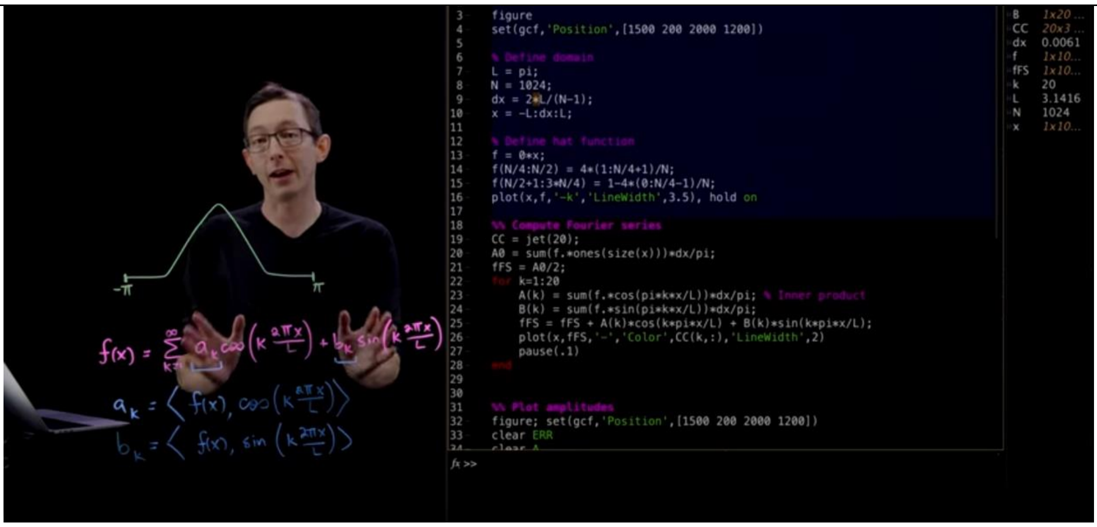
DAILY ASSESSMENT

Date:	**25-5-2020**	**Name:**	**Kavyashree m**
Course:	**Digitall signal processing**	**USN:**	**4al15ec036**
Topic:	**Fourier series-with complex series,using matlab,using python implementation,using ghibs phenomena .**	**Semester & Section:**	**8th A**
Github Repository:	**Kavya**		

