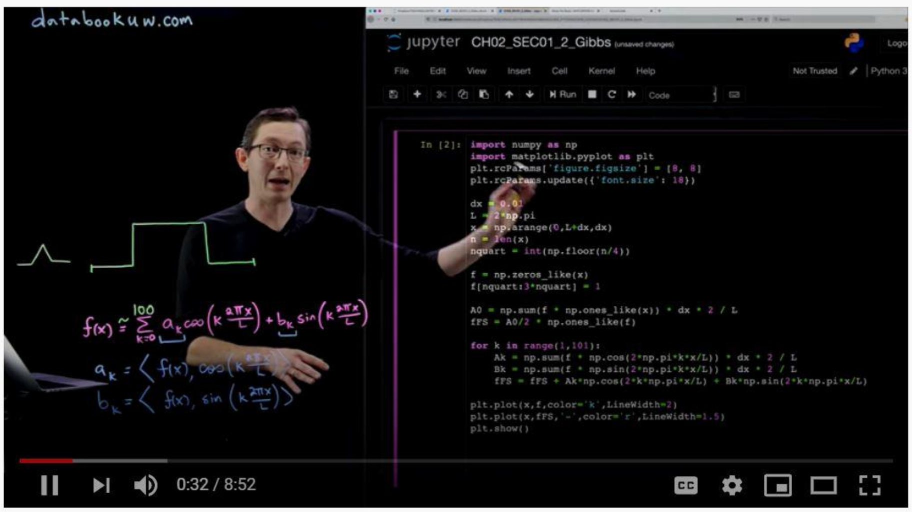
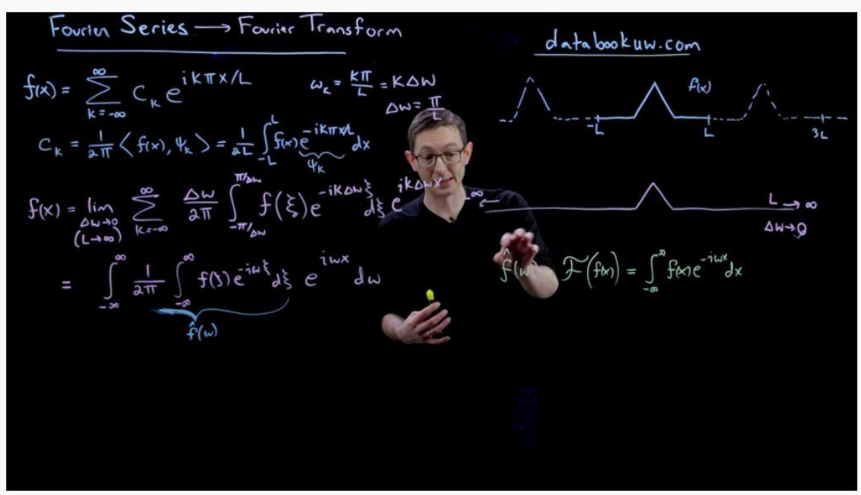
DAILY ASSESSMENT FORMAT

Date:	**27/05/2020**	**Name:**	**Sachin Krishna Moger**
Course:	**Digital Signal Processing**	**USN:**	**4AL17EC103**
Topic:	**Fourier Transforms**	**Semester & Section:**	**6-B**
Github Repository:	**Sachin-Courses**		

FORENOON SESSION DETAILS



Fourier Series and Gibbs Phenomena [Python]



Fourier series AfundamentalresultinFourieranalysisisthatiff(x)isperiodicandpiecewise smooth, then it can be written in terms of a Fourier series, which is an infinite sum of cosines and sines of increasing frequency. In particular, if f(x) is 2πperiodic, it may be written as:

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

FOURIER SERIES AND FOURIER TRANSFORMS

The coefficients ak and bk are given by

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

which may be viewed as the coordinates obtained by projecting the function ontotheorthogonalcosineandsinebasis{cos(kx),sin(kx)}∞ k=0. Inotherwords, the integrals in (2.6) may be re-written in terms of the inner product as:

$$\begin{aligned} a_k &= \frac{1}{\| \cos(kx) \|^2} \langle f(x), \cos(kx) \rangle \\ b_k &= \frac{1}{\| \sin(kx) \|^2} \langle f(x), \sin(kx) \rangle, \end{aligned}$$

where kcos(kx)k2 = ksin(kx)k2 = π. This factor of 1/π is easy to verify by numerically integrating cos(x)2 and sin(x)2 from−π to π. The Fourier series for an L-periodic function on [0,L) is similarly given by:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos\left(\frac{2\pi k x}{L}\right) + b_k \sin\left(\frac{2\pi k x}{L}\right) \right),$$

with coefficients ak and bk given by

$$\begin{aligned} a_k &= \frac{2}{L} \int_0^L f(x) \cos\left(\frac{2\pi k x}{L}\right) dx \\ b_k &= \frac{2}{L} \int_0^L f(x) \sin\left(\frac{2\pi k x}{L}\right) dx. \end{aligned}$$

