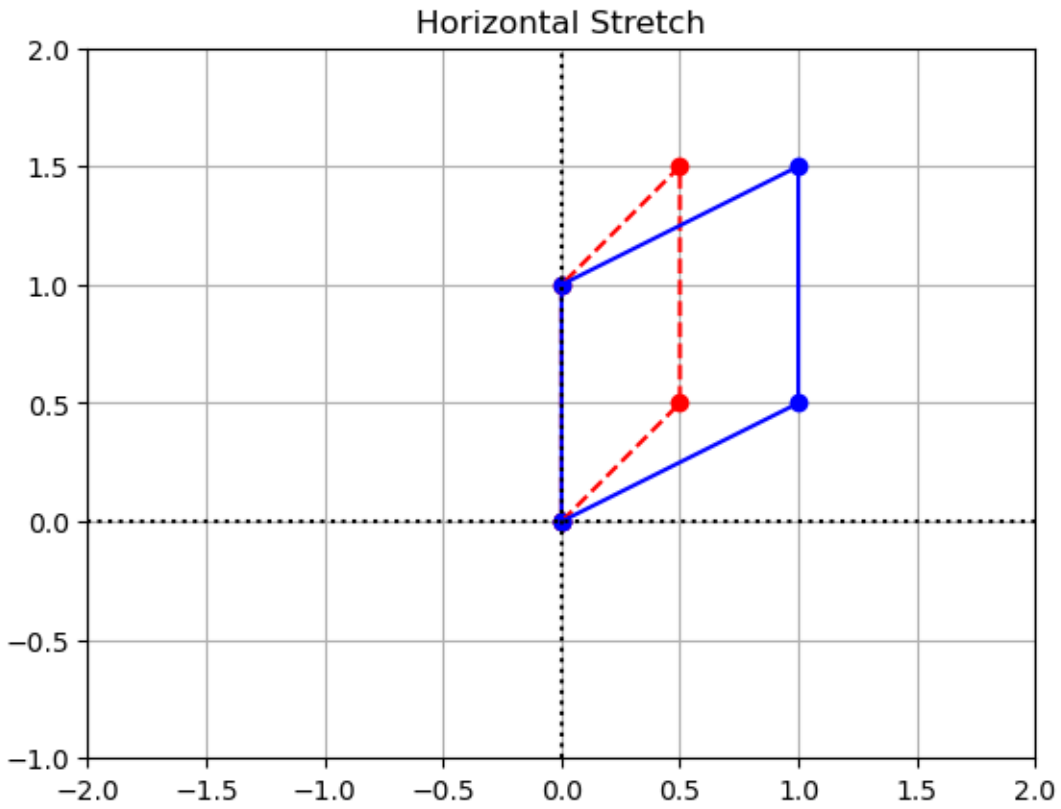
Third semester lab programs Maths Department MCE

July 18, 2024

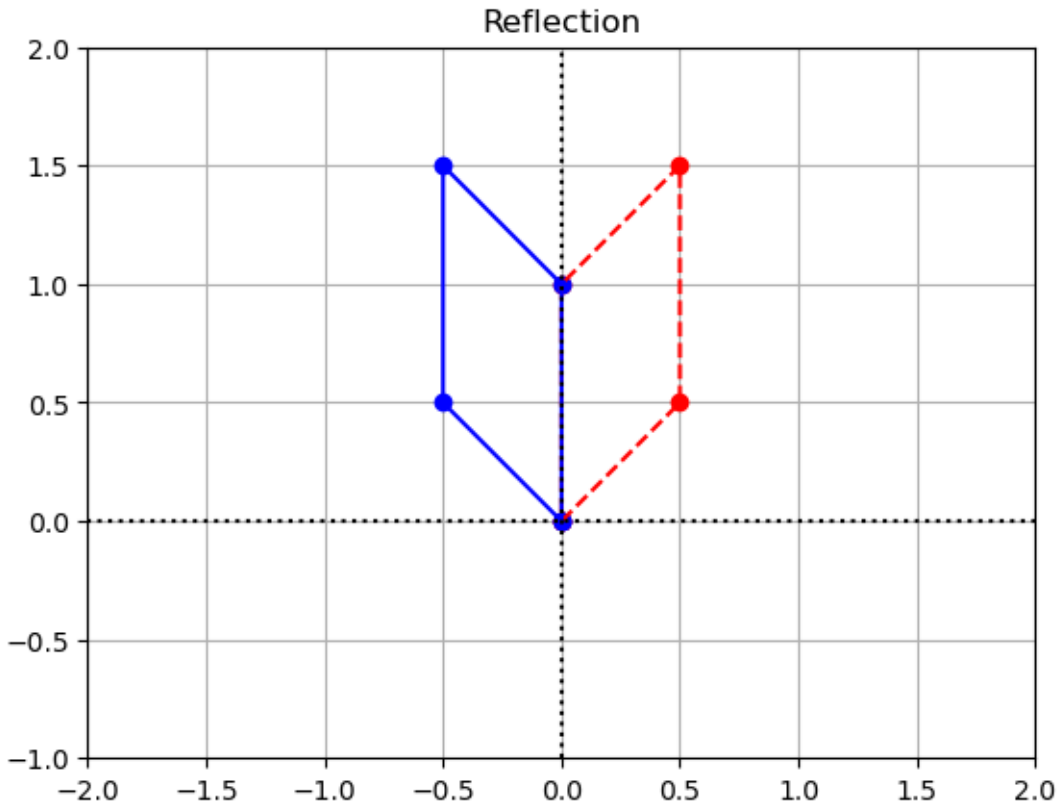
[7]:
```
%matplotlib inline
from math import pi, sin, cos
import matplotlib.pyplot as plt
import numpy as np

coords = np.array([[0,0],[0.5,0.5],[0.5,1.5],[0,1],[0,0]])
coords = coords.transpose()
coords
x = coords[0,:]
y = coords[1,:]

A = np.array([[2,0],[0,1]])
A_coords = A@coords
```