FORENOON SESSION DETAILS

Image of session

Fig 1 : fourier series

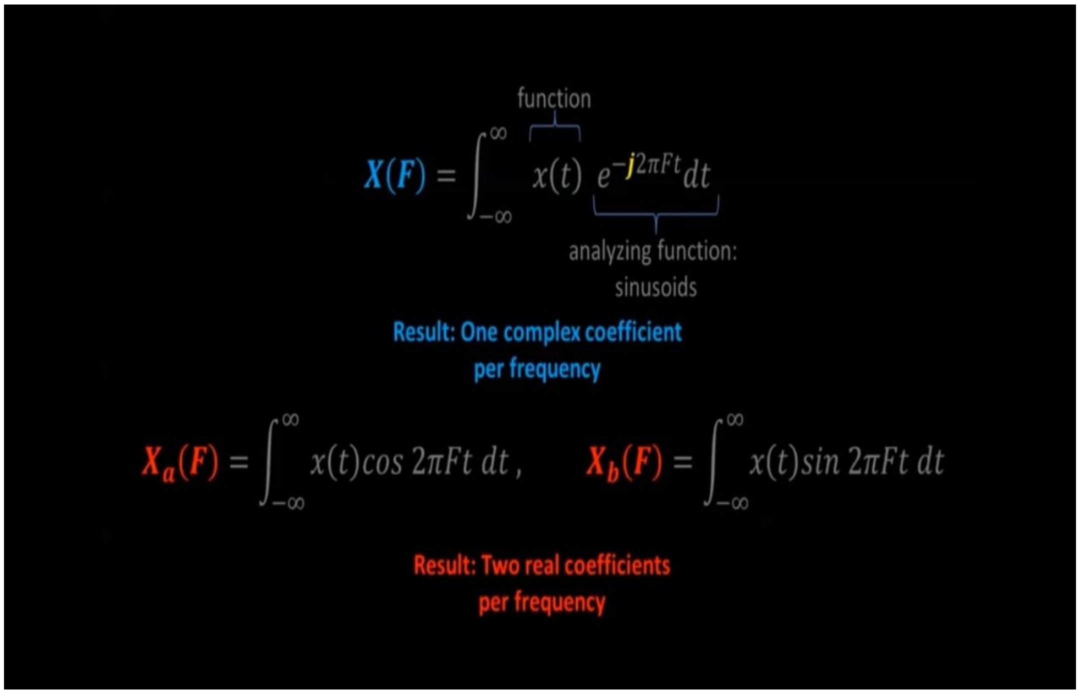

$-\pi$
π
$f(x) = \sum_{k=1}^{\infty} a_k \cos\left(k\frac{2\pi x}{L}\right) + b_k \sin\left(k\frac{2\pi x}{L}\right)$
$a_k = \langle f(x), \cos\left(k\frac{2\pi x}{L}\right)\rangle$
$b_k = \langle f(x), \sin\left(k\frac{2\pi x}{L}\right)\rangle$

```
figure
set(gcf,'Position',[1500 200 2000 1200])

% Define domain
L = pi;
N = 1024;
dx = 2*L/(N-1);
x = -L:dx:L;

% Define hat function
f = 0*x;
f(N/4:N/2) = 4*(1:N/4+1)/N;
f(N/2+1:3*N/4) = 1-4*(0:N/4-1)/N;
plot(x,f,'-k','LineWidth',3.5), hold on

%% Compute Fourier series
CC = jet(20);
A0 = sum(f.*ones(size(x)))*dx/pi;
fFS = A0/2;
for k=1:20
    A(k) = sum(f.*cos(pi*k*x/L))*dx/pi; % Inner product
    B(k) = sum(f.*sin(pi*k*x/L))*dx/pi;
    fFS = fFS + A(k)*cos(k*pi*x/L) + B(k)*sin(k*pi*x/L);
    plot(x,fFS,'-','Color',CC(k,:),'LineWidth',2)
    pause(.1)
end


%% Plot amplitudes
figure; set(gcf,'Position',[1500 200 2000 1200])
clear ERR
```

```
fx >>
```

```
B    1x20 ...
CC   20x3 ...
dx   0.0061
f    1x10...
fFS  1x10...
k    20
L    3.1416
N    1024
x    1x10...
```

Fig 2 : using matlab



$$X(F) = \int_{-\infty}^{\infty} \overbrace{x(t)}^{\text{function}} \underbrace{e^{-j2\pi Ft} dt}_{\text{analyzing function: sinusoids}}$$

Result: One complex coefficient per frequency

$$X_a(F) = \int_{-\infty}^{\infty} x(t) \cos 2\pi Ft \, dt, \qquad X_b(F) = \int_{-\infty}^{\infty} x(t) \sin 2\pi Ft \, dt$$

