Lecture 1
Basics of Fourier Series and the Fourier Transform

Jason J. Bramburger

Motivating Example: Let's say you have an audio signal - our example uses a clip of Handel's Messiah that is built-in to MATLAB. We can plot the audio signal as amplitude vs. time, which lets us think of it as a function of time. As we listen, we can hear some parts that are higher frequency and some parts that are lower frequency. When we listen for different frequencies, what we are doing in our heads is taking the single audio signal and breaking it down into different frequencies. Some are stronger, while some are more faint or not present at all. So, how do we precisely break a signal down into its frequencies?

```
load handel % a clip of Handel's "Messiah" is included with MATLAB

v = y'/2;
plot((1:length(v))/Fs,v);
xlabel('Time [sec]');
ylabel('Amplitude');

soundsc(v,Fs) % Play the audio
```

Question: If we are given a signal, how do we break it down into different frequencies?

Idea: If we are given a function, we would like to break it down into sines and cosines of different frequencies: $\sin(kt)$ and $\cos(kt)$.

Recall: I remind you of an important formula, referred to as *Euler's formula*:

$$\mathrm{e}^{\mathrm{i}\theta} = \cos(\theta) + \mathrm{i}\sin(\theta),$$

where $\mathrm{i} = \sqrt{-1}$ is the imaginary constant. Some of the formulas I will present will use sines and cosines and others will use complex exponentials. In either case, you can convert back and forth using Euler's formula. When you see complex exponentials, think oscillations like sine and cosine!

Fourier Transform

Suppose we are given a function $f(x)$ with $x \in \mathbb{R}$. We define the *Fourier transform* of $f(x)$, written $\hat{f}(k)$, by the formula

$$\hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x)\mathrm{e}^{-\mathrm{i}kx}\mathrm{d}x.$$

Furthermore, if you are given $\hat{f}(k)$ and want to recover $f(x)$, you can use the *inverse Fourier transform*:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(k)\mathrm{e}^{\mathrm{i}kx}\mathrm{d}k.$$

Knowing that $\mathrm{e}^{\mathrm{i}kx}$ is like $\sin(kx)$ and $\cos(kx)$, the value of k gives the frequencies of sine and cosine waves. Therefore, the Fourier transform takes a function of space (or time), x, and converts it to a function of frequencies, k.

The Fourier transform has a number of uses throughout math, but you will notice that it assumes an infinite domain and so is not necessarily what we want for the practical situations we are considering in this class. For example, our audio signal is a clip from a song and so doesn't go on forever. Let's now consider something that works for finite domains.

<u>Fourier Series</u>

Suppose we are given a function $f(x)$ for a limited range of x values. For simplicity, we will take $x \in [-\pi, \pi]$. The first thing to note is that all frequencies won't 'fit' into this interval. That is, we will only consider $\sin(kx)$ and $\cos(kx)$ with integer k since these are the only frequencies that lead to functions that completely repeat over the interval. This leaves infinitely many frequencies, given by $k = 1, 2, 3, 4, \dots$, and we write $f(x)$ as the infinite sum of these sines and cosines:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_n \cos(kx) + b_n \sin(kx)), \quad x \in [-\pi, \pi].$$

The constant term $a_0/2$ is for shifting the function up and down.

<u>Fourier Coefficients</u>

The above formula for the Fourier series looks great, but leaves us with a big question: how do we find the *Fourier coefficients* a_k and b_k? We can obtain them using the following formulas (left for you to check on your own):

$$\begin{aligned}
\int_{-\pi}^{\pi} \sin(nx)\cos(mx)\mathrm{d}x &= 0, \quad \forall n, m \\
\int_{-\pi}^{\pi} \cos(nx)\cos(mx)\mathrm{d}x &= \begin{cases} 0, & n \neq m \\ \pi, & n = m \end{cases} \\
\int_{-\pi}^{\pi} \sin(nx)\sin(mx)\mathrm{d}x &= \begin{cases} 0, & n \neq m \\ \pi, & n = m \end{cases}
\end{aligned}$$

