Brain Waves and the EEG

Signals are sent through the brain using both chemical and electrical means. The synchronized electrical activity of individual neurons adds up to something big enough to detect on from outside the head. To measure it, we use a set of electrical nodes called an electroencephalogam (EEG). The measured activity reflects different states of the brain which in turn tell us something about the mindset of the person. Our goal in this module is to decompose an EEG signal into its different frequencies, which is intuitively the most meaningful piece of information.

1 All About Waves

Brainwaves have complex shapes that are not easily interpreted. In order to study these waves, we need to develop some mathematical tools that will tell us about different waves. To outline, we begin by talking about pure (sine or cosine) waves, then move to the trapezoid rule for estimating area under a curve. Next, we develop Fourier analysis for picking out the frequencies in a jumbled signal, and finally use these tools to create spectrograms, which allow us to track different frequencies over time.

1.1 Sine Waves

The sine wave is a mathematical function. It describes many physical phenomena, including sound waves and oscillation. It looks just like a wave. MATLAB uses the `sin` function to make sin waves. For example, to make Figure 1, we use the code:

```
t = 0:.01:1;
y = sin(2*pi*t);
plot(t,y);
```

The sine wave is defined by the lengths and angles of a triangle. Run `sincirc.m` (copied below) to see how the sine and cosine values relate to the angle φ of the triangle. As you can see, if φ is the angle of a right triangle with hypotenuse 1 (illustrated by the circle) , $\sin(\varphi)$ is the height of the triangle and $\cos(\varphi)$ is the base of it:

```
% sincirc.m
%
% sincirc.m illustrates the relation of the sin and cosine waves to the circle.

%define parameters
Nturns = 2;
steps_per_turn = 9;
step_inc = 2*pi/steps_per_turn;
```

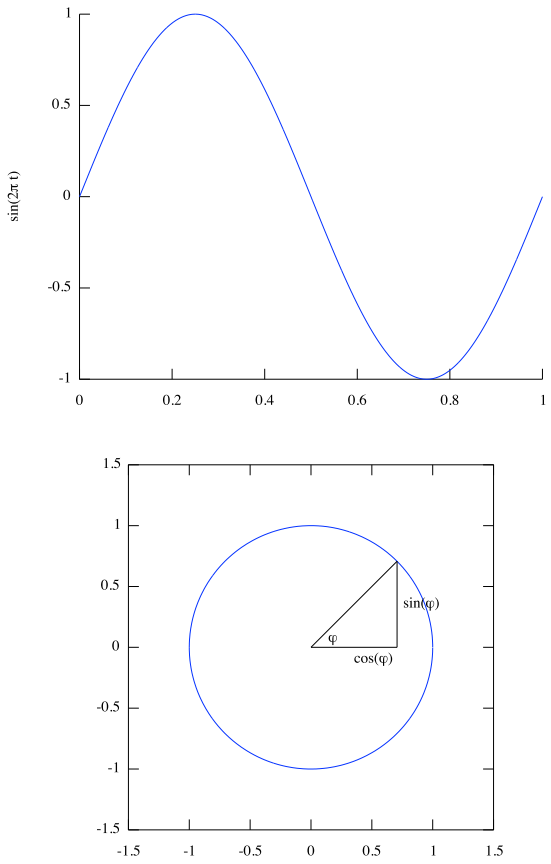

Figure 1: Left: A sine wave. Right: the relation of sine and cosine to the triangle.

```
%set up points for circle
circ_x = cos(0:.01:2*pi);
circ_y = sin(0:.01:2*pi);
axis equal

%loop over triangles with different angles
for n = 1:Nturns * steps_per_turn;
    phi = n * step_inc + pi/4;

    %plot circle, then triangle, then text
    plot(circ_x, circ_y);
    axis([-1 1 -1 1] * 1.5);
    line([0 cos(phi)], [0 sin(phi)]);
    line([1 1] * cos(phi), [0 sin(phi)]);
    line([0 cos(phi)], [0 0]);
    text(cos(phi)/2 , -.1*sign(sin(phi)),'cos(\varphi)')
    text(cos(phi) + .1*(sign(cos(phi))-.5), sin(phi)/2, 'sin(\varphi)')
    text(cos(phi)*.2, sin(phi)*.1,'\varphi');
    pause(.5);
end
```

1.2 Characteristics of the Sine Wave

The sin wave has three primary characteristics:

1. Frequency measures how often a wave passes. We can make a wave with frequency ω by writing:

$$\sin(2\pi\omega t)$$

Aside: The wave described by $\sin(t)$ has frequency $1/2\pi$. If we instead write $\sin(2\pi t)$, we will have a wave with frequency 1, which is easier to work with.

We can express the same information in terms of wavelength. Wavelength is how close neighboring waves are to each other. It is inversely proportional to frequency, which means that the higher the frequency, the smaller the wavelength. If ℓ is wavelength, we write this in the following way:

$$\ell = \frac{1}{\omega}$$

A wave with wavelength ℓ is written as $\sin(\frac{2\pi t}{\ell})$.

2. Amplitude measures how high the wave is. We can make a wave of amplitude a by writing:

$$a \cdot \sin(2\pi t)$$

Sometimes a will be negative. In this case, we still say that the wave has amplitude a, but note that the function will be flipped across the x-axis.