The DFT is tremendously useful for numerical approximation and computation, but it does not scale well to very large n 1, as the simple formulation involves multiplication by a dense n × n matrix, requiring O(n2) operations. In 1965, James W. Cooley (IBM) and

John W. Tukey (Princeton) developed the revolutionary fast Fourier transform (FFT) algorithm [137, 136] that scales as O(nlog(n)). As n becomes very large, the log(n) component grows slowly, and the algorithm approaches a linear scaling. Their algorithm was based on afractal symmetry in the Fourier transform that allows an n dimensional DFT to be solved with a number of smaller dimensional DFT computations. Although thedifferentcomputationalscalingbetweentheDFTandFFTimplementations may seem like a small difference, the fast O(nlog(n)) scaling is what enables the ubiquitous use of the FFT in real-time communication, based on audio and image compression

Discrete Fourier transform
AlthoughwewillalwaysusetheFFTforcomputations,itisillustrativetobegin with the simplest formulation of the DFT. The discrete Fourier transform is given by:

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j e^{-i2\pi jk/n},$$

and the inverse discrete Fourier transform (iDFT) is given by:

$$f_k = \frac{1}{n} \sum_{j=0}^{n-1} \hat{f}_j e^{i2\pi jk/n}.$$

Thus, the DFT is a linear operator (i.e., a matrix) that maps the data points in f to the frequency domainˆ f:

$$\{f_1, f_2, \cdots, f_n\} \quad \xrightarrow{\text{DFT}} \quad \{\hat{f}_1, \hat{f}_2, \cdots \hat{f}_n\}.$$

For a given number of points n, the DFT represents the data using sine and cosine functions with integer multiple so fundamental frequency,ωn = e−2πi/n. The DFT may be computed by matrix multiplication:

$$\begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \vdots \\ \hat{f}_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{bmatrix}.$$

Fast Fourier transform to compute derivatives:

```
n = 128;
L = 30;
dx = L/(n);
x = -L/2:dx:L/2-dx;
f = cos(x).*exp(-x.^2/25);
% Function df = -(sin(x).*exp(-x.^2/25) + (2/25)*x.*f);
% Derivative
%% Approximate derivative using finite Difference... for
kappa=1:length(df)-1
dfFD(kappa) = (f(kappa+1)-f(kappa))/dx;
end dfFD(end+1) = dfFD(end);
%% Derivative using FFT (spectral derivative)
fhat = fft(f); kappa = (2*pi/L)*[-n/2:n/2-1];
kappa = fftshift(kappa);
% Re-order fft frequencies dfhat = i*kappa.*fhat;
dfFFT = real(ifft(dfhat));
%% Plotting commands plot(x,df,'k','LineWidth',1.5),
hold on plot(x,dfFD,'b--','LineWidth',1.2)
plot(x,dfFFT,'r--','LineWidth',1.2)
legend('True Derivative','Finite Diff.','FFT Derivative')
```

Code to simulate the 1D heat equation using the Fourier transform.

```matlab
a = 1;          % Thermal diffusivity constant
L = 100;        % Length of domain
N = 1000;       % Number of discretization points
dx = L/N;
x = -L/2:dx:L/2-dx; % Define x domain

% Define discrete wavenumbers
kappa = (2*pi/L)*[-N/2:N/2-1];
kappa = fftshift(kappa);    % Re-order fft wavenumbers

% Initial condition
u0 = 0*x;
u0((L/2 - L/10)/dx:(L/2 + L/10)/dx) = 1;

% Simulate in Fourier frequency domain
t = 0:0.1:10;
[t,uhat]=ode45(@(t,uhat)rhsHeat(t,uhat,kappa,a),t,fft(u0));

for k = 1:length(t) % iFFT to return to spatial domain
    u(k,:) = ifft(uhat(k,:));
end
```