Then, if we take the Fourier series and multiply both sides by $\cos(mx)$ for some positive integer m, we get

$$f(x)\cos(mx) = \frac{a_0}{2}\cos(mx) + \sum_{k=1}^{\infty} [a_k \cos(kx)\cos(mx) + b_k \sin(kx)\cos(mx)].$$

Integrating from $-\pi$ to π gives:

$$\begin{aligned}
\int_{-\pi}^{\pi} f(x)\cos(mx)\mathrm{d}x &= \int_{-\pi}^{\pi} \frac{a_0}{2}\cos(mx)\mathrm{d}x \\
&\qquad + \sum_{k=1}^{\infty} [a_k \int_{-\pi}^{\pi} \cos(kx)\cos(mx)\mathrm{d}x + b_k \int_{-\pi}^{\pi} \sin(kx)\cos(mx)\mathrm{d}x] \\
&= \pi a_m.
\end{aligned}$$

This holds for any m that you choose, so it gives a formula for every Fourier coefficient a_k. Multiplying instead by $\sin(mx)$ and integrating from $-\pi$ to π will similarly get you the b_k terms. In sum, we have the following formulas:

$$\begin{aligned}
a_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)\mathrm{d}x, \\
b_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)\mathrm{d}x, \\
a_0 &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\mathrm{d}x.
\end{aligned}$$

The formula for a_0 gives that it represents the average of $f(x)$ over the interval $[-\pi, \pi]$.

<u>Remarks:</u>

1. Since sine and cosine are periodic functions, the function $f(x)$ that can be expressed as a Fourier series must be a periodic function.

2. Amazingly, the representation of a function as a sum of sines and cosines also works for discontinuous functions. One thing we have been sweeping under the rug is what we mean by saying $f(x)$ is equal to the infinite series. When we have an infinite series, the sequence of partial sums must converge to something. In this case, the partial sums are functions. There are several different mathematical notions on what it means for a sequence of functions to converge. It is beyond this class to get into those kinds of details, but we will say that for our purposes, the series converges to the value $f(x)$ for each x where f is continuous and to the average of the left and right values if $f(x)$ is discontinuous.

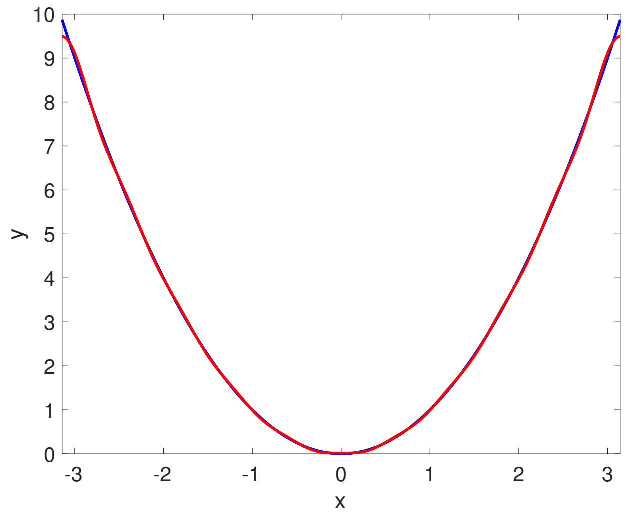
3. Even functions must have $b_k = 0$ for every k. Similarly, even functions have $a_0 = 0$ and $a_k = 0$ for every $k \geq 1$.