3. Phase describes how far the wave has been shifted from center. To create a wave with phase p, we write:

$$\sin(2\pi\omega t + p)$$

Since a sin wave repeats every 2π, the following is always true (that is, for any φ):

$$\sin(\varphi) = \sin(\varphi + 2\pi)$$

Aside: A cosine wave is a sine wave shifted back by a $\pi/2$, a quarter of the standard wave:

$$\cos(\varphi) = \sin(\varphi + \pi/2)$$

Every sine (cosine) wave can be described completely by these characteristics. These are shown in the figure below (phase = 0 for simplicity):

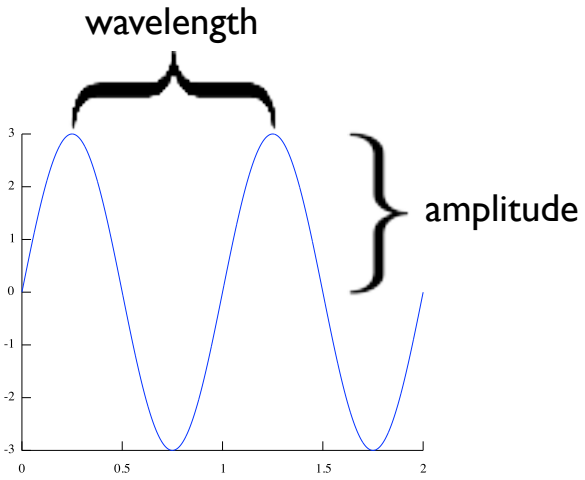

Figure 2: A sin wave

To code Figure 2 in MATLAB, use:

```
t = 0:.01:1;
amp = 3;
freq = 1;
phase = 0
y = amp*sin(freq*2*pi*t + phase);
```

1.2.1 Hearing Sine Waves

The sine wave represents a pure tone. To hear one, we use the MATLAB function `sound()`, which converts a vector into sound. Find the frequency that is specified, and compare to human range of hearing. Should we be able to hear this sound?

```
freq = 1000;
samp_rate = 1e4;
duration = 1;
samples = 0 : (1/samp_rate) : duration;
sound_wave = sin(2 * pi * samples * freq);
sound(sound_wave, samp_rate);
```

Enter the above code into Matlab to hear the sound.

1.3 Adding Sine Waves

We can add together multiple sin waves to accomplish different shapes. For example, if we add the wave $[4 \cdot \sin(2\pi t)]$ and the wave $[\sin(6 \cdot (2\pi t))]$ we get:

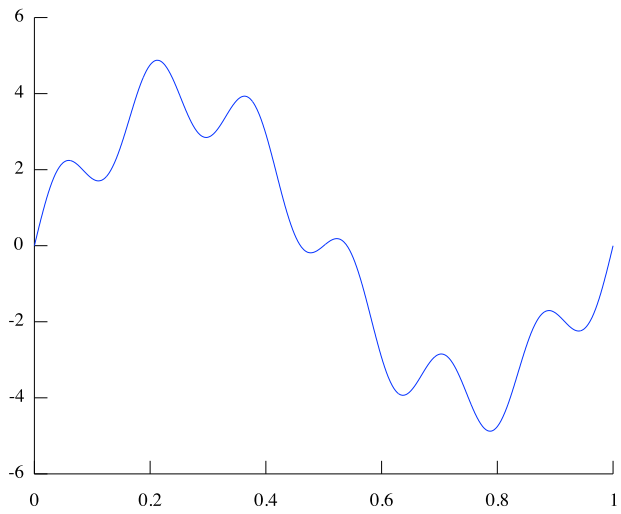

Figure 3: A compound wave: $4 \cdot \sin(2\pi t) + \sin(6 \cdot (2\pi t))$

Look at the figure and try to identify the effect of each wave. The first wave has frequency 1 and amplitude 4. This accounts for the large up-and-down motion that only goes through one cycle in the figure. The second wave has frequency 6 (wavelength $\frac{1}{6}$) and amplitude 1. This accounts for the small wiggles

that happen many times in the figure.

Figure 4 is implemented in MATLAB with the following code:

```
t = 0:.01:1;
y = 4*sin(2*pi*t)+sin(6*2*pi*t);
plot(t,y)
```

A wave of any shape can be expressed as a sum of sin and cosine waves, although it may take infinitely many. In the end of this module we will find interesting uses for the this fact.

Exercises

1.1 Figure 1.3 two shows a sum of two sine waves (with phase 0). Try to replicate it, and report what the amplitudes and frequencies of each wave is.

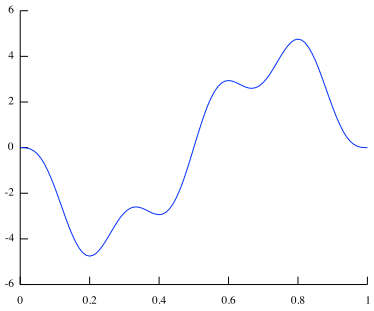

Figure 4: The compound wave for exercise 1.1

1.2 Use the identities $\cos(\varphi) = \sin(\varphi + \pi/2)$ and $\sin(\varphi) = \sin(\varphi + 2\pi)$ to solve for p in the following equations by making the above substitutions. Check your answer visually against the figures of sine and cosine waves:

Equations:

1. $\cos(\varphi + 1) = \sin(\varphi + p)$
2. $\sin(\varphi) = -\sin(\varphi + p)$
3. $\tan(\varphi) = \frac{\sin(\varphi)}{\sin \varphi + p}$
4. $\cos(\varphi) = -\tan(\varphi + p) \cdot \cos(\varphi - p)$

($\tan(\varphi)$ is defined in exercise 1)

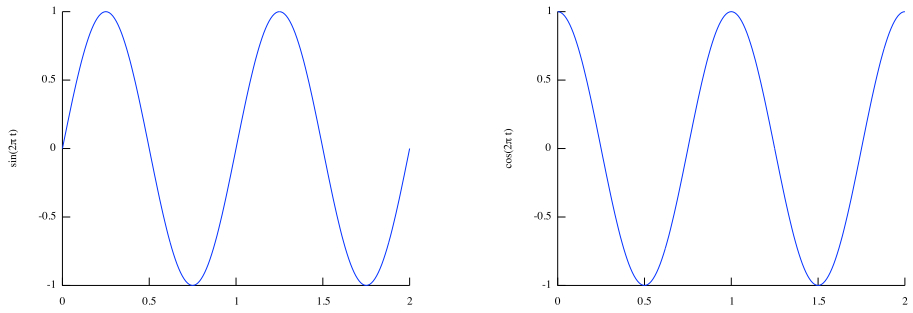

Figure 5: Sine and Cosine waves.