Result: Two real coefficients per frequency

Fig 3: using python implementation

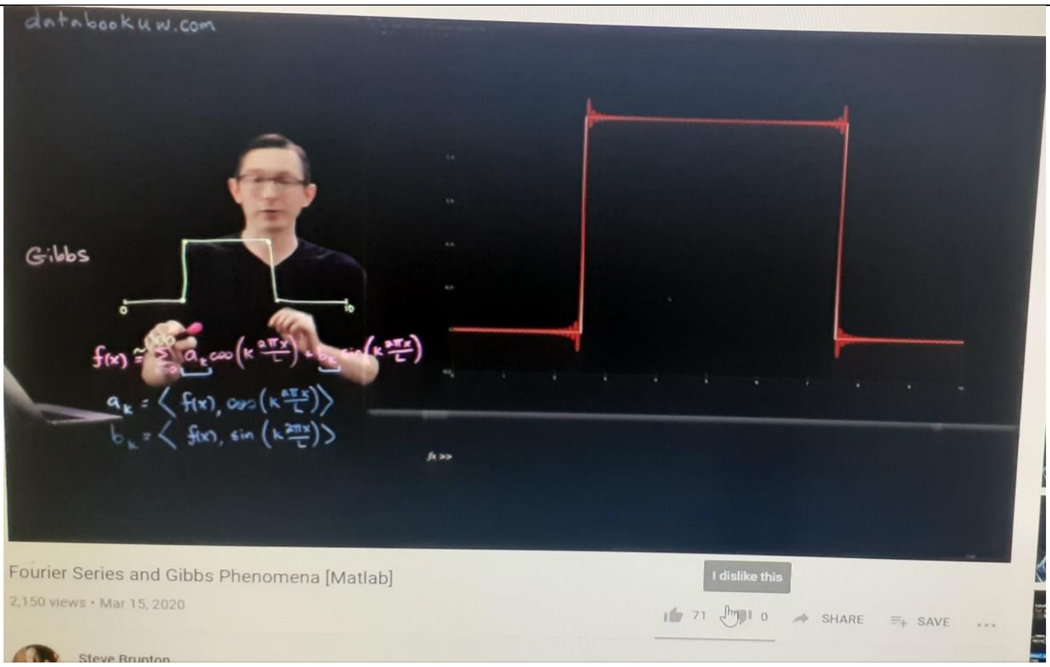

databookuw.com
Gibbs
$f(x) \approx \sum a_k \cos(k\frac{2\pi x}{L}) + b_k \sin(k\frac{2\pi x}{L})$
$a_k = \langle f(x), \cos(k\frac{2\pi x}{L})\rangle$
$b_k = \langle f(x), \sin(k\frac{2\pi x}{L})\rangle$
Fourier Series and Gibbs Phenomena [Matlab]
2,150 views • Mar 15, 2020
I dislike this
71
0
SHARE
SAVE
Steve Brunton

Fig 4: using ghibs phenomena using matlab

Fourier series

A Fourier series is an expansion of a periodic function f(x) in terms of an infinite sum of sines and cosines. Fourier series make use of the orthogonality relationships of the sine and cosine functions. The computation and study of Fourier series is known as harmonic analysis and is extremely useful as a way to break up an arbitrary periodic function into a set of simple terms that can be plugged in, solved individually, and then recombined to obtain the solution to the original problem or an approximation to it to whatever accuracy is desired or practical.