[2]:
```
x_LT1 = A_coords[0,:]
y_LT1 = A_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are images
ax.plot(x,y,'ro')
ax.plot(x_LT1,y_LT1,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT1,y_LT1,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Horizontal Stretch");
```



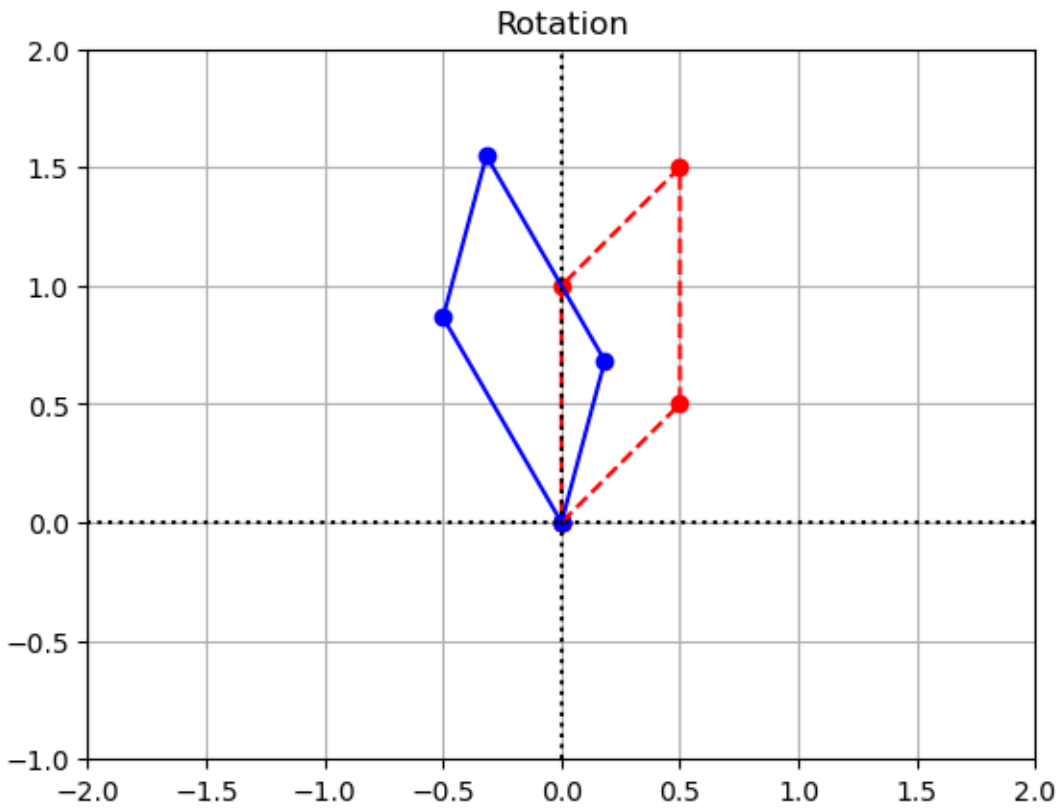
[3]:
```
B = np.array([[-1,0],[0,1]])
B_coords = B@coords

x_LT2 = B_coords[0,:]
y_LT2 = B_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are images
ax.plot(x,y,'ro')
ax.plot(x_LT2,y_LT2,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT2,y_LT2,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
```

```
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Reflection");
```



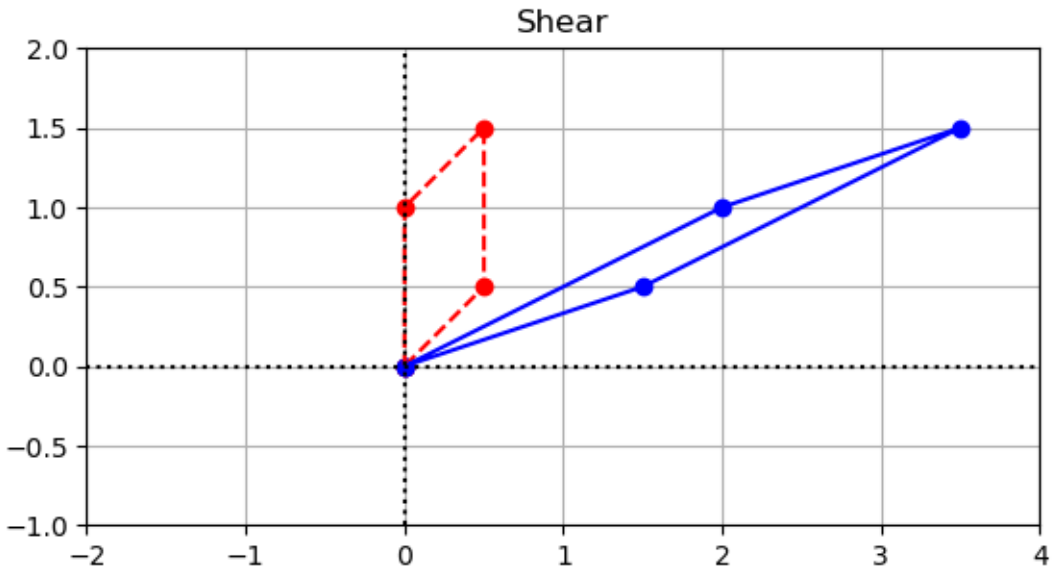
[4]:
```
theta = pi/6
R = np.array([[cos(theta),-sin(theta)],[sin(theta),cos(theta)]])
R_coords = R@coords

x_LT3 = R_coords[0,:]
y_LT3 = R_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are images
ax.plot(x,y,'ro')
ax.plot(x_LT3,y_LT3,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT3,y_LT3,'b')
```

```
# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Rotation");
```