2 Finding area under a curve (Integration)

2.1 Trapezoid Rule

A useful tool for analyzing curves is finding the area underneath them. When we have an unknown combination of waves, we can estimate the area under the curve using the *trapezoid rule*. We will use $f(t) = \sin(t)$ as an example. If we were to estimate part of the area under the curve with one trapezoid, we might do the following:

We have labeled the two heights, h_1 and h_2, and the length of the base b. The area of the square is:

$$\text{area of square} = (\text{base}) \times (\text{height}) = b \cdot h_1$$

The area of the top triangle is:

$$\text{area of triangle} = \frac{(\text{base}) \times (\text{height})}{2} = \frac{b \cdot (h_2 - h_1)}{2}$$

The total area of the trapezoid is then:

$$\text{area of trapezoid} = b \cdot h_1 + \frac{b \cdot (h_2 - h_1)}{2} = \frac{b \cdot (h_2 + h_1)}{2}$$

If we know that the two points on the x-axis are t_1 and t_2, then $b = t_2 - t_1$. In the figure above, $t_1 = .5$ and $t_2 = 1.5$. Then the heights follow from the function: $h_1 = f(t_1) = \sin(t_1)$ and $h_2 = f(t_2) = \sin(t_2)$. Thus in general, the area of a trapezoid approximating the area under f between the points t_1 and t_2 is:

$$\text{area of trapezoid} = \frac{b \cdot (h_2 + h_1)}{2} = (t_2 - t_1)\frac{f(t_1) + f(t_2)}{2} \tag{1}$$

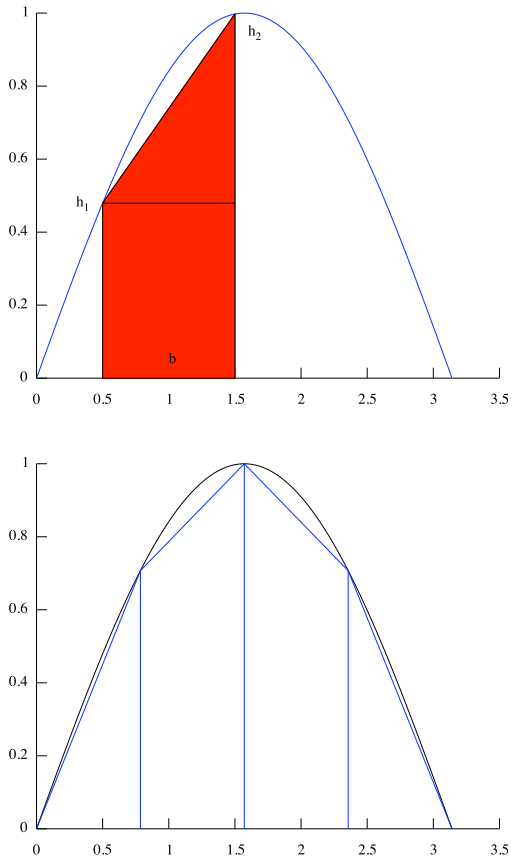

Figure 6: Applying the trapezoid rule to a sine wave.

In order to get a good estimate, we split up the domain of the function $f(t)$ into several intervals $[t_i, t_{i+1}]$. For each interval, we calculate the area of the trapezoid that approximates the area under that curve. For example, we could approximate $f(t)$ over $[0, 1]$ using four equal intervals. This would look like:

In this case, our estimate would be

$$\text{approximate area of curve} = (t_2 - t_1)\frac{f(t_1) + f(t_2)}{2} + \cdots + (t_4 - t_3)\frac{f(t_3) + f(t_4)}{2}$$

As we take smaller and smaller intervals, our approximation will get better, because there will be less space between the trapezoids and the curve. We can prove that the trapezoid rule given 'order 2' convergence–that is, if we cut our intervals in half, our error gets four times smaller.
In the general case, if we split up the domain of the function at points $\{t_1, t_2, \ldots, t_n\}$, then the rule for the estimate is

$$\text{approximate area of curve} = (t_1 - t_2)\frac{f(t_1) + f(t_2)}{2} + \cdots + (t_n - t_{n-1})\frac{f(t_{n-1}) + f(t_n)}{2} \tag{2}$$

This formula can be further reduced, which is the subject of Exercise 2.1.

2.2 Coding the trapezoid rule

Here we present a code that uses the trapezoid rule to find the area under any function we provide. We need a vector `x` that holds the values of the domain, for example `x = 0:.01:pi`. We then need a vector `y` that holds the function values at those `x` points, for example `y = sin(x)`.

```
function curve_area = mytrapz(x, y, fast)
% function curve_area = mytrapz(x, y, fast)
%
% mytrapz.m performs the trapezoid rule on the vector given by x and y.
%
% Input:
%   x - a vector containing the domain of the function
%   y - a vector containing values of the function corresponding to the
%   values in 'x'

curve_area = 0;

 %loop through and add up trapezoids for as many points as we are given
for n = 2 : numel(x)
```

We start the code with zero area under the curve, since we haven't counted anything yet. Then we create a `for` loop to count each triangle individually. As we see above, more trapezoids leads to better answers, so we want to use as

many trapezoids as we possibly can. In this situation, that means using every point in `x` and `y`. The function `numel` simply counts the number of elements in `x`. We then calculate the area of the current triangle (within the loop):

```
    height = (y(n) + y(n-1))/2;     %average height of function across interval
    base = x(n) - x(n-1);           %length of interval
```

The area, as in Equation (1), is base times height. The height is the average of the height of the function at the two corner points. The length of the interval is the difference between the two corner points.

Lastly, we compute the area of the trapezoid and add it to our answer:

```
   trap_area = base * height;      %area of trapezoid
   curve_area = curve_area + trap_area;    %add to continuing sum
end
```

The `end` statement closes the loop over triangles.

To check the accuracy of our code, we test it with many examples, one of which is shown here. The function `trapz()` is a built-in function that performs the same operation:

```
>> x = 0:.01:pi;
>> y = sin(x);
>> mytrapz(x,y)

   ans =  1.99998206504367

>> trapz(x,y)

   ans =  1.99998206504367
```

Additionally, we know that $\int_0^\pi \sin(x)dx = 2$, so the answer we are getting is very close to the true value.