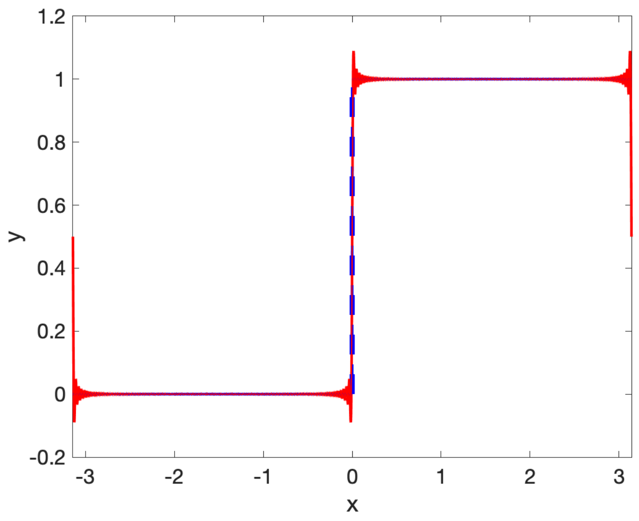
4. We presented the Fourier series for a function on the interval $[-\pi, \pi]$. For a function on an interval $[-L, L]$ we have:
$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_n \cos\left(\frac{\pi k x}{L}\right) + b_n \sin\left(\frac{\pi k x}{L}\right) \right), \quad x \in [-L, L].$$
The coefficients then take the form:
$$a_k = \frac{1}{L} \int_{-L}^{L} f(x) \cos\left(\frac{\pi k x}{L}\right) \mathrm{d}x,$$
$$b_k = \frac{1}{L} \int_{-L}^{L} f(x) \sin\left(\frac{\pi k x}{L}\right) \mathrm{d}x,$$
$$a_0 = \frac{1}{2L} \int_{-L}^{L} f(x) \mathrm{d}x.$$

5. The Fourier series is also sometimes written in terms of complex exponentials. The formula for a function $f(x)$ with $x \in [-L, L]$ is
$$f(x) = \sum_{k=-\infty}^{\infty} c_k \mathrm{e}^{\mathrm{i}k\pi x/L}$$
where
$$c_k = \frac{1}{2L} \int_{-L}^{L} f(x) \mathrm{e}^{-\mathrm{i}k\pi x/L} \mathrm{d}x, \quad k \in \mathbb{Z}$$
<u>WARNING</u>: the c_k above are now complex numbers. If the function $f(x)$ is a real-valued function, then c_0 is real and $c_{-k} = \bar{c}_k$ for all $k \neq 0$. If $f(x)$ is even, all c_k are real. If $f(x)$ is odd, $c_0 = 0$ and all c_k are purely imaginary.

```
% Continuous function: f(x) = x^2
x = linspace(-pi,pi,10001);
y = x.^2;

% Plot x^2 function
figure(1)
plot(x,y,'b','Linewidth',4)
set(gca, 'Fontsize', 16)
xlabel('x')
ylabel('y')

% Plot Nth partial sum of Fourier series
N = 10;
yN = pi^2/3*ones(size(x)); %0th term
for k = 1:N %1 to Nth terms
    yN = yN + (-1)^k*4/(k^2)*cos(k*x);
end

hold on
```

```
plot(x,yN,'r','Linewidth',2)
xlim([-pi,pi])

% Discontinuous function: unit step function
y = (x > 0);

figure(2)
plot(x(1:5001),y(1:5001),'b','Linewidth',2)
hold on
plot([0 0],[0 1],'--b','Linewidth',4)
plot(x(5002:10000),y(5002:10000),'b','Linewidth',2)
axis([-pi pi -.2 1.2])

% Plot Nth partial sum of Fourier series
N = 100;
yN = 0.5*ones(size(x));
for k = 1:N
   yN = yN + 2/((2*k-1)*pi)*sin((2*k-1)*x);
end

plot(x,yN,'r','Linewidth',1)
set(gca, 'Fontsize', 16)
xlabel('x')
ylabel('y')
```





More Problems: We saw above that the Fourier transform won't be useful for our audio clip example since it requires an infinite domain. You should be convinced that the Fourier series is more appropriate, but we still have an issue. The Fourier series requires me to have an actual function $f(x)$ to turn into a series. If you're given a an audio clip, you certainly don't have a function, you just have the amplitude of the signal at discrete points in time: a vector. Therefore, for data applications, we need to be able to compute the Fourier series even when you only know the function at some discrete set of points.