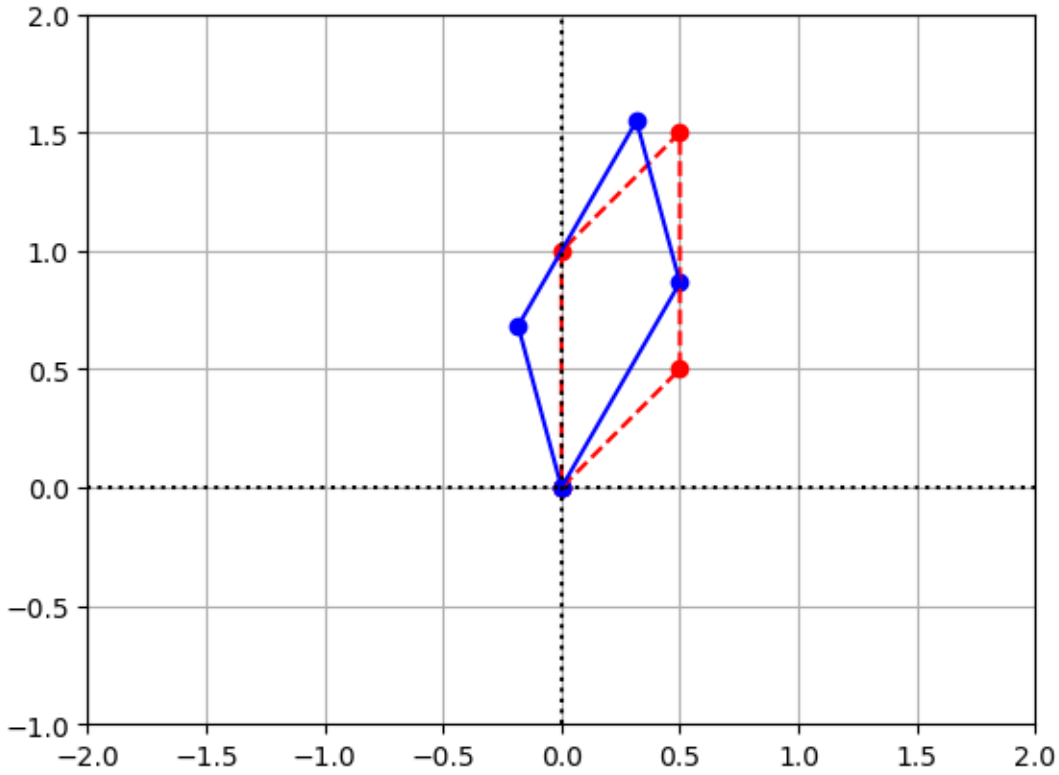
[5]:
```
S = np.array([[1,2],[0,1]])
S_coords = S@coords

x_LT4 = S_coords[0,:]
y_LT4 = S_coords[1,:]


# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are images
ax.plot(x,y,'ro')
```

```
ax.plot(x_LT4,y_LT4,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT4,y_LT4,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,4,-1,2])
ax.set_aspect('equal')
ax.set_title("Shear");
```



[6]:
```
C = np.array([[-cos(theta),sin(theta)],[sin(theta),cos(theta)]])
C_coords = C@coords

x_LT5 = C_coords[0,:]
y_LT5 = C_coords[1,:]


# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are images
ax.plot(x,y,'ro')
ax.plot(x_LT5,y_LT5,'bo')
```

```
# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT5,y_LT5,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
```



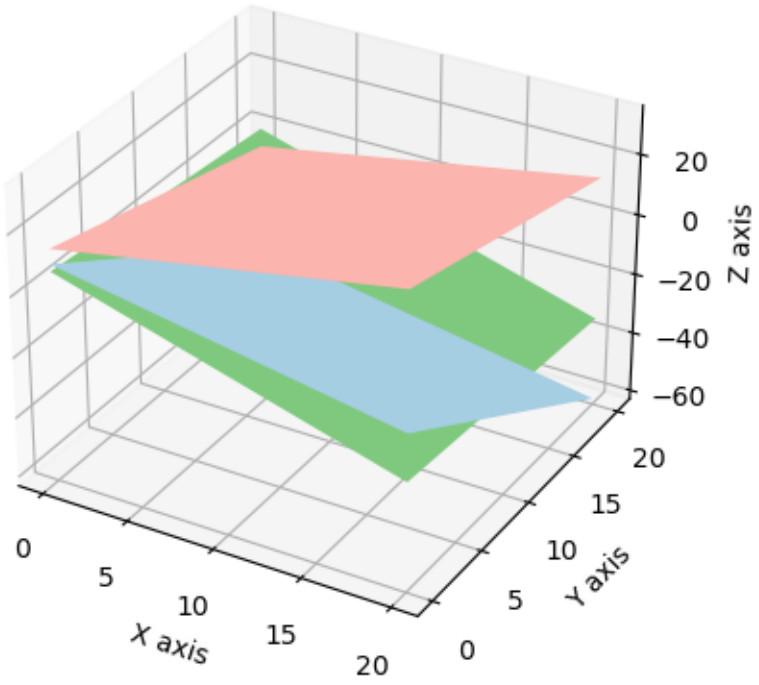
[1]:
```
import numpy as np

x1, y1, z1, w1 = 5, 1, 3, 20

# Take the input for equation-2
x2, y2, z2, w2 = 2, 5, 2, 18

# Take the input for equation-3
x3, y3, z3, w3 = 3, 2, 1, 14
```

```
# Create an array for LHS variables
LHS = np.array([[x1, y1, z1],
                [x2, y2, z2],
                [x3, y3, z3]])

# Create another array for RHS variables
RHS = np.array([w1, w2, w3])

# Apply linear algebra on any numpy
# array created and printing the output
sol = np.linalg.solve(LHS, RHS)
print(sol)
```

```
[3. 2. 1.]
```