A simpler implementation of the code is included at the end, if we specify a third argument '`fast`':

```
    %alternate (fast) implementation
    curve_area = sum(y(2:end-1).*(x(3:end) - x(1:end-2)));
    curve_area = curve_area + y(1)*(x(2) - x(1)) + y(end)*(x(end) - x(end-1));
    curve_area = curve_area/2;
```

The explanation for this code is left as exercise 2.1.

2.3 Integration and the Trapezoid Rule

The trapezoid rule is a way of estimating the area under a function. This is exactly what we call an *integral*. The integral of $f(x)$ from a to b looks like this:

$$\int_a^b f(x)dx$$

For well-behaved functions, evaluation of the trapezoid rule approaches the true value of the integral (true area underneath the curve) as we evaluate smaller and smaller intervals. Of course, there are other ways to characterize integration, namely the **Fundamental Theorem of Calculus**. For more on this, see the Connexions module *Integration, Average Behavior: The Fundamental Theorem of Calculus*.

2.4 Exercises

2.1 Show that Equation 2 is equivalent to the "fast" version of the code:

$$\text{approximate area of curve} = \sum_{i=2}^{n-1} f(t_1)(x_{i+1}-x_{i-1})+f(t_1)(x_2-x_1)+f(t_n)(x_n-x_{n-1})$$

2.2 Test the accuracy of `mytrapz.m` for $y = \sin(x)$ (or some other function) as you increase the number of trapezoids. Plot your result on a `loglog()` plot: on the x-axis, number of trapezoids, and on the y-axis, error.

3 Fourier Analysis

We now introduce the key mathematical ideas that will allow us to break down signals into their component frequencies.

3.1 Building the Tools

Suppose the function $f(t)$ consists of several sine waves added together:

$$f(t) = a_1 \sin(1 \cdot 2\pi t) + a_2 \sin(2 \cdot 2\pi t) + \cdots + a_n \sin(n \cdot 2\pi t)$$

Suppose we know $f(t)$ over some interval, say $[0, 1]$, and we want to find a_j. Before we jump into this, we need to calculate the value of a certain integral. For integers h and k, with $h \neq k$,

$$
\begin{aligned}
&\int_0^1 \sin(h \cdot 2\pi t)\sin(k \cdot 2\pi t)dt \\
&= \int_0^1 \frac{1}{2}\left[\cos((h-k)\cdot 2\pi t) - \cos((h+k)\cdot 2\pi t)\right] dt && \text{trigononmic identity (substitution)} \\
&= \frac{1}{2(h-k)2\pi t}\sin((h-k)\cdot 2\pi t) - \frac{1}{2(h+k)2\pi}\sin((h+k)\cdot 2\pi t)\Big|_{t=0}^{1} && \text{integral of cosine is sine} \\
&= 0 && \text{evaluate from 0 to 1}
\end{aligned}
$$

If instead $h = k$,

$$
\begin{aligned}
\int_0^1 \sin(h \cdot 2\pi t)^2 dt &= \int_0^1 \frac{1}{2}(1 - \cos(2h \cdot 2\pi t))dt && \text{trigononmic identity (substitution)} \\
&= \frac{x}{2} - \frac{1}{(2h \cdot 2\pi)}\sin(2h \cdot 2\pi t)\Big|_{t=0}^{1} && \text{integral of cosine is sine} \\
&= 1/2 && \text{evaluate from 0 to 1}
\end{aligned}
$$

Now we wish to find the coefficients of $f(t)$. If we multiply $f(t)$ by $\sin(j \cdot 2\pi t)$ and integrate, we get:

$$
\begin{aligned}
&\int_0^1 f(t)\sin(j \cdot 2\pi t)dt \\
&= \int_0^1 \left(a_1 \sin(1 \cdot 2\pi t)\sin(j \cdot 2\pi t) + \cdots + a_j \sin(j \cdot 2\pi t)\sin(j \cdot 2\pi t) + \cdots + a_n \sin(n \cdot 2\pi t)\sin(j \cdot 2\pi t)\right) dt \\
&\qquad\qquad \text{(substitute for } f \text{ and distribute)} \\
&= \int_0^1 a_1 \sin(1 \cdot 2\pi t)\sin(j \cdot 2\pi t)dt + \cdots + \int_0^1 a_j \sin(j \cdot 2\pi t)\sin(j \cdot 2\pi t)dt + \cdots + \int_0^1 a_n \sin(n \cdot 2\pi t)\sin(j \cdot 2\pi t)dt \\
&\qquad\qquad \text{(the integral of a sum is the sum of integrals)} \\
&= 0 + \cdots + a_j/2 + \cdots + 0 \\
&\qquad\qquad \text{(all integrals are zero except for the } j^{\text{th}} \text{ integral by the above equations)} \\
&= a_j/2
\end{aligned}
$$

Then equating the first and last lines, we have a formula for recovering a_j:

$$a_j = 2\int_0^1 f(t)\sin(j \cdot 2\pi t)dt \tag{3}$$

This derivation is valid for as large of n as we want. In fact, the most general application of this is infinite series of sine waves.
Additionally, we can do the same thing with sums of cosines. If $f = b_1\cos(1 \cdot 2\pi t) + b_2\cos(2 \cdot 2\pi t) + \ldots$, then we can recover the coefficients in the same way (Exercise 3.1):

$$b_j = 2\int_0^1 f(t)\cos(j \cdot 2\pi t)dt \tag{4}$$

And if we have a function that is a mix of the two, such as

$$f = \sin(2\pi t) + \sin(2 \cdot 2\pi t) + \cdots + \cos(2\pi t) + \cos(2 \cdot 2\pi t) + \ldots$$

we can extract the coefficients by the same equations as above (Exercise 3.2).

