses (1,:) );
> f = (0:n/2) * fs/n;
> pow = abs( Y(1:n/2+1) );
> plot([f;f], [0*pow;pow])
```

We zoom in a bit around the regions of ”peak power”.
The plot is given at right.

```
> plot([f(1:1300);f(1:1300)],[0* pow(1:1300);pow(1:1300)])
```

It seems to confirm the first pulse as being a ’1’.


600
500
400
300
200
100
0
0
500
1000
1500
2000