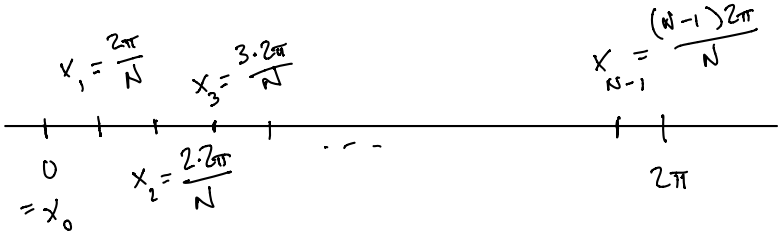
Math W84, Mon 18-Jan-2016 -- Mon 18-Jan-2016

Monday, January 18th 2016
---
day08, Mo
Topic:: Rhythm

Project details:
- worked on in your group
- collective presentation
  offered on either Mon or Tues, Day 14 or 15 (chosen randomly?)
  35-45 minutes
- topic chosen as a team
  to investigate on own
  may extend topics covered, but not be a rehashing of them
  must be approved by professor
- taught to class at appropriate level with
  motivation
  helpful background
  examples of use
  involving interesting math, yet musical relevant
  uses examples from both, most likely with A/V support
- possible resources:
  "Music, a Mathematical Offering", by Dave Benson
  "Musimathics: The Mathematical Foundations of Music", by Gareth Loy
  "Rhythm and Transforms", by William Sethares
  "Tuning, Timbre, Spectrum, Scale", by William Sethares
  "A Geometry of Music", by Dmitri Tymoczko



The x_i serve as left endpoints in a Riemann sum approximation of

$$\int_0^{2\pi} f(x)e^{-kix}\,dx.$$

In Homework we took $\varphi_n(x) = e^{2\pi i n x/\ell}$. Setting $\ell = 2\pi$, these simplify to $\varphi_n(x) = e^{inx} = \cos(nx) + i\sin(nx)$ with complex conjugate $\overline{\varphi_n(x)} = e^{-inx} = \cos(nx) - i\sin(nx)$

Motivation for the DFT

Last time we studied the complex exponential Fourier series expansion of ℓ-periodic functions with $\ell > 0$. To simplify the appearance of our complex exponential functions, we assume $\ell = 2\pi$, giving us basis functions

$$\mathcal{B} = \{\ldots, \phi_{-3}(x), \phi_{-2}(x), \phi_{-1}(x), \phi_0(x), \phi_1(x), \phi_2(x), \ldots\},$$

where each $\phi_k(x) := e^{ikx}$, $k = 0, \pm 1, \pm 2, \ldots$. The resulting complex exponential Fourier series is

When f is real-valued, $c_{-1} = \overline{c_1}$ (and $c_{-k} = \overline{c_k}$). These terms constitute the fundamental frequency.

$$\sum_{k=-\infty}^{\infty} c_k \phi_k(x) = \cdots + c_{-2}e^{-2ix} + c_{-1}e^{-ix} + c_0 + c_1 e^{ix} + c_2 e^{2ix} + c_3 e^{3ix} + \cdots \quad (1)$$

(c_0: constant; these terms constitute the 2nd harmonic)

with

$$c_k = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-kix}\,dx. \quad (2)$$

The Octave script `complexCoeffs.m` can be used to carry out approximate integration to determine a finite collection of the c_k using formula (2).

Some observations:

(i) For real-valued functions $f(x)$, $2|c_k|$ (twice the modulus of the complex number c_k), gives the strength of the k^{th} harmonic (comprised of the e^{-ikx} and e^{kix} terms) in (1).

(ii) Assuming $f(x)$ is real-valued, let's truncate the series (1), this time keeping terms from $k = 0$ up to some integer $N > 0$:

$$\sum_{k=0}^{N-1} c_k\phi_k(t), = c_0 + c_1 e^{ix} + c_2 e^{2ix} + \cdots + c_{N-1} e^{(N-1)ix} \quad (3)$$