3.2 Numerical Implementation

Now we see why the trapezoid rule was so important. In order to recover the sine coefficients of a function, we need to be able to find the integral of f multiplied against different sine functions. We can do this pretty well with our trapezoid code. To show this method in action, we show the code `myfreq.m` (full code in Appendix). We'll walk through it here. First, we create a sum of sine waves:

```
T = 5; % duration of signal
dt = 0.001;     % time between signal samples
t = 0:dt:T;
N = length(t);
y = 2.5*sin(3*2*pi*t) - 4.2*sin(4*2*pi*t);    % a 2-piece wave
plot(t,y)
xlabel('time   (seconds)')
ylabel('signal')
```

Next, we use equation (3) to find the coefficients:

```
for j = 1:5,     % compute the amplitudes as ratios of areas
    a(j) = mytrapz(t,y.*sin(j*2*pi*t))/mytrapz(t,sin(j*2*pi*t).^2);
end
```

Let's break down the syntax of this operation. We loop through the computation five times, each time representing a frequency `j` from 1 to 5. Within the loop, the first `mytrapz` is the integral we care about. We give it two arguments: `t` and `y.*sin(j*2*pi*t)`. The first, "`t`", is simply the time grid that we will approximately integrate over. The second corresponds to $f(t)\cos(j2\pi t)$, as in equation (3). Note that `j` is the frequency we are testing. The operator (`.*`) performs *elementwise* multiplication, that is, multiplying the corresponding elements in each array. (Many operators such as `+` and `sin()` automatically do this.) Thus each entry in the `y.*sin(j*2*pi*t)` corresponds to one in `t`.
The second instance of `mytrapz()` is just normalization (accounts for the length of the wave). After this, we plot the recovered coefficients against the originals. This effectively checks the error in our process.

```
figure
plot(1:5,a,'ko')    % plot the amplitudes vs frequency
hold on
plot(1:5, [0 0 2.5 -4.2 0], 'b*')
```

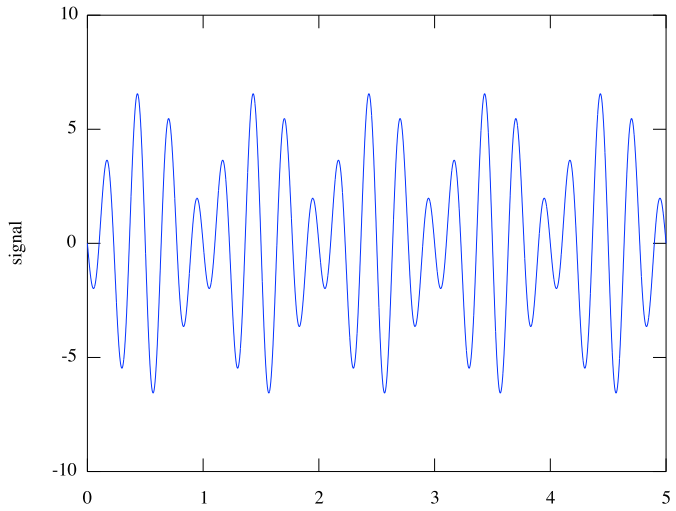

Figure 7: The function in question

As we can see from Figure 8, our method works well.

The remainder of the code uses the Matlab function `fft()`. This stands for "Fast Fourier Transform." This transforms maps a function from the *time* domain (the way we normally think about it) to the *frequency* domain. Basically, we express a signal in terms of the frequency coefficients of which it is composed. The plots from this analysis confirm that our analysis is the same as that of the Matlab-tested functions.

3.3 Estimating other functions

You may be pleasantly surprised to hear that *any* periodic function can be broken down into an infinite sum of sine and cosine waves. The elements of such a sum are together called a Fourier Series. The method for recovering the coefficients of the Fourier Series for any periodic function are the same as in equation (3). Thus we create a function that will find the decomposition of any function. We do this in `myfourier.m`. The entire code is in the appendix, but it uses very similar ideas as in `myfreq()` (above).

```
function [freq mag] = myfourier(y, sr, use_fft, plot_on)
(...snip...)
freq = (0:N-1)/T;        %fft frequencies
for n = 1 : numel(freq)
    %obtain coefficient for freqency 'freq(n)'
    sinmag(n) = mytrapz(t, y.*sin(freq(n)*2*pi*t), 1);
```

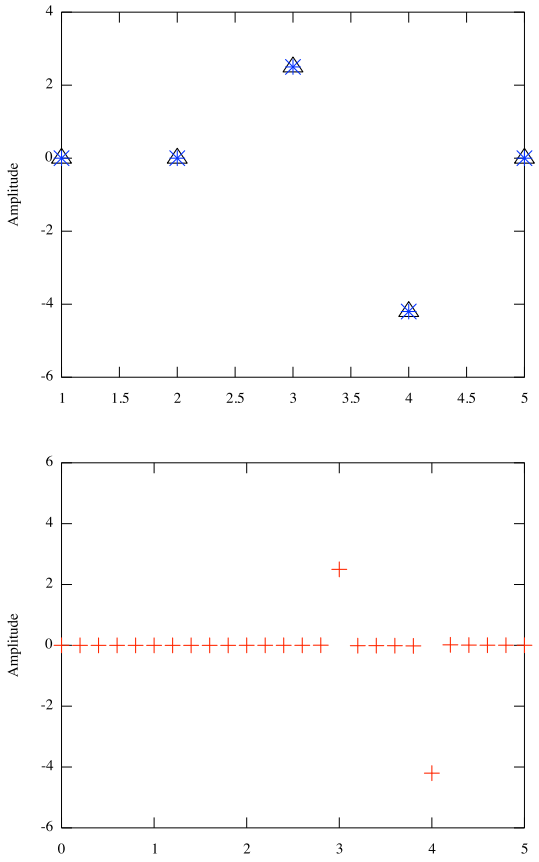

Figure 8: Left: Frequencies captured by `myfreq()`. Right: Frequencies captured by Matlab's `fft()`

```
    cosmag(n) = mytrapz(t, y.*cos(freq(n)*2*pi*t), 1);
end
```

