

Let’s us try to approximate some of the function using Fourier series

// CODE

1. Fourier Series Approximation code part

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.cm import get\_cmap

# Define domain

dx = 0.001

L = np.pi #We can change the limit of function from -pi to pi to any other domain -l to l as we want

x = L \* np.arange(-1+dx,1+dx,dx) # This represent the x axis as discontinuos point at dx interval

n = len(x) # no of points from -L to L at discreate value of dx intervals

f = np.zeros\_like(x) #Gives zeros of same data type as of x

f = x\*x\*(np.sin(x))

fig, ax = plt.subplots()

ax.plot(x,f,'-',color='k')

# Compute Fourier series

name = "Accent"

cmap = get\_cmap('tab10')

colors = cmap.colors

ax.set\_prop\_cycle(color=colors)

A0 = np.sum(f \* np.ones\_like(x)) \* dx

fFS = A0/2

A = np.zeros(200)

B = np.zeros(200)

# Summing the series of sine and consine with inner product of f(x)

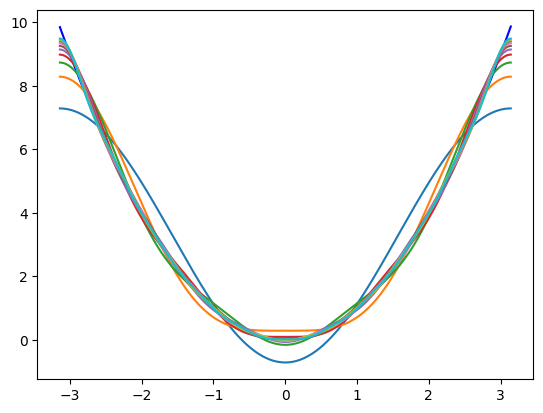
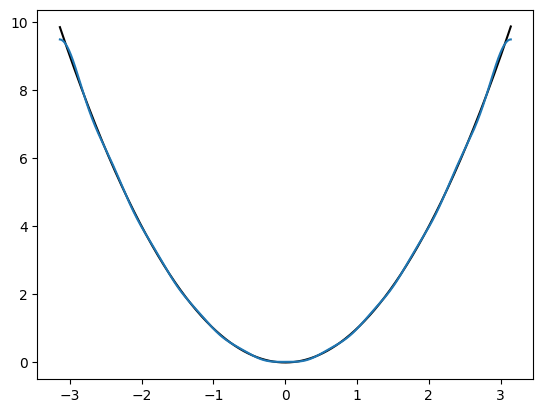
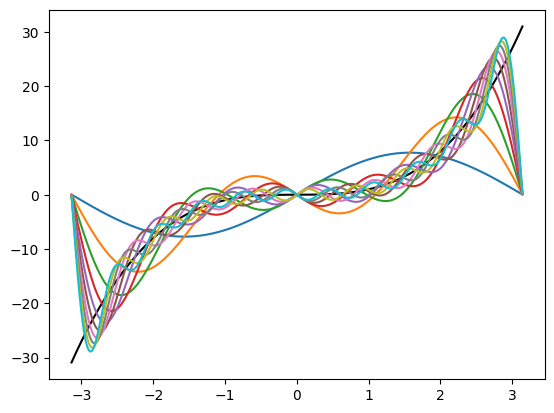
for k in range(0,5):

    A[k] = np.sum(f \* np.cos(np.pi\*(k+1)\*x/L)) \* dx # Inner product

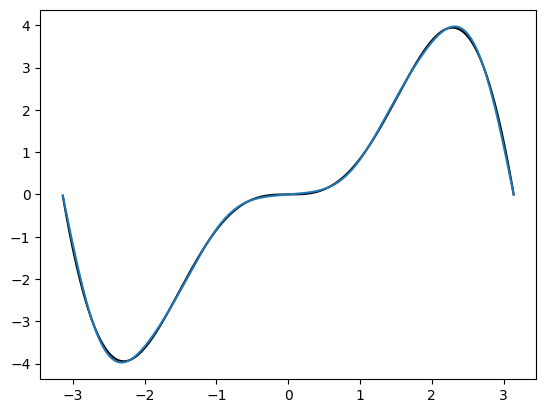
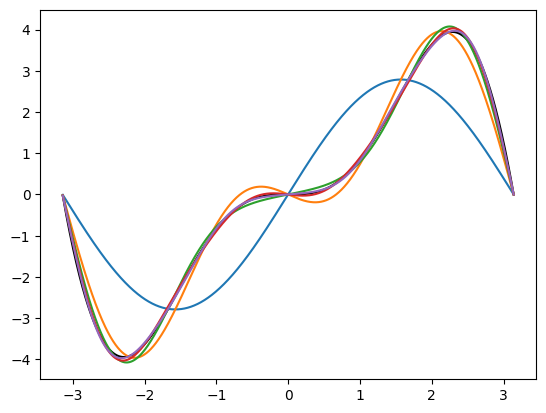
    B[k] = np.sum(f \* np.sin(np.pi\*(k+1)\*x/L)) \* dx