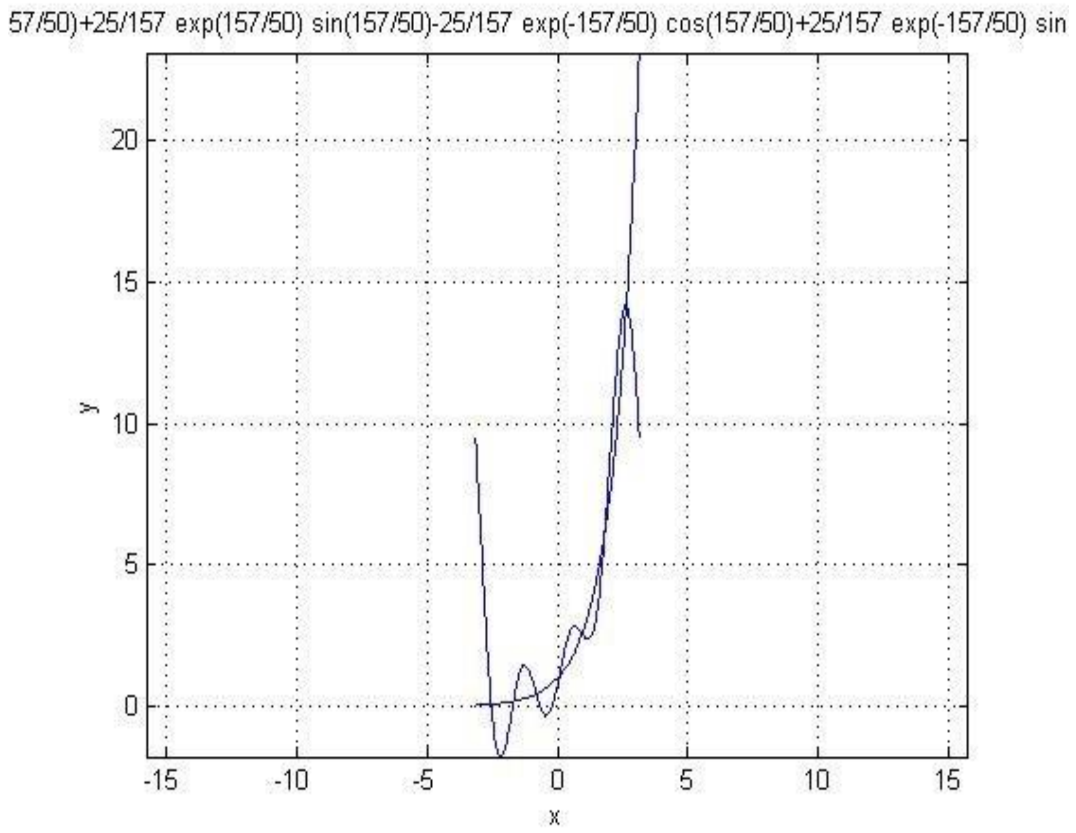
MATLAB

We can also have MATLAB calculuate the general Fourier coefficients. To do this and get MATLAB to simplify the results, we can use simple. The following command gives the kth Fourier cosine coefficient, suppressing the results of all of the steps of simple except for the simplest.

First of all, find the coefficients of fourier series ao,an,bn.Here is the matlab code:

```
clearall;clc;
symsx
pi=3.14;
sum=0;
y=exp(x);

%functionyouwant
a0=(1/pi)*Int(y,x,-pi,pi);
forn=1:3
%findingthecoefficients
an=(1/pi)*int(y*cos(n*x),x,-pi,pi);
bn=(1/pi)*int(y*sin(n*x),x,-pi,pi);
sum=sum+(an*cos(n*x)+bn*sin(n*x));
end
ezplot(x,y,[-pi,pi]);
gridon;

holdon;
ezplot(x,(sum+a0/2),[-pi,pi]);
```

where a0 models a constant (intercept) term in the data and is associated with the $i = 0$ cosine term, w is the fundamental frequency of the signal, n is the number of terms (harmonics) in the series, and $1 \leq n \leq 8$.



FOURIER USING PYTHON

PYTHON CODE:

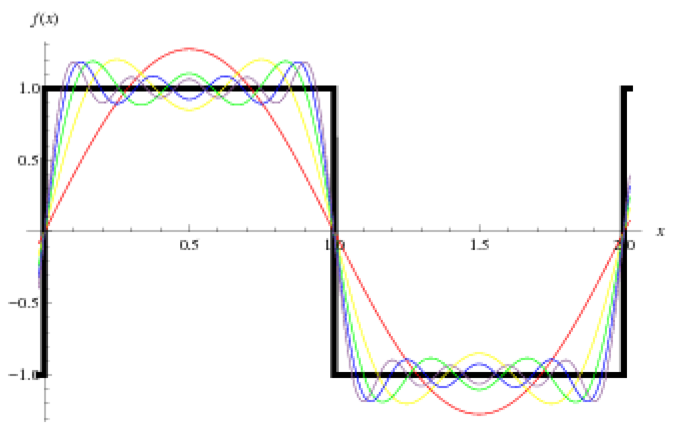
The following example for complex numbers

```
x=complex(1,2)

print x

y=complex(3,4)

print y

z=x+y
```

```
print x

print z

z=x*y

print z

z=x/y

print z

print x.conjugate()

print x.imag

print x.real

print x>y

Traceback (most recent call last):

File "<pyshell#149>", line 1, in <module>

print x>y

TypeError: no ordering relation is defined for complex numbers

print x==y

False
```

Ghibs phenomena

From a signal processing point of view, the Gibbs phenomenon is the step response of a low-pass filter, and the oscillations are called ringing or ringing artifacts. Truncating the Fourier transform of a signal on the real line, or the Fourier series of a periodic signal corresponds to filtering out the higher frequencies by an ideal low-pass/high-cut filter. This can be represented as convolution of the original signal with the impulse response of the filter ,which is the sinc function