As a test, we compare the "answer" `fft` code with our own for an arbitrary function and find good agreement:

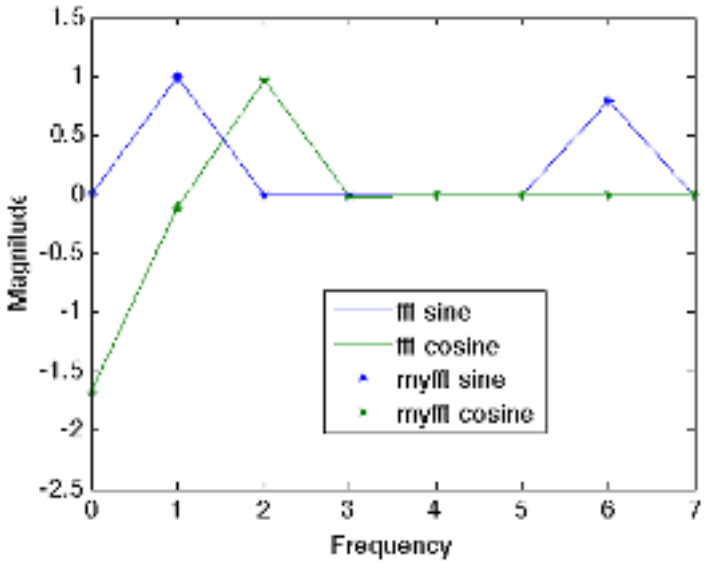

Figure 9: Comparison of `myfft` to the true frequencies

Exercises

3.1 Suppose $f = b_1 \cos(1 * 2\pi t) + b_2 \cos(2 * 2\pi t) + \ldots$. Then prove Equation 4 that helps recover the coefficients:

$$b_j = 2 \int_0^1 f(t) \cos(j \cdot 2\pi t) dt$$

3.2 Suppose f is an infinite sum of sine and cosine waves:

$$f = \sin(2\pi t) + \sin(2 \cdot 2\pi t) + \cdots + \cos(2\pi t) + \cos(2 \cdot 2\pi t) + \ldots$$

Show that Equations (3) and (4), which give expressions for a_j and b_j, are still valid.

4 Applications to EEG

Here is the application we have been seeking the whole time. Given an EEG signal, we would like to find its corresponding Fourier series. We have just built up the tools to decompose the signal into component frequencies. This is useful, but the brain changes over time, and we want to be able to track changes in the signal over time. One way to do this is the spectrogram, or the short-term Fourier transform. It allows us to watch the way a signal changes over time.

4.1 Short-time Fourier Transform and Spectrogram

The idea behind the Short-time Fourier Transform is that we break up the signal into several time windows, then take the Fourier transform of each one of these. In Figure (10), we analyze the function $f(x) = \cos(4 \cdot 2\pi t) + 3\cos(10 \cdot 2\pi t) + 5t^2 + 5\sin(35 \cdot 2\pi t)$ from time $t = 0$ to $t = 1$. First, we break it up into 10 segments of 100 ms each. Each segment corresponds to one block in the "Time" dimension. The line in that block is the magnitudes of sine and cosine waves from the Fourier transform of the function over that time window. The colors on the plane below it represent the magnitude of that frequency in that time interval. The higher the magnitude, the closer to red, and the lower, the closer to blue. The resulting image that we get is called a *spectrogram*.

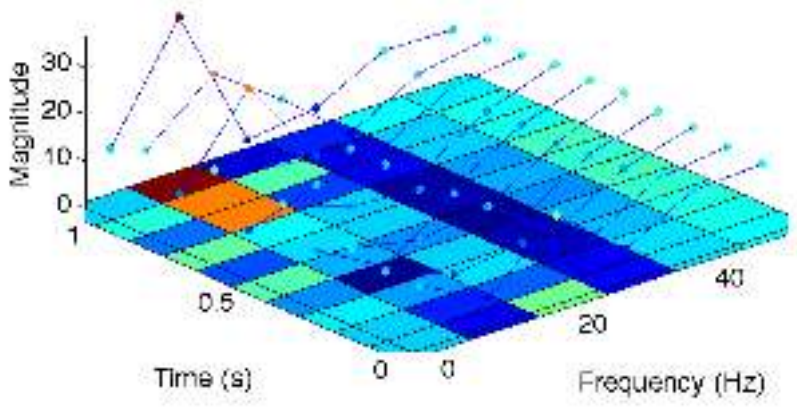

Figure 10: Visual illustration of a spectrogram

To code something like this, we consider each time window separately. This takes the form of a loop:

```
function [stft_plot freq tm] = my_stft(y, sr, win_len)
```

```
(...snip...)
for n = 1:Nwindow
    %isolate the part of the signal we want to deal with
    sig_win = y((n-1)*win_len + 1 : n*win_len);
```

Within this window, we perform the Fourier Transform (using our homemade code!) and record it:

```
    %perform the fourier transform
    [mg freq] = myfourier(sig_win, sr, 1);
    sm(:,n) = mg(:,1);
    cm(:,n) = mg(:,2);
end
```

We then plot the results on a 2-D plane using `imagesc()`. This function associates colors with the values in the matrix, so that you can see where the values in the matrix are larger. Thus we get a pretty picture as above.

4.1.1 Testing the Spectrogram

For the first test, we will try a simple sin wave.

For the second test, we will use a more interesting function. Below we have plotted $\sin(6 * 2\pi t) + \sin(20 * 2\pi t) + 2 * \sin(e^{t/1.5})$ over $[0, 10]$.