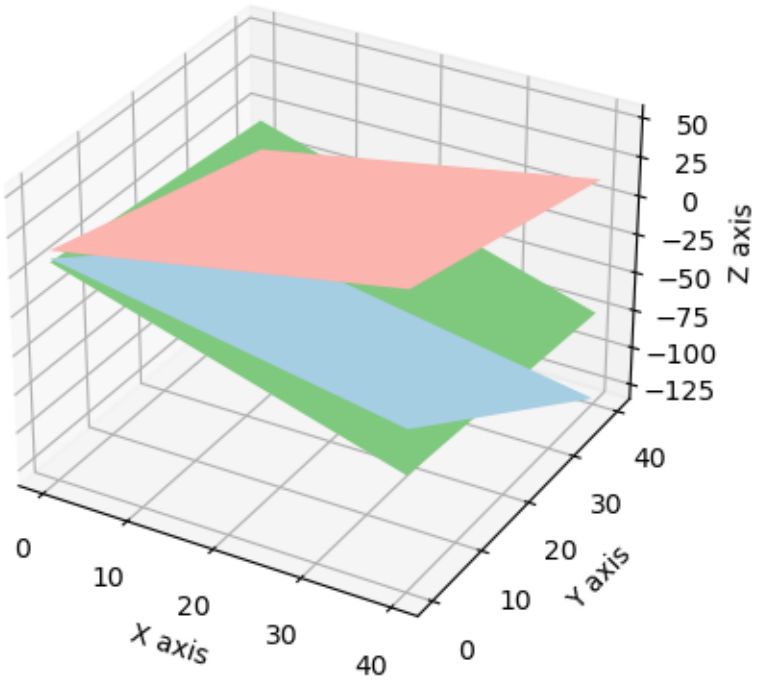
[2]:
```
import matplotlib.pyplot as plt
from matplotlib import cm

# Returns number spaces evenly w.r.t
# interval
x_axis, y_axis = np.linspace(0, 20, 10),  np.linspace(0, 20, 10)

# Create a rectangular grid out of
# two given one-dimensional arrays
X, Y = np.meshgrid(x_axis, y_axis)

# Make a rectangular grid
# 3-dimensional by calculating z1, z2, z3
Z1 = (w1-x1*X-y1*Y)/z1
Z2 = (w2-x2*X-y2*Y)/z2
Z3 = (w3+X-Y)/z3

# Create 3D graphics and add
# an add an axes to the figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create a 3D Surface Plot
ax.plot_surface(X, Y, Z1, alpha=1,
                cmap=cm.Accent,
                rstride=100, cstride=100)
ax.plot_surface(X, Y, Z2, alpha=1,
                cmap=cm.Paired,
                rstride=100, cstride=100)
ax.plot_surface(X, Y, Z3, alpha=1,
                cmap=cm.Pastel1,
                rstride=100, cstride=100)
```

```
# Draw points and make lines
ax.plot((sol[0],), (sol[1],), (sol[2],),
        lw=2, c='k', marker='o',
        markersize=7, markeredgecolor='g',
        markerfacecolor='white')

# Set the label for x-axis, y-axis and
# z-axis
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

# Display all figures
plt.show()
```



[3]:
```
import matplotlib.pyplot as plt
from matplotlib import cm

# Returns number spaces evenly w.r.t
# interval
x_axis, y_axis = np.linspace(0, 40, 10), np.linspace(0, 40, 10)
```

```
# Create a rectangular grid out of
# two given one-dimensional arrays
X, Y = np.meshgrid(x_axis, y_axis)

# Make a rectangular grid
# 3-dimensional by calculating z1, z2, z3
Z1 = (w1-x1*X-y1*Y)/z1
Z2 = (w2-x2*X-y2*Y)/z2
Z3 = (w3+X-Y)/z3

# Create 3D graphics and add
# an add an axes to the figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create a 3D Surface Plot
ax.plot_surface(X, Y, Z1, alpha=1,
                                cmap=cm.Accent,
                                rstride=100, cstride=100)
ax.plot_surface(X, Y, Z2, alpha=1,
                                cmap=cm.Paired,
                                rstride=100, cstride=100)
ax.plot_surface(X, Y, Z3, alpha=1,
                                cmap=cm.Pastel1,
                                rstride=100, cstride=100)

# Draw points and make lines
ax.plot((sol[0],), (sol[1],), (sol[2],),
                lw=2, c='k', marker='o',
                markersize=7, markeredgecolor='g',
                markerfacecolor='white')

# Set the label for x-axis, y-axis and
# z-axis
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

# Display all figures
plt.show()
```