Discrete Fourier Transform (DFT)

One thing to keep in mind about being given a function at a discrete set of points is that we can never really know about extremely high frequency phenomena that is happening between these points. Therefore, we can't expect information about a signal for large k. The DFT is just like a Fourier series but truncated at some maximum frequency to reflect this potential shortcoming.

Suppose we are given a sequence (or vector) of N values that are a function sampled at equally-spaced points $\{x_0, x_1, x_2, \dots, x_{N-1}\}$. The discrete Fourier transform is a sequence of numbers given by

$$\hat{x}_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n \mathrm{e}^{\frac{2\pi \mathrm{i} kn}{N}}.$$

Here we have only finitely-many frequencies present: the ones given by $k = 0, 1, 2, \dots, N-1$.

There is something tricky here that is important when we get to the MATLAB implementation. Since we are only sampling a signal at discrete values, it is possible that we may mistake it for another, different signal. This is referred to as **aliasing** and comes from the fact that $-k$ and $k+N$ are the same in the DFT.

Because of aliasing, you could instead consider the frequencies which come from

$$k = -N/2, -N/2+1, \dots, -1, 0, 1, 2, \dots, N/2-1.$$

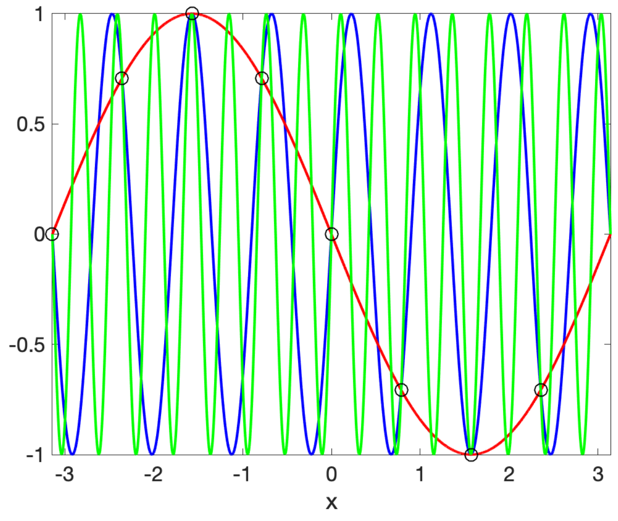
It is arbitrary whether you choose $N/2$ or $-N/2$, but this is how MATLAB does it. To make things more confusing, MATLAB gives the DFT values in this order:

$$\{\hat{x}_0, \hat{x}_1, \dots, \hat{x}_{N/2-1}, \hat{x}_{-N/2}, \hat{x}_{-N/2+1}, \dots, \hat{x}_{-1}\}.$$

```matlab
N = 8;
x2 = linspace(-pi,pi,N+1); % domain discretization
x = x2(1:N); % consider only the first N points (periodicity)
xplot = linspace(-pi,pi,10001); %domain for plotting

% sin((N-1)*x) function plot
y1 = sin((N-1)*x);
y1plot = sin((N-1)*xplot);
plot(xplot,y1plot,'b','Linewidth',2)
xlabel('x')
set(gca, 'Fontsize', 16)

% Add in sin(-x)
hold on
y2plot = sin(-xplot);
plot(xplot,y2plot,'r','Linewidth',2)

% Add in sin((2*N-1)*x)
y3plot = sin((2*N-1)*xplot);
plot(xplot,y3plot,'g','Linewidth',2)
xlim([-pi,pi])

% Add in discrete points
plot(x,y1,'ok','Markersize',10)
```

Fast Fourier Transform (FFT)

The FFT is an algorithm that does the DFT faster. For the DFT using the formula above, calculating each number