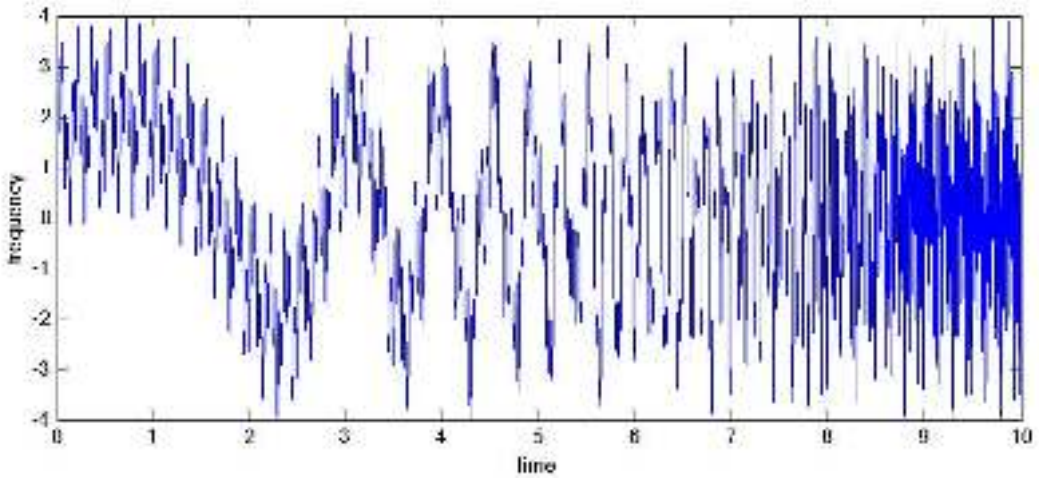

Figure 11: An interesting signal

This function is interesting because it contains a frequency component that is changing over time. While we have waves at a constant 6 and 20 Hertz, the third component speeds up as t gets larger and the exponential curve gets

steeper. Thus for the plot we expect to see a frequency component that is increasing. This is exactly what we see in Figure (12)–two constant band of frequency, and one train of frequency that increases with time.

```
>> sr = 1e-4;
>> t = 0:sr:10;
>> y = sin(6*2*pi*t)+sin(20*2*pi*t)+2*sin(exp(t/1.5));
>> my_stft(y, sr, 5000);
```

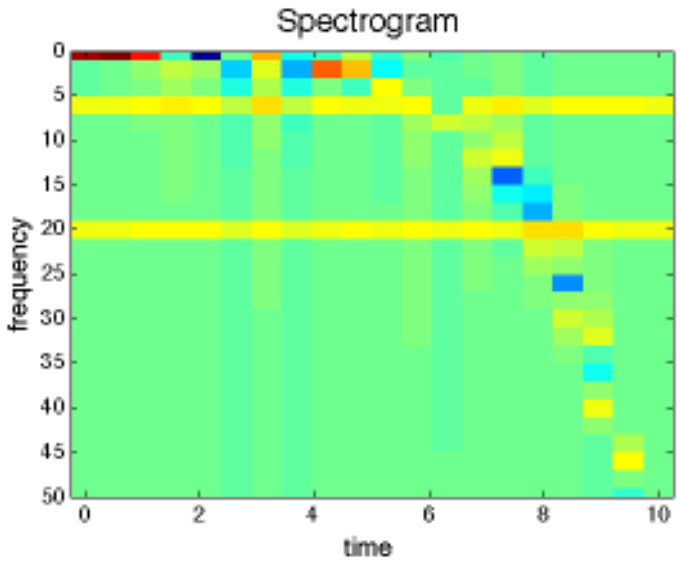

Figure 12: The spectrogram for the above function.

4.2 Application to EEG data

For the final section, we will analyze actual brain waves. We recorded from and EEG, and got the signal in Figure 13.

To analyze, we approximate the uneven signal as an even signal, finding the average sampling rate, in Hz. We then call myfourier

```
octave>  avg_interval = mean(data(2:end,1)-data(1:end-1,1));
octave>  sr = 1/(avg_interval/1000);
```

5 Conclusion

In this module we developed the tools to decompose an arbitrary signal, such as an EEG, into is component frequencies. We began with sine waves, established

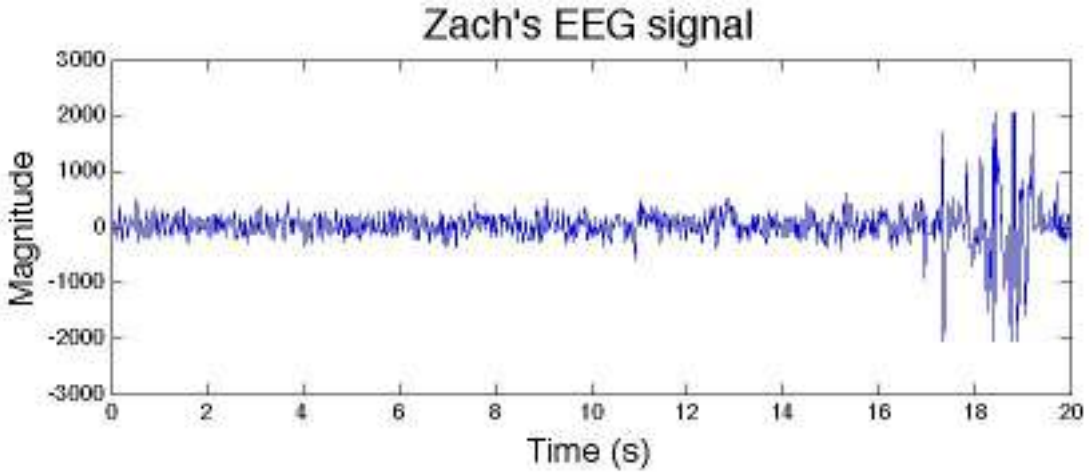

Figure 13: An EEG wave

the trapezoid scheme, and finally introduced Fourier analysis. This same flavor of analysis is used in many other settings, too–see the related documents.