[4]:
```python
# Python 3 program to find rank of a matrix
class rankMatrix(object):
    def __init__(self, Matrix):
        self.R = len(Matrix)
        self.C = len(Matrix[0])

    # Function for exchanging two rows of a matrix
    def swap(self, Matrix, row1, row2, col):
        for i in range(col):
            temp = Matrix[row1][i]
            Matrix[row1][i] = Matrix[row2][i]
            Matrix[row2][i] = temp

    # Function to Display a matrix
    def Display(self, Matrix, row, col):
        for i in range(row):
            for j in range(col):
                print (" " + str(Matrix[i][j]))
            print ('\n')

    # Find rank of a matrix
    def rankOfMatrix(self, Matrix):
        rank = self.C
```

```
        for row in range(0, rank, 1):

            # Before we visit current row
            # 'row', we make sure that
            # mat[row][0],....mat[row][row-1]
            # are 0.

            # Diagonal element is not zero
            if Matrix[row][row] != 0:
                for col in range(0, self.R, 1):
                    if col != row:

                        # This makes all entries of current
                        # column as 0 except entry 'mat[row][row]'
                        multiplier = (Matrix[col][row] /
                                      Matrix[row][row])
                        for i in range(rank):
                            Matrix[col][i] -= (multiplier *
                                               Matrix[row][i])

            # Diagonal element is already zero.
            # Two cases arise:
            # 1) If there is a row below it
            # with non-zero entry, then swap
            # this row with that row and process
            # that row
            # 2) If all elements in current
            # column below mat[r][row] are 0,
            # then remove this column by
            # swapping it with last column and
            # reducing number of columns by 1.
            else:
                reduce = True

                # Find the non-zero element
                # in current column
                for i in range(row + 1, self.R, 1):

                    # Swap the row with non-zero
                    # element with this row.
                    if Matrix[i][row] != 0:
                        self.swap(Matrix, row, i, rank)
                        reduce = False
                        break

                # If we did not find any row with
                # non-zero element in current
```

```
                # column, then all values in
                # this column are 0.
                if reduce:

                    # Reduce number of columns
                    rank -= 1

                    # copy the last column here
                    for i in range(0, self.R, 1):
                        Matrix[i][row] = Matrix[i][rank]

                # process this row again
                row -= 1

        # self.Display(Matrix, self.R,self.C)
        return (rank)

# Driver Code
if __name__ == '__main__':
    Matrix = [[10, 20, 10],
              [-20, -30, 10],
              [30, 50, 0]]
    RankMatrix = rankMatrix(Matrix)
    print ("Rank of the matrix is:",
           (RankMatrix.rankOfMatrix(Matrix)))
```

```
Rank of the matrix is: 2
```

[5]:
```
# importing numpy library
import numpy as np

# create numpy 2d-array
m = np.array([[1, 2], [2, 3]])

print("Printing the Original square array:\n", m)

# finding eigenvalues and eigenvectors
w, v = np.linalg.eig(m)

# printing eigen values
print("Printing the Eigen values of the given square array:\n", w)

# printing eigen vectors
print("Printing Right eigenvectors of the given square array:\n",v)
```

```
Printing the Original square array:
 [[1 2]
 [2 3]]
```

```
Printing the Eigen values of the given square array:
 [-0.23606798  4.23606798]
Printing Right eigenvectors of the given square array:
 [[-0.85065081 -0.52573111]
 [ 0.52573111 -0.85065081]]
```

[6]:
```
# importing numpy library
import numpy as np

# create numpy 2d-array
m = np.array([[1, 1, 3],
                        [1, 5, 1],
                        [3, 1, 1]])

print("Printing the Original square array:\n",
        m)

# finding eigenvalues and eigenvectors
w, v = np.linalg.eig(m)

# printing eigen values
print("Printing the Eigen values of the given square array:\n",
        w)

# printing eigen vectors
print("Printing Right eigenvectors of the given square array:\n",
        v)
```

```
Printing the Original square array:
 [[1 1 3]
 [1 5 1]
 [3 1 1]]
Printing the Eigen values of the given square array:
 [-2.  3.  6.]
Printing Right eigenvectors of the given square array:
 [[ 7.07106781e-01 -5.77350269e-01 -4.08248290e-01]
 [-2.21146215e-17  5.77350269e-01 -8.16496581e-01]
 [-7.07106781e-01 -5.77350269e-01 -4.08248290e-01]]
```

[7]:
```
# import the important module in python
import numpy as np

# make a matrix with numpy
gfg = np.matrix('[1, 1, 3; 1, 5, 1; 3, 1, 1]')

# applying matrix.diagonal() method
geeks = gfg.diagonal()
```

```
print(geeks)
```

```
[[1 5 1]]
```

[8]:
```
#Diagonalization and eigen values of matrix using Numpy linalg module  (Method 2)

import numpy as np

A=np.array([[-1,3],[-2,4]])
print ("The matrix is:\n", A)
u,v=np.linalg.eig(A)   #(Eigen value,eigen vector)
#Compute the eigenvalues and right eigenvectors of a square array.
print ('eigen value:',u)
print ('eigen vector:\n',v)
print ('1st eigen vector',v[:,0])      #1st eigen vector
print ('2nd eigen vector',v[:,1])      #2nd eigen vector
dia=np.diag([u[0],u[1]])  #diagonalization of matrix
print ('The diagonal matrix is:\n', dia)
```

```
The matrix is:
 [[-1  3]
 [-2  4]]
eigen value: [1. 2.]
eigen vector:
 [[-0.83205029 -0.70710678]
 [-0.5547002  -0.70710678]]
1st eigen vector [-0.83205029 -0.5547002 ]
2nd eigen vector [-0.70710678 -0.70710678]
The diagonal matrix is:
 [[1. 0.]
 [0. 2.]]
```