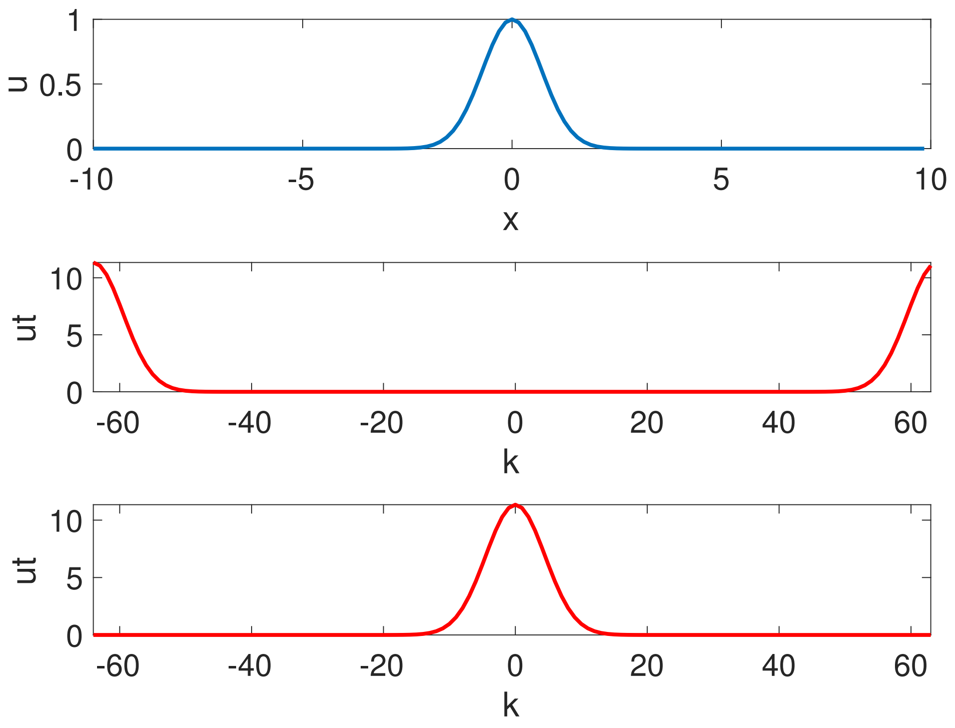


takes N operations. Since you need to compute N numbers, the total complexity is $\mathcal{O}(N^2)$, which is most likely very large since you want to have lots of data (N is big). The FFT is $\mathcal{O}(N\log(N))$. This doesn't look like a big deal, but applications that use the FFT, like signal processing, have very large N. For example, consider a signal with a million data points: $N = 10^6$. Then, computing the DFT by hand takes $N^2 = 10^{12}$ operations - that's a trillion operations! On the other hand, $\log(10^6) = 6\log(10) = 6$, so the number of operations for the DFT is on the order of 10^6.

We won't go into detail about the FFT algorithm, but the basic idea is that if you have a sequence of length N, you can split the DFT into two DFTs of sequences of length $N/2$. Then, you can repeat this process over and over. This is referred to as a *divide and conquer algorithm*. Typically, people will use powers of 2 for N (i.e. $N = 2^n$). Furthermore, MATLAB has a built-in function to perform the FFT.

```
L = 20;
N = 128; % N = 2^7

x2 = linspace(-L/2,L/2,N+1); % domain discretization
x = x2(1:N); % consider only the first N points (periodicity)

u = exp(-x.^2); % Gaussian function

subplot(3,1,1)
plot(x,u,'Linewidth',2)
set(gca, 'Fontsize', 16)
xlabel('x')
ylabel('u')

% Plot fft of u
ut = fft(u); % fft of u
k = -N/2:N/2-1; %frequency domain
subplot(3,1,2)
plot(k,abs(ut),'r','Linewidth',2)
set(gca, 'Fontsize', 16)
xlabel('k')
ylabel('ut')
xlim([-64,63])
% the frequencies and ut don't match up because ut has frequencies in the
% order k = 0,1,...,N/2-1,-N/2,...,-1

% Plot shifted fft of u
utshift = fftshift(ut); % shift Fourier transform ut
subplot(3,1,3)
plot(k,abs(utshift),'r','Linewidth',2)
set(gca, 'Fontsize', 16)
```

```
xlabel('k')
ylabel('ut')
xlim([-64,63])
```