    fFS = fFS + A[k]\*np.cos((k+1)\*np.pi\*x/L) + B[k]\*np.sin((k+1)\*np.pi\*x/L)

    ax.plot(x,fFS,'-')

1. Fourier Series approximation of x^2 with limit of k = 10  
2. Approximating x^3 using the Fourier Series approximation 

For k = 20

1. X^2 sin(x)
2. 

Representation of an image

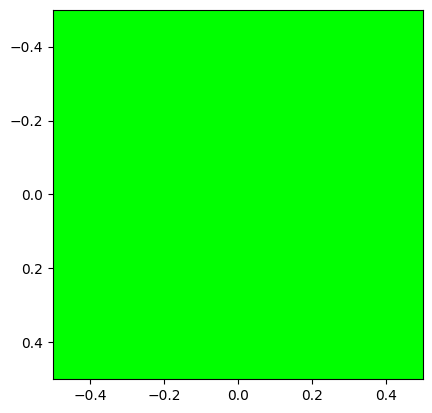
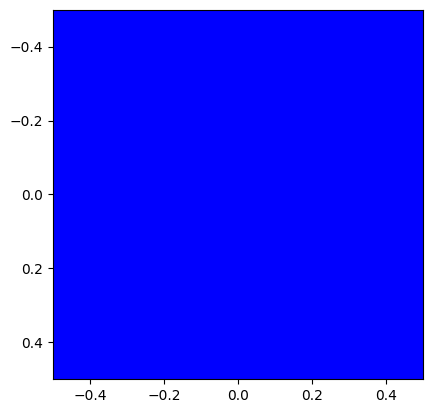
# Red Green Blue

Code

ar3 = np.zeros(shape=(1,1,3))

ar3[:,:,:] = [0,0,255]

plt.imshow(ar3)

Red  Green  Blue 

[[[ 255, 0, 0]]] [[[ 0, 255, 0]]] [[[ 0, 0, 255]]]

Code

limit = 20

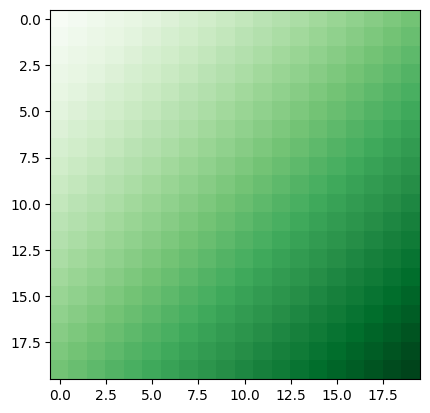
x = np.linspace(1,100,limit)

y = np.linspace(1,100,limit)

x,y = np.meshgrid(x,y)