[9]:
```
#GRAM schmidt
import numpy as np
A = np.array([[1.0, 1.0, 2.0], [1.0, -1.0, 1.0], [1.0, 2.0, 2.0]])

n = A.shape[1] #number of columns in the matrix. shape[0] give row

for j in range(n):
# To orthogonalize the vector in column j with respect to the previous vectors, subtract from
#it its projection onto each of the previous vectors.
   for k in range(j):
      A[:, j] = A[:, j] - np.dot(A[:, k], A[:, j]) * A[:, k]
   A[:, j] = A[:, j] / np.linalg.norm(A[:, j])
```

```
print(A)
```

```
[[ 0.57735027  0.15430335  0.80178373]
 [ 0.57735027 -0.77151675 -0.26726124]
 [ 0.57735027  0.6172134  -0.53452248]]
```

[10]:
```
import numpy as np

x1, y1, z1, w1 = 1, -2, 3, 9

# Take the input for equation-2
x2, y2, z2, w2 = -1, 3, -1, -6

# Take the input for equation-3
x3, y3, z3, w3 = 2, -5, 5, 17

# Create an array for LHS variables
LHS = np.array([[x1, y1, z1],
                [x2, y2, z2],
                [x3, y3, z3]])

# Create another array for RHS variables
RHS = np.array([w1, w2, w3])

# Apply linear algebra on any numpy
# array created and printing the output
sol = np.linalg.solve(LHS, RHS)
print(sol)
```

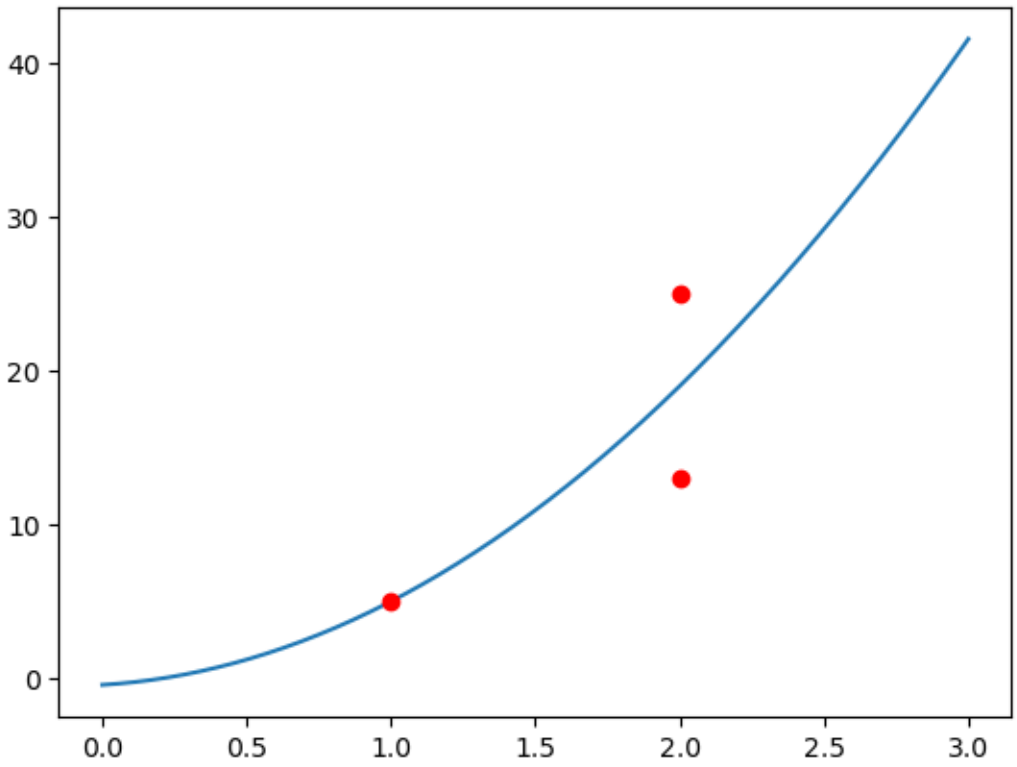
```
[ 1. -1.  2.]
```

[13]:
```
# least square
import numpy as np
from scipy import linalg

A = np.array([[1, 1, 1], [1, 2, 4], [1, 2, 4]])
b = np.array([5, 13, 25]).reshape((3, 1))

p, *_ = linalg.lstsq(A, b)
p
```