The Fourier Transform for Derivatives

One practical application for the Fourier transform is that it makes it easy to take derivatives. Consider a function $f(x)$ and its Fourier transform $\hat{f}(k)$. Then,

$$\begin{aligned}\hat{f}'(k) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f'(x)\mathrm{e}^{-\mathrm{i}kx}\mathrm{d}x \\ &= \frac{1}{\sqrt{2\pi}}\left[f(x)\mathrm{e}^{-\mathrm{i}kx}\Big|_{-\infty}^{\infty} + \mathrm{i}k\int_{-\infty}^{\infty} f(x)\mathrm{e}^{-\mathrm{i}kx}\mathrm{d}x\right],\end{aligned}$$

using integration by parts. When we introduced the Fourier transform earlier we failed to mention that we require functions that decay at infinity. That is, $f(x) \to 0$ as $x \to \pm\infty$, and so

$$\hat{f}'(k) = \mathrm{i}k\underbrace{\int_{-\infty}^{\infty} f(x)\mathrm{e}^{-\mathrm{i}kx}\mathrm{d}x}_{=\hat{f}(k)}.$$

This means that in frequency space (k domain), taking a derivative just means multiplying by $\mathrm{i}k$: $\hat{f}' = \mathrm{i}k\hat{f}$.

Application: One way that this is used is in solving differential equations. Consider the equation

$$y'' - \omega^2 y = -f(x), \quad x \in (-\infty, \infty),$$

where $\omega > 0$ is an arbitrary constant. Taking the Fourier transform of both sides of the differential equation gives:

$$\begin{aligned} \hat{y''} - \omega^2\hat{y} &= -\hat{f} \\ \implies -k^2\hat{y} - \omega^2\hat{y} &= -\hat{f} \\ \implies (k^2+\omega^2)\hat{y} &= \hat{f} \\ \implies \hat{y} &= \frac{\hat{f}}{k^2+\omega^2}.\end{aligned}$$

Hence, we have turned a differential equation into an algebraic equation. If you want the solution to the differential equation, you can use the inverse Fourier transform:

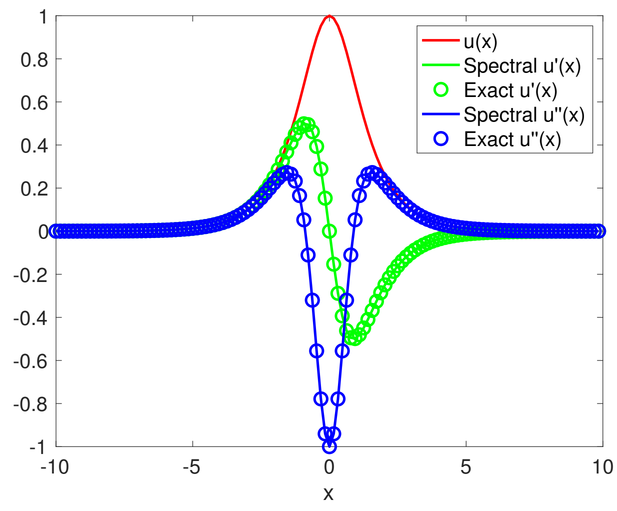
$$y(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \frac{\hat{f}(k)}{k^2 + \omega^2} \mathrm{e}^{\mathrm{i}kx} \mathrm{d}k.$$

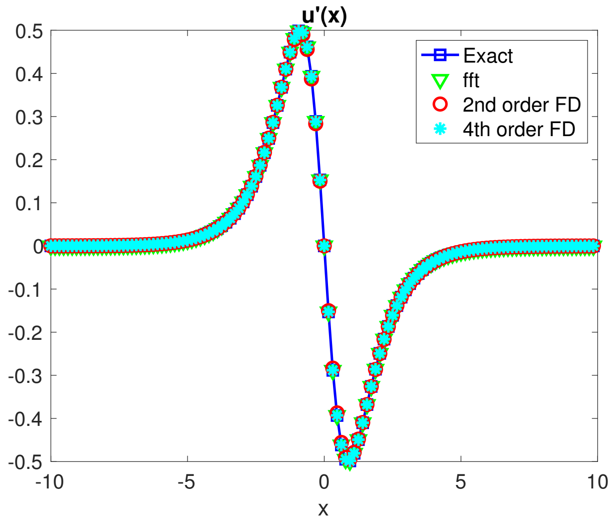
In some cases you can solve this integral exactly, otherwise you can find $y(x)$ numerically.