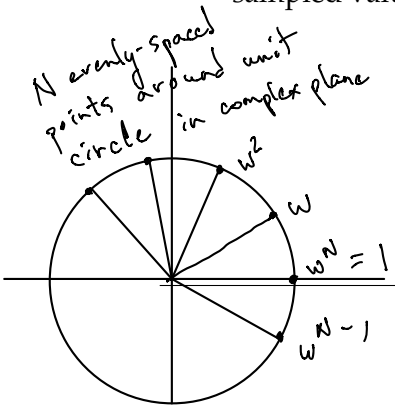
We might approximate the c_k in (2) using a Riemann sum. Specifically, if we break up the interval $[0, 2\pi)$ into N subintervals with left endpoints $x_0 = 0$, $x_1 = 2\pi/N$, $x_2 = 2 \cdot 2\pi/N, \ldots$, $x_{N-1} = (N-1)\cdot 2\pi/N$, then the left-hand Riemann sum approximation of c_k, $k = 0, \ldots, N-1$, is — see above

$$c_k \approx \frac{1}{2\pi}\sum_{j=0}^{N-1} f(x_j)e^{-kix_j}\cdot \underbrace{\frac{2\pi}{N}}_{\Delta x} = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)e^{-ki2\pi j/N} = \frac{1}{N}\sum_{j=0}^{N-1}\left(e^{-i2\pi/N}\right)^{kj} f(x_j).$$

Here, the complex number $w = e^{i2\pi/N}$ is called a **principle N^{th} root of unity**, and the vector $\mathbf{f} = \langle f(x_0), f(x_1), f(x_2), \ldots f(x_{N-1})\rangle$ represents the function $f(x)$ sampled at evenly-spaced points through one cycle $[0, \ell)$. We can obtain the approximate vector of coefficients $\mathbf{c} = \langle c_0, c_1, c_2, \ldots, c_{N-1}\rangle$, neglecting the factor N, all at once, by multiplying the vector of sampled values of $f(x)$ by the corresponding matrix

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & \overline{w} & \cdots & \overline{w}^k & \cdots & \overline{w}^{N-1} \\ 1 & \overline{w}^2 & \cdots & \overline{w}^{2k} & \cdots & \overline{w}^{2(N-1)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & \overline{w}^{N-1} & \cdots & \overline{w}^{(N-1)k} & \cdots & \overline{w}^{(N-1)^2} \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \end{bmatrix}. \quad (4)$$

This vector is the time-domain signal/tone $f(t)$ sampled at points $t = x_0, x_1, \ldots, x_{N-1}$.



The discrete Fourier transform

For a given positive integer N, let

$$\mathbf{F}_N = \begin{bmatrix} 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & w & \cdots & w^k & \cdots & w^{N-1} \\ 1 & w^2 & \cdots & w^{2k} & \cdots & w^{2(N-1)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & w^{N-1} & \cdots & w^{(N-1)k} & \cdots & w^{(N-1)^2} \end{bmatrix}$$

be called the **Fourier matrix**. The matrix in (4) is the conjugate transpose of $\mathbf{F}_N$, labeled $\mathbf{F}_N^{\mathrm{H}}$. We call the product

$$\mathbf{F}_N^{\mathrm{H}}\mathbf{f} = \begin{bmatrix} 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & \overline{w} & \cdots & \overline{w}^k & \cdots & \overline{w}^{N-1} \\ 1 & \overline{w}^2 & \cdots & \overline{w}^{2k} & \cdots & \overline{w}^{2(N-1)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & \overline{w}^{N-1} & \cdots & \overline{w}^{(N-1)k} & \cdots & \overline{w}^{(N-1)^2} \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \end{bmatrix}.$$

the **discrete Fourier transform**, or DFT, of $f(t)$.

While you can find the DFT through matrix multiplication, as described above, once the function $f(t)$ is sampled to produce the vector $\mathbf{f}$, it is easier and more efficient to obtain the vector $\mathbf{c} = \mathbf{F}_N^{\mathrm{H}}\mathbf{f}$ using the **fast Fourier transform** (FFT) algorith developed by J.W. Cooley and John Tukey in 1965.