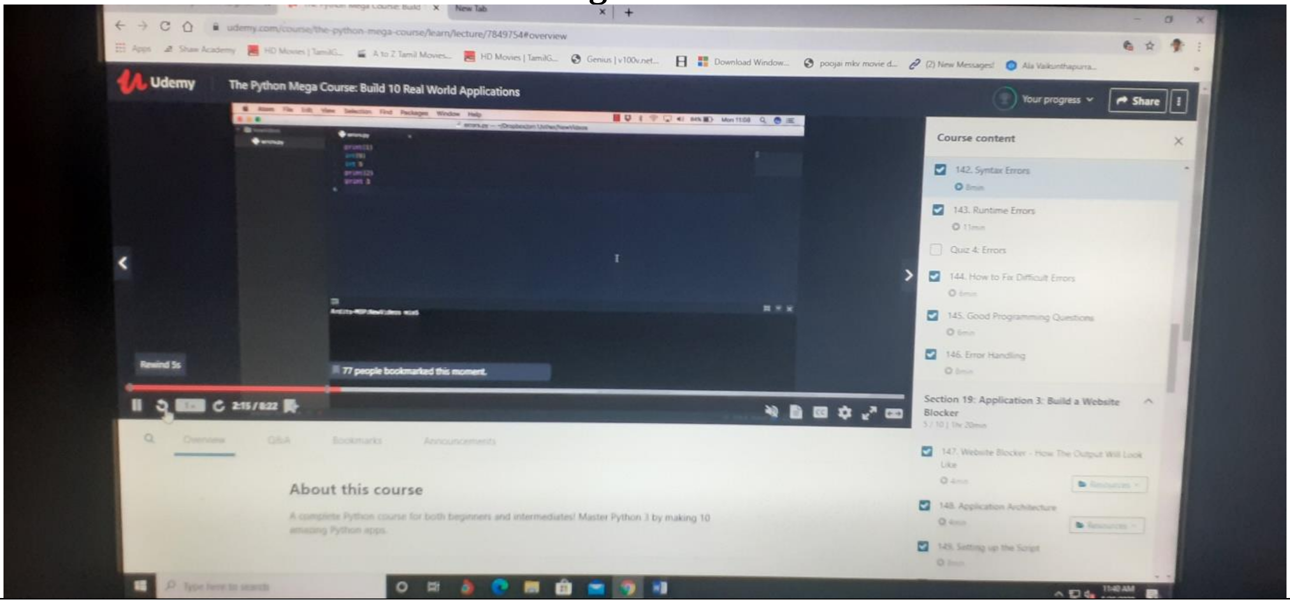
the features of the Gibbs phenomenon are interpreted as follows:

- the undershoot is due to the impulse response having a negative tail integral, which is possible because the function takes negative values;
- the overshoot offsets this, by symmetry (the overall integral does not change under filtering);
- the persistence of the oscillations is because increasing the cutoff narrows the impulse response, but does not reduce its integral – the oscillations thus move towards the discontinuity, but do not decrease in magnitude.

AFTERNOON SESSION DETAILS

Date:	**25-5-2020**	**Name:**	**Kavyashree m**
Course:	**Python programming**	**USN:**	**4al15ec036**
Topic:	**Fixing program errors,Build a website blocker**	**Semester & Section:**	**8th A**
Github Repository:	**Kavya**		

Image of session


Udemy
The Python Mega Course: Build 10 Real World Applications
Your progress
Share
Course content
142. Syntax Errors
143. Runtime Errors
Quiz 4: Errors
144. How to Fix Difficult Errors
145. Good Programming Questions
146. Error Handling
Section 19: Application 3: Build a Website Blocker
147. Website Blocker - How The Output Will Look Like
148. Application Architecture
149. Setting up the Script
77 people bookmarked this moment.
Rewind 5s
2:15 / 8:22
About this course

Fixing program errors

Errors

Errors or mistakes in a program are often referred to as bugs. They are almost always the fault of the programmer. The process of finding and eliminating errors is called debugging. Errors can be classified into three major groups:

- Syntax errors
- Runtime errors
- Logical errors

Syntax errors

Python will find these kinds of errors when it tries to parse your program, and exit with an error message without running anything. Syntax errors are mistakes in the use of the Python language, and are analogous to spelling or grammar mistakes in a language like English: for example, the sentence Would you some tea? does not make sense – it is missing a verb.

Common Python syntax errors include:

- leaving out a keyword
- putting a keyword in the wrong place
- leaving out a symbol, such as a colon, comma or brackets
- misspelling a keyword
- incorrect indentation
- empty block

Python will do its best to tell you where the error is located, but sometimes its messages can be misleading: for example, if you forget to escape a quotation mark inside a string you may get a syntax error referring to a place later in your code, even though that is not the real source of the problem. If you can't see anything wrong on the line specified in the error message, try backtracking through the previous few lines. As you program more, you will get better at identifying and fixing errors.

Here are some examples of syntax errors in Python:

```
myfunction(x, y):
    return x + y


else:
    print("Hello!")

if mark >= 50
    print("You passed!")

if arriving:
    print("Hi!")
esle:
    print("Bye!")

if flag:
print("Flag is set!")
```

Runtime errors

If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter. However, the program may exit unexpectedly during execution if it encounters a runtime error – a problem which was not detected when the program was parsed, but is only revealed when a particular line is executed. When a program comes to a halt because of a runtime error, we say that it has crashed.

Consider the English instruction flap your arms and fly to Australia. While the instruction is structurally correct and you can understand its meaning perfectly, it is impossible for you to follow it.

Some examples of Python runtime errors:

division by zero

performing an operation on incompatible types

using an identifier which has not been defined

accessing a list element, dictionary value or object attribute which doesn’t exist

trying to access a file which doesn’t exist

Runtime errors often creep in if you don’t consider all possible values that a variable could contain, especially when you are processing user input. You should always try to add checks to your code to make sure that it can deal with bad input and edge cases gracefully. We will look at this in more detail in the chapter about exception handling.

How to Fix Errors in Python

The Python interpreter takes in each line and operates on it immediately (more or less) after you press the Enter key. In Hello World! you use Python’s print feature. print takes what’s inside the parentheses and outputs it to the command line (also called the *console*).

Python is sensitive to both the grammar and punctuation. If you misspell something, the program won’t work. If Python is expecting special characters and you don’t put them in, then Python will fail. Some Python issues are shown here. Can you work out how you would fix them?

```
>>> pritn('Hello World!')
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  NameError: name 'pritn' is not defined
```

Error handling

Python has many built-in exceptions that are raised when your program encounters an error.

When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

For example, let us consider a program where we have a function A that calls function B, which in turn calls function C. If an exception occurs in function C but is not handled in C, the exception passes to B and then to A.

If never handled, an error message is displayed and our program comes to a sudden unexpected halt.

Catching Exceptions in Python

In Python, exceptions can be handled using a try statement.

The critical operation which can raise an exception is placed inside the try clause. The code that handles the exceptions is written in the except clause.

We can thus choose what operations to perform once we have caught the exception. Here is a simple example.

```
# import module sys to get the type of exception
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

Run Code

Output

```
The entry is a
Oops! <class 'ValueError'> occurred.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError'> occured.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

Build a website blocker

The script web-blocker.py is given below.

web-blocker.py

```
from time import *
from datetime import *

host_path = r"/etc/hosts"
redirect = "127.0.0.1"
websites = ["www.facebook.com", "https://www.facebook.com"]
```

```
while True:
    if
datetime(datetime.now().year,datetime.now().month,datetime.now().day,9)<datetime.n
ow()<datetime(datetime.now().year,datetime.now().month,datetime.now().day,17):
        with open(host_path,"r+") as fileptr:
            content = fileptr.read()
            for website in websites:
                if website in content:
                    pass
                else:
                    fileptr.write(redirect+"        "+website+"\n")
    else:
        with open(host_path,'r+') as file:
            content = file.readlines();
            file.seek(0)
            for line in content:
                if not any(website in line for website in               websites):
                    file.write(line)
            file.truncate()
    sleep(5)
```