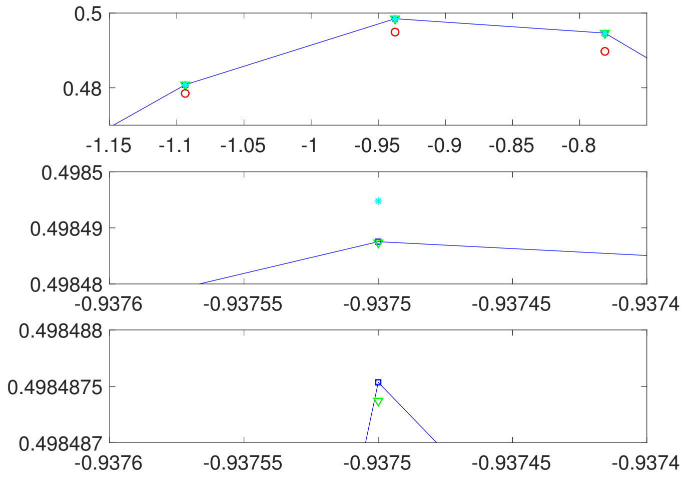
Numerical Derivatives: We can also use the Fourier transform to numerically calculate derivatives. To do anything numerically, we need things to be finite so will be using the FFT, but we will still use the fact that derivatives are the same as multiplication by $\mathrm{i}k$. The more traditional way of numerically calculating derivatives is via finite differences, which come from Taylor series expansions. Finite difference formulas are local - they only take into account points close to where you take the derivative. Using the Fourier transform is global - it uses information about the function at all points.