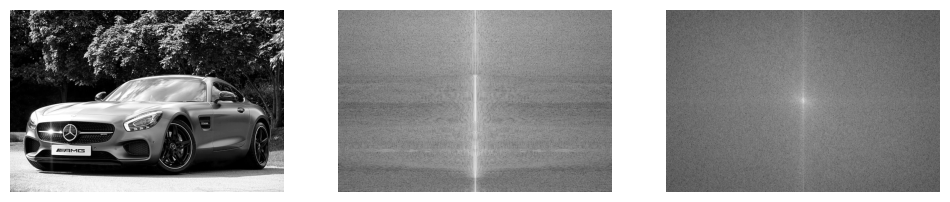
z = x+y

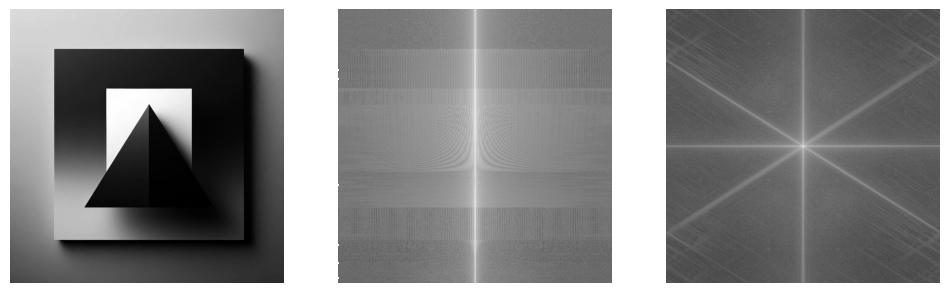
plt.imshow(z,cmap="Greens")



Fourier Transformation of images

Original image : Grayscale Row-wise Fourier Transform Column wise Fourier Transform





Code for Fourier Transformation of image

from matplotlib.image import imread

import numpy as np

import matplotlib.pyplot as plt

import os

Original\_Image = imread(os.path.join(r"images\triangle.jpeg"))

Gray\_Image = np.mean(Original\_Image, -1); # Convert RGB to grayscale

FIgure,axs = plt.subplots(1,3)

# Plot image

Image = axs[0].imshow(Gray\_Image)

Image.set\_cmap('gray')

axs[0].axis('off')

# Compute row-wise FFT

Cshift = np.zeros\_like(Gray\_Image,dtype='complex\_')

C = np.zeros\_like(Gray\_Image,dtype='complex\_')

for j in range(Gray\_Image.shape[0]):

    Cshift[j,:] = np.fft.fftshift(np.fft.fft(Gray\_Image[j,:]))

    C[j,:] = np.fft.fft(Gray\_Image[j,:])

Image = axs[1].imshow(np.log(np.abs(Cshift)))

Image.set\_cmap('gray')

axs[1].axis('off')

# Compute column-wise FFT

D = np.zeros\_like(C)

for j in range(C.shape[1]):

    D[:,j] = np.fft.fft(C[:,j])

Image = axs[2].imshow(np.fft.fftshift(np.log(np.abs(D))))

Image.set\_cmap('gray')

axs[2].axis('off')

plt.show()

More efficient and convenient way to find Fourier Transform is:

Discrete Fourier Transform (**[numpy.fft](https://numpy.org/doc/stable/reference/routines.fft.html" \l "module-numpy.fft" \o "numpy.fft)**)

The SciPy module **[scipy.fft](https://docs.scipy.org/doc/scipy/reference/fft.html" \l "module-scipy.fft" \o "(in SciPy v1.11.2))** is a more comprehensive superset of numpy.fft, which includes only a basic set of routines.

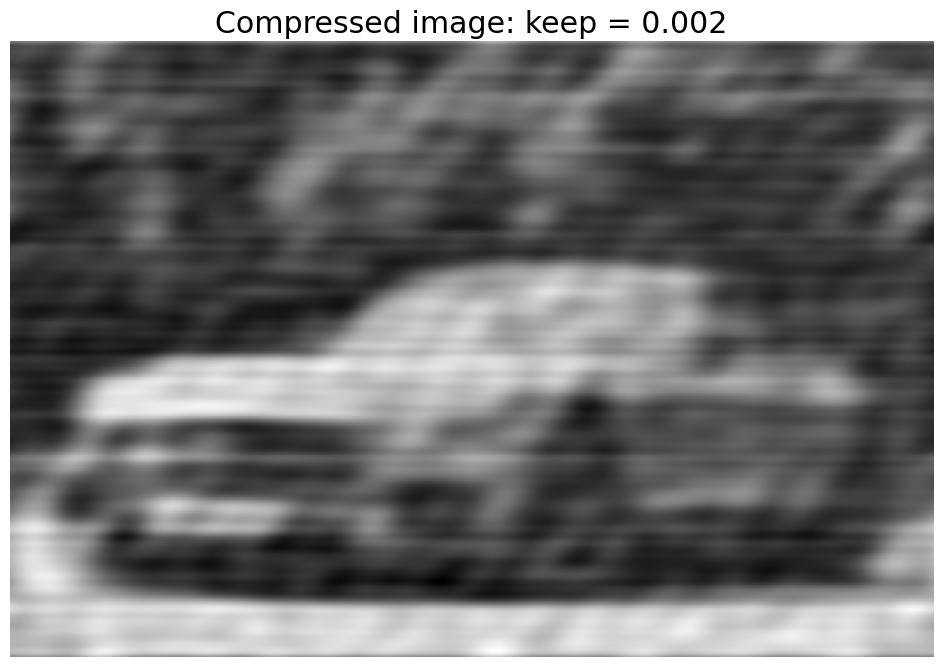
Standard FFTs

|  |  |
| --- | --- |
| [**fft**](https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html#numpy.fft.fft)(a[, n, axis, norm]) | Compute the one-dimensional discrete Fourier Transform. |
| [**ifft**](https://numpy.org/doc/stable/reference/generated/numpy.fft.ifft.html#numpy.fft.ifft)(a[, n, axis, norm]) | Compute the one-dimensional inverse discrete Fourier Transform. |
| [**fft2**](https://numpy.org/doc/stable/reference/generated/numpy.fft.fft2.html#numpy.fft.fft2)(a[, s, axes, norm]) | Compute the 2-dimensional discrete Fourier Transform. |
| [**ifft2**](https://numpy.org/doc/stable/reference/generated/numpy.fft.ifft2.html#numpy.fft.ifft2)(a[, s, axes, norm]) | Compute the 2-dimensional inverse discrete Fourier Transform. |

Image Compression using Fast Fourier Transform



Code :

from matplotlib.image import imread

import numpy as np

import matplotlib.pyplot as plt

import os

plt.rcParams['figure.figsize'] = [12, 8]

plt.rcParams.update({'font.size': 18})

A = imread(os.path.join(r'images\car01.jpg'))

B = np.mean(A, -1); # Convert RGB to grayscale

plt.imshow(A)

plt.axis('off')

plt.title('Original Image')

Bt = np.fft.fft2(B)

Btsort = np.sort(np.abs(Bt.reshape(-1))) # sort by magnitude

# Zero out all small coefficients and inverse transform

for keep in (0.1, 0.05, 0.01, 0.002):

    thresh = Btsort[int(np.floor((1-keep)\*len(Btsort)))]

    ind = np.abs(Bt)>thresh          # Find small indices

    Atlow = Bt \* ind                 # Threshold small indices

    Alow = np.fft.ifft2(Atlow).real  # Compressed image

    plt.figure()

    plt.imshow(Alow,cmap='gray')

    plt.axis('off')

    plt.title('Compressed image: keep = ' + str(keep))

INDEX:

1. Theory
2. Fourier Series
3. Fourier Transformation
4. Discrete Fourier Transform
5. Image representation
6. Denoising
7. Image Compression using Fourier Transform
8. Edge detection

\documentclassarticle\ \usepackageamsmath\ \ \usepackage{amssymb}\ \begin{document}\ %\ Define\ N\ and\ the\ DFT\ matrix\ elements\ \newcommand{\N}{4}\ %\ Change\ this\ to\ your\ desired\ N\ \ \ \newcommand{\w}{e^{-\frac{2\pi i}{\N}}}\ \ \newcommand{\dftentry}[2]{⍁{1}{√{\N}}\w^{{#1} ⋅{#2}}} % Define N and the DFT matrix elements \newcommand{\N}{4} % Change this to your desired N \newcommand{\w}{e^{-⍁{2πi}{\N}}} \newcommand{\dftentry}[2]{⍁{1}{√{\N}}\w^{{#1} ⋅{#2}}} % Begin the matrix [ F\_{\N} = ⍁{1}{√{\N}} 〖{bmatrix} 1 & 1 & 1 & ⋯& 1 \\ 1 & \w & \w^2 & ⋯& \w^{\N-1} \\ 1 & \w^2 & \w^4 & ⋯& \w^{2(\N-1)} \\ ⋮& ⋮& ⋮& ⋱& ⋮\\ 1 & \w^{\N-1} & \w^{2(\N-1)} & ⋯& \w^{(\N-1)^2} 〗{bmatrix} ] 〗{document}