Example 1:

Suppose we sample the 2π-periodic function $f(x) = \sin(x) - 2\cos(x)$ at $N = 10$ locations in the cycle $[0, 2\pi)$ using the command

```
> xs = (0:2*pi/10:2*pi)(1 : end-1);
> vec_f = sin(xs) - 2*cos(xs);
```

sampling at 10 evenly-spaced points in $[0, 2\pi)$ makes it possible to discover spectrum for 1st through 4th harmonics

sampled signal f, contains only fundamental

You can view the sampled signal as a sequence of points using

```
plot(xs, vec_f, 'k.', "markersize", 20)
```

You can take the FFT using

```
> fft(vec_f)
```

Notice the numbers $c_0, \ldots, c_9$ are generally complex. Still, you can obtain reletive strength of the overtones by viewing

```
> plot(0:9, abs(fft(vec_f)), 'k.', "markersize:, 20)
```

graph of the spectrum, shows power only in 1st harmonic (of dots 1, 2, 3, 4)

at $k=9$

Because of aliasing, the latter peak is just a mirroring the peak at $k = 1$, and conclude that the only frequency present is the fundamental.

■

Example 2:

no fundamental present, just 3rd and 8th harmonics

Suppose we sample the 2π-periodic function $f(x) = \sin(3x) - 2\cos(8x)$. The 3rd and 8th ~~overtones~~ harmonics are present, so we should have a high enough sampling rate to be above twice the highest overtone present, say $N = 20 > 2(8)$: (enables us to discover spectrum for first 9 harmonics)

```
> xs = (0:2*pi/20:2*pi)(1 : end-1);
> vec_f = sin(3*xs) - 2*cos(8*xs);
```

Look at the FFT using

```
> plot(0:19, abs(fft(vec_f)), 'k.', "markersize:, 20)
```

We detect the presence of the 3rd and 8th overtones, and that the 8th overtone is there in twice the measure.

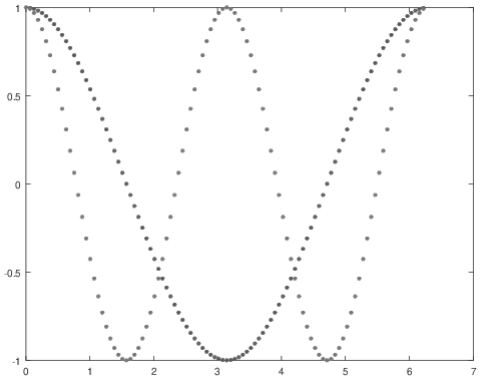
■

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.

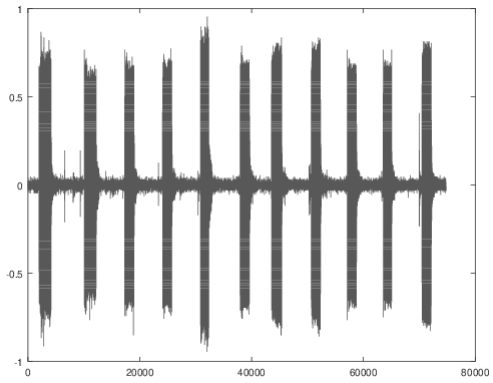
```
> load touchtone.dat
> whos
Variables in the current scope:

   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.



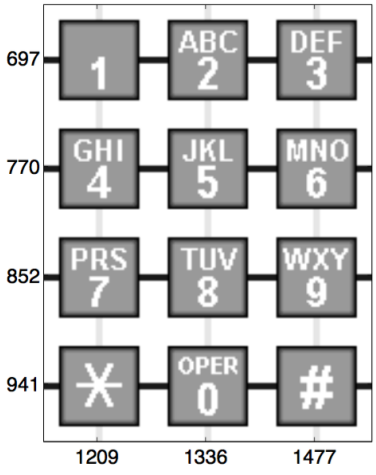

```
> fs  = y.fs;
> y = double(y.sig)/128;
> plot(y)
> length(y)
ans =   74800
```

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 ≐ 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

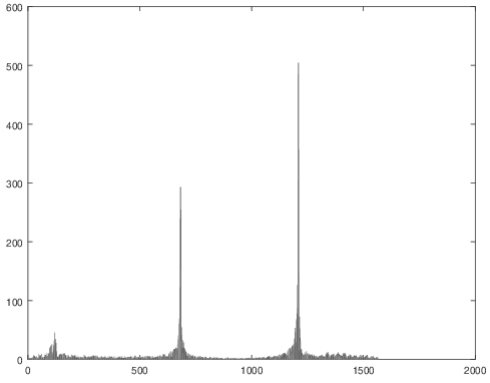
```
> plot(  abs( fft (pulses (1,:) ))  )
```

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'.