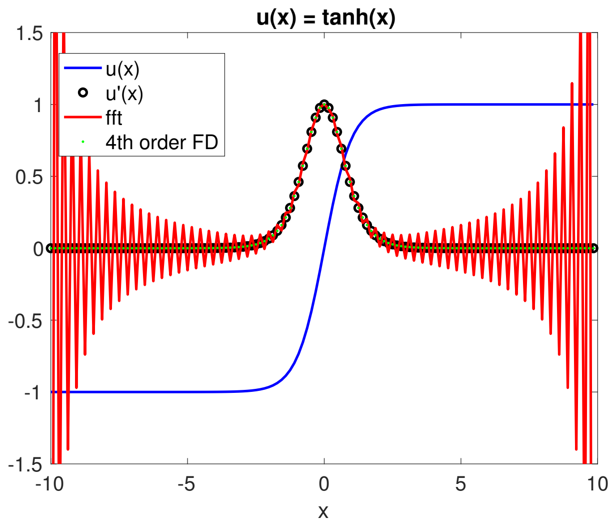
```
L = 20; % length of the computational domain: [-L/2,L/2]
n = 128; % number of Fourier modes: n = 2^7

x2 = linspace(-L/2,L/2,n+1); %domain discretization
x = x2(1:n); % only the first n points (periodicity)
dx = x(2) - x(1); % finite difference step size
u = sech(x); % function to take the derivative of
ut = fft(u); % fft of the function
k = (2*pi/L)*[0:(n/2 - 1) (-n/2):-1]; % rescaled frequencies

% fft derivatives
dutdk = 1i*k.*ut; % first derivative
d2utdk2 = -k.^2.*ut; % second derivative
dudx = real(ifft(dutdk)); %inverse transform - round off imaginary part due to error
d2udx2 = real(ifft(d2utdk2));

dudx_exact = -sech(x).*tanh(x); % analytic first derivative
d2udx2_exact = sech(x)-2*sech(x).^3; % analytic second derivative

% Finite difference second derivative

% Second order accuracy
ufd2(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);
for j=2:n-1
    ufd2(j)=(u(j+1)-u(j-1))/(2*dx);
end
ufd2(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);

% Fourth order accuracy
ufd4(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);
ufd4(2)=(-3*u(2)+4*u(3)-u(4))/(2*dx);
for j=3:n-2
    ufd4(j)=(-u(j+2)+8*u(j+1)-8*u(j-1)+u(j-2))/(12*dx);
end
ufd4(n-1)=(3*u(n-1)-4*u(n-2)+u(n-3))/(2*dx);
ufd4(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);

% Spectral derivatives
figure(1)
plot(x,u,'r',x,dudx,'g',x,dudx_exact,'go',x,d2udx2,'b',x,d2udx2_exact,'bo','Linewidth',2,'Markersize',10
legend('u(x)',"Spectral u'(x)","Exact u'(x)","Spectral u''(x)","Exact u''(x)",'Fontsize',16)
xlabel('x')
set(gca,'Fontsize',16)

% Compare first derivatives
figure(2)
plot(x,dudx_exact,'bs-',x,dudx,'gv',x,ufd2,'ro',x,ufd4,'c*','Linewidth',2,'Markersize',10)
```

```
legend('Exact','fft','2nd order FD','4th order FD','Fontsize',16)
xlabel('x')
title("u'(x)",'Fontsize',16)
set(gca,'Fontsize',16)

% Zoom in on first derivatives
figure(3)
subplot(3,1,1), plot(x,dudx_exact,'bs-',x,dudx,'gv',x,ufd2,'ro',x,ufd4,'c*')
set(gca,'Fontsize',12)
axis([-1.15 -0.75 0.47 0.5])
set(gca,'Fontsize',16)
subplot(3,1,2); plot(x,dudx_exact,'bs-',x,dudx,'gv',x,ufd2,'ro',x,ufd4,'c*')
axis([-0.9376 -0.9374 0.49848 0.49850])
set(gca,'Fontsize',16)
subplot(3,1,3), plot(x,dudx_exact,'bs-',x,dudx,'gv',x,ufd2,'ro',x,ufd4,'c*')
axis([-0.9376 -0.9374 0.498487 0.498488])
set(gca,'Fontsize',16)

% -----------------------------------------------------------------------
% Function with a jump discontinuity at the boundaries
u = tanh(x); % function to take a derivative of
ut = fft(u); % fft of the function

% FFT calculation of derivatives
dutdk = 1i*k.*ut; % first derivative
dudx = real(ifft(dutdk)); %inverse transform - round off imaginary part due to error
dudx_exact = sech(x).^2; % analytic first derivative

% 4th-order accurate finite difference

ufd2(1)=(-3*u(1)+4*u(2)-u(3))/(2*dx);
ufd2(2)=(-3*u(2)+4*u(3)-u(4))/(2*dx);
for j=3:n-2
    ufd2(j)=(-u(j+2)+8*u(j+1)-8*u(j-1)+u(j-2))/(12*dx);
end
ufd2(n-1)=(3*u(n-1)-4*u(n-2)+u(n-3))/(2*dx);
ufd2(n)=(3*u(n)-4*u(n-1)+u(n-2))/(2*dx);

figure(4)
plot(x,u,'b',x,dudx_exact,'ko',x,dudx,'r',x,ufd2,'g.','Linewidth',2,'Markersize',6)
axis([-10 10 -1.5 1.5])
legend('u(x)',"u'(x)",'fft','4th order FD','Fontsize',16,'Location','Best')
xlabel('x')
title("u(x) = tanh(x)",'Fontsize',16)
set(gca,'Fontsize',16)
```









Higher Dimensional Fourier Transforms

You can also take the Fourier transform for functions in 2D, 3D, etc. It decomposes the signal into frequencies in each direction. You can take them in MATLAB with the commands fft2 for the 2D FFT and fftn for the n-dimensional FFT.