[13]:
```
array([[-0.42857143],
       [ 1.14285714],
       [ 4.28571429]])
```

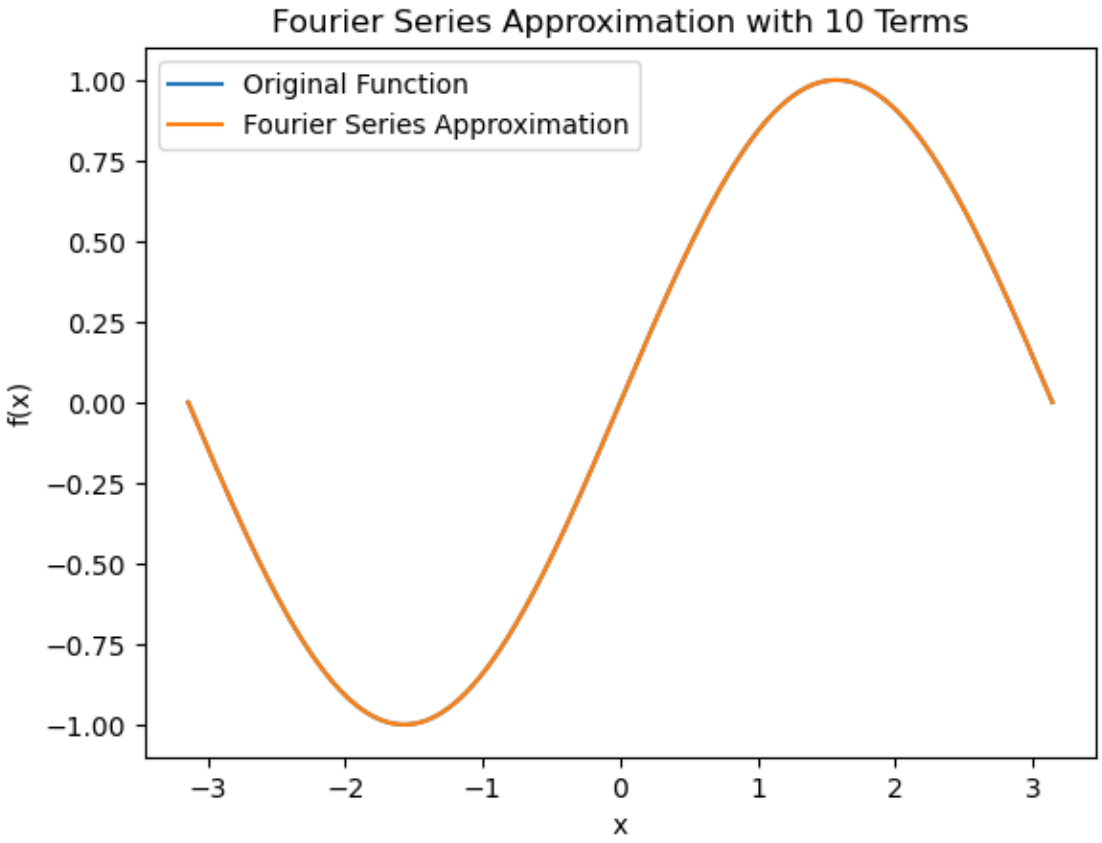
[14]:
```
import matplotlib.pyplot as plt

x = np.linspace(0, 3, 1000)
y = p[0] + p[1] * x + p[2] * x ** 2

plt.plot(x, y)
plt.plot(1, 5, "ro")
plt.plot(2, 13, "ro")
plt.plot(2, 25, "ro")
```

[14]:
```
[<matplotlib.lines.Line2D at 0x21aa3b018e0>]
```



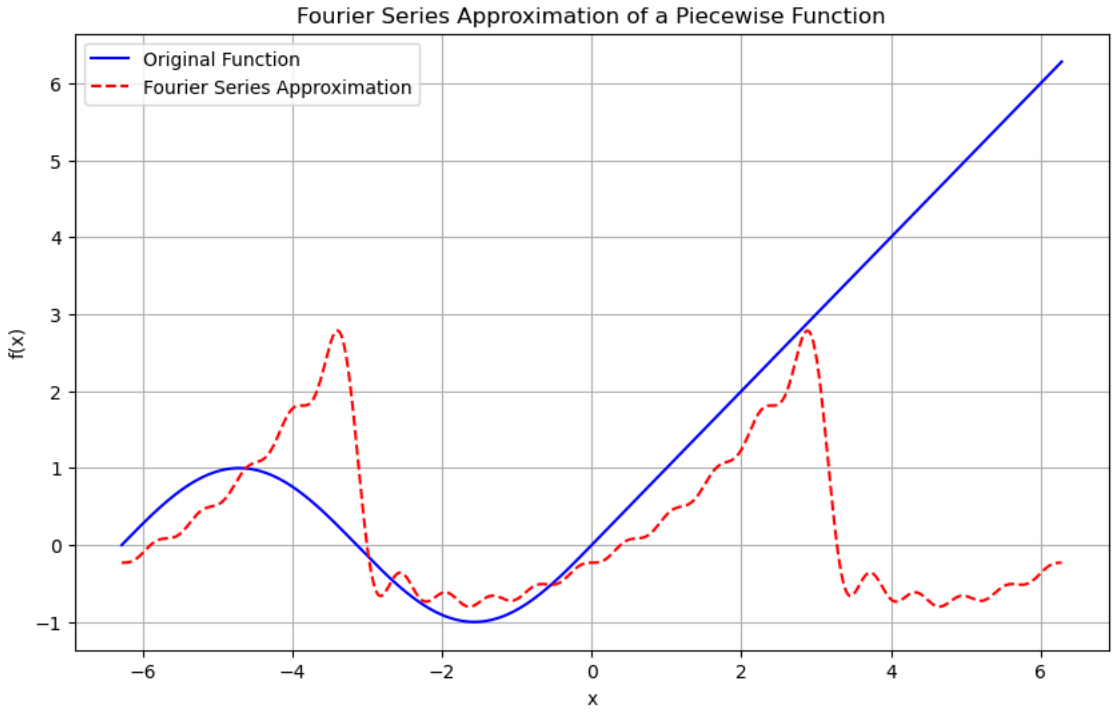
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# Define the function to be approximated
def f(x):
    return np.sin(x)  # Example function

# Function to calculate the a0 coefficient
def a0():
```

```
    result, _ = quad(f, -np.pi, np.pi)
    return result / (2 * np.pi)

# Function to calculate the an coefficient
def an(n):
    integrand = lambda x: f(x) * np.cos(n * x)
    result, _ = quad(integrand, -np.pi, np.pi)
    return result / np.pi

# Function to calculate the bn coefficient
def bn(n):
    integrand = lambda x: f(x) * np.sin(n * x)
    result, _ = quad(integrand, -np.pi, np.pi)
    return result / np.pi