6 Code

Code for `mytrapz.m`

```
function curve_area = mytrapz(x, y, fast)
% function curve_area = mytrapz(x, y, fast)
%
% mytrapz.m performs the trapezoid rule on the vector given by x and y.
%
% Input:
%   x - a vector containing the domain of the function
%   y - a vector containing values of the function corresponding to the
%   values in 'x'

if nargin < 3
    curve_area = 0;

    %loop through and add up trapezoids for as many points as we are given
    for n = 2 : numel(x)
        height = (y(n) + y(n-1))/2;     %average height of function across interval
        base = x(n) - x(n-1);           %length of interval
        trap_area = base * height;      %area of trapezoid
        curve_area = curve_area + trap_area;    %add to continuing sum
```

```
    end

elseif fast
    %alternate (fast) implementation
    curve_area = sum(y(2:end-1).*(x(3:end) - x(1:end-2)));
    curve_area = curve_area + y(1)*(x(2) - x(1)) + y(end)*(x(end) - x(end-1));
    curve_area = curve_area/2;
end
```

Code for `myfreq.m`

```
%
% myfreq.m
%
% find the frequencies and amplitudes at which a wave is "vibrating"
%
% Contrast simple (but laborious) trapezoid computations to the fast
% and flexible built-in fft command (fft stands for fast Fourier transform).
% To make full sense of this we will need to think about complex
% numbers and the complex exponential function.
%

T = 5; % duration of signal
dt = 0.001;     % time between signal samples
t = 0:dt:T;
N = length(t);
y = 2.5*sin(3*2*pi*t) - 4.2*sin(4*2*pi*t);    % a 2-piece wave
plot(t,y)
xlabel('time   (seconds)')
ylabel('signal')

for j = 1:5,     % compute the amplitudes as ratios of areas
    a(j) = trapz(t,y.*sin(j*2*pi*t))/trapz(t,sin(j*2*pi*t).^2);
end

figure
plot(1:5,a,'ko')    % plot the amplitudes vs frequency
hold on
plot(1:5, [0 0 2.5 -4.2 0], 'b*')


f = (0:N-1)/T; % fft frequencies
sc = N*trapz(t,sin(2*pi*t).^2)/T;   % fft scale factor
A = fft(y);
newa = -imag(A)/sc;
plot(f,newa,'r+')
```

```
y = y + 3*cos(6*2*pi*t);    % add a cosine piece
figure(1)
hold on
plot(t,y,'g')    % plot it
hold off
legend('2 sines','2 sines and 1 cosine')

figure(2)
A = fft(y);           % take the fft of the new signal
newa = -imag(A)/sc;
plot(f,newa,'gx')
b = real(A)/sc;
plot(f,b,'gx')
xlim([0 7])   % focus in on the low frequencies
hold off
xlabel('frequency  (Hz)')
ylabel('amplitude')
legend('by hand','by fft','with cosine')
```

Code for `myfourier.m`

```
% function [mag freq] = myfourier(y, dt, use_fft)
%
% myfourier.m decomposes the signal 'y', taken with sample interval dt,
% into its component frequencies.
%
% Input:
%
%   y       -- signal vection
%   dt      -- sample interval (s/sample) of y
%   use_fft -- if designated, use matlab's fft instead of trapezoid method
%
% Output:
%
%   freq -- frequency domain
%   mag  -- magnitude of frequency components of y corresponding to 'freq'

function [freq mag] = myfourier(y, dt, use_fft)

y = y(:);
N = numel(y); %number of samples
T = N*dt;                %total time
t = linspace(0,T,N)';     %reconstruct time vector
half_N = floor(N/2);  %ensures that N/2 is an integer
```

```
freq = (-half_N:half_N)'/T;   %fft frequencies

if nargin < 3  %perform explicit Fourier transform
    sinmag = zeros(size(freq));  %vector for component magnitudes
    cosmag = zeros(size(freq));  %vector for component magnitudes

    %loop through each frequency we will test
    for n = 1 : numel(freq)
        %obtain coefficient for freqency 'freq(n)'
        sinmag(n) = mytrapz(t, y.*sin(freq(n)*2*pi*t), 1);
        cosmag(n) = mytrapz(t, y.*cos(freq(n)*2*pi*t), 1);
    end

    %scale to account for sample length
    scale_factor = mytrapz(t, sin(2*pi*t).^2);
    sinmag = sinmag / scale_factor;
    cosmag = cosmag / scale_factor;
    mag = [sinmag(:)  cosmag(:)];

elseif use_fft   %use built-in MATLAB fft() for speed
    fft_scale_factor = mytrapz(t, sin(2*pi*t).^2) * N / T;
    A = fft(y);
    mag(:,1) = -imag(A)/fft_scale_factor;
    mag(:,2) = real(A)/fft_scale_factor;
    mag = circshift(mag, half_N);
end
```

Code for `mysgram.m`

```
%
% function [stft_plot freq tm] = my_stft(y, dt, Nwindow)
%
% my_stft splits the signa 'y' into time windows, the breaks each
% segment into its component frequencies.  See "Short-time Fourier Transform"
%
%
% Input:
%   y       -- signal
%   dt      -- sample interval
%   Nwindow -- number of time intervals to analyze
%
% Output:
%   stft_plot -- values plotted in the spectrogram
%   freq      -- frequency domain
%   tm        -- time domain
```

```
function [stft_plot freq tm hh] = mysgram(y, dt, Nwindow)

%count the number of windows
N = numel(y);
win_len = floor(N/Nwindow);
sm = zeros(win_len, Nwindow);
cm = zeros(win_len, Nwindow);
tm = linspace(0, numel(y) * dt, Nwindow);

%for each window
for n = 1:Nwindow
    %isolate the part of the signal we want to deal with
    sig_win = y((n-1)*win_len + 1 : n*win_len);
    %perform the fourier transform
    [freq mg] = myfourier(sig_win, dt, 1);
    sm(:,n) = mg(1:win_len,1);
    cm(:,n) = mg(1:win_len,2);
end

stft_plot = abs(sm + cm);
stft_plot = stft_plot(round(end/2):end, :);

%plot the fourier transform over time
hh = imagesc(tm, freq(round(end/2):end), stft_plot);
title('Spectrogram', 'FontSize', 20)
xlabel('time', 'FontSize', 16)
ylabel('frequency', 'FontSize', 16)
set(gca, 'ydir', 'normal')

%just look at lower frequencies
ylim([0-win_len/2 50+win_len/2])
```