# Function to construct the Fourier series approximation
def fourier_series(x, terms):
    a0_value = a0()
    series = a0_value
    for n in range(1, terms + 1):
        an_value = an(n)
        bn_value = bn(n)
        series += an_value * np.cos(n * x) + bn_value * np.sin(n * x)
    return series

# Number of terms in the Fourier series
terms = 10

# Create an array of x values
x_values = np.linspace(-np.pi, np.pi, 1000)

# Calculate the Fourier series approximation for each x value
fourier_values = np.array([fourier_series(x, terms) for x in x_values])

# Plot the original function and the Fourier series approximation
plt.plot(x_values, f(x_values), label='Original Function')
plt.plot(x_values, fourier_values, label='Fourier Series Approximation')
plt.legend()
plt.title(f'Fourier Series Approximation with {terms} Terms')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```



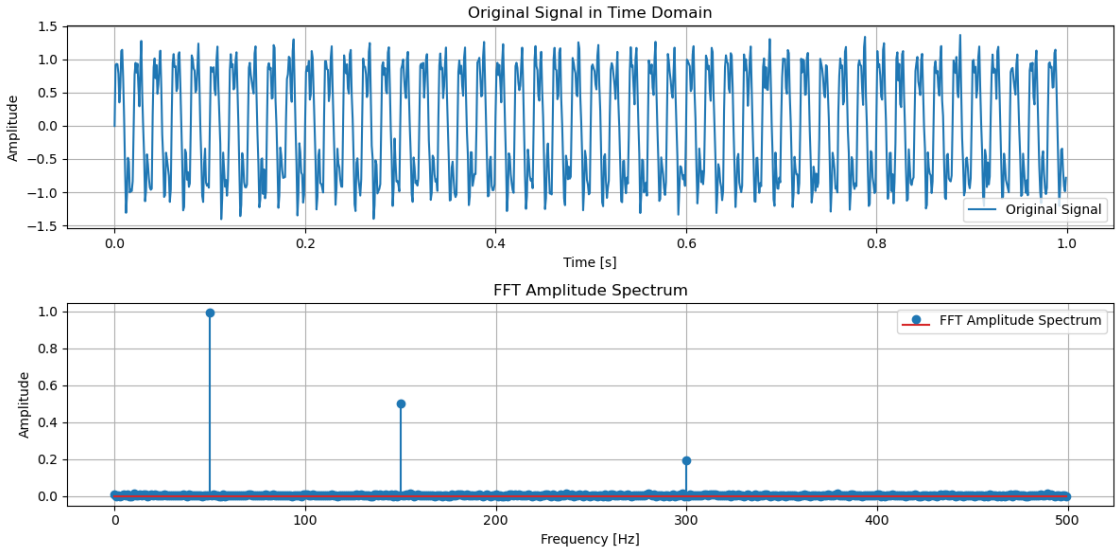
[2]:
```python
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

# Step 1: Define the piecewise function
x = sp.symbols('x')
# Example piecewise function
f = sp.Piecewise((sp.sin(x), x < 0), (x, x >= 0))

# Define the period
T = 2 * np.pi  # Adjust this according to your function's period
L = T / 2

# Step 2: Compute the Fourier coefficients
def fourier_series(f, x, n_terms, T):
    a0 = (1 / T) * sp.integrate(f, (x, -L, L))
    series = a0 / 2
    for n in range(1, n_terms + 1):
        an = (1 / L) * sp.integrate(f * sp.cos(n * sp.pi * x / L), (x, -L, L))
        bn = (1 / L) * sp.integrate(f * sp.sin(n * sp.pi * x / L), (x, -L, L))
```

```
        series += an * sp.cos(n * sp.pi * x / L) + bn * sp.sin(n * sp.pi * x /␣
 ↪L)
    return series

# Number of Fourier terms
n_terms = 10

# Get the Fourier series
fourier = fourier_series(f, x, n_terms, T)

# Step 3: Convert the symbolic Fourier series to a numerical function
fourier_func = sp.lambdify(x, fourier, 'numpy')

# Step 4: Plot the original function and its Fourier series approximation
x_vals = np.linspace(-2 * np.pi, 2 * np.pi, 1000)
f_func = sp.lambdify(x, f, 'numpy')
f_vals = f_func(x_vals)
fourier_vals = fourier_func(x_vals)

plt.figure(figsize=(10, 6))
plt.plot(x_vals, f_vals, label='Original Function', color='blue')
plt.plot(x_vals, fourier_vals, label='Fourier Series Approximation',␣
 ↪color='red', linestyle='--')
plt.legend()
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Fourier Series Approximation of a Piecewise Function')
plt.grid(True)
plt.show()
```



[3]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Generate a signal composed of multiple sine waves
fs = 1000  # Sampling frequency
T = 1.0  # Signal duration in seconds
t = np.linspace(0, T, int(fs * T), endpoint=False)  # Time vector

# Frequencies of the sine waves (in Hz)
freqs = [50, 150, 300]

# Amplitudes of the sine waves
amps = [1, 0.5, 0.2]

# Generate the signal
signal = np.zeros_like(t)
for freq, amp in zip(freqs, amps):
    signal += amp * np.sin(2 * np.pi * freq * t)

# Add some noise to the signal
noise = 0.1 * np.random.randn(len(t))
signal += noise

# Step 2: Apply FFT to the signal
```

```
n = len(t)
yf = np.fft.fft(signal)
xf = np.fft.fftfreq(n, 1 / fs)

# Only keep the positive frequencies
xf = xf[:n // 2]
yf = 2.0 / n * np.abs(yf[:n // 2])

# Step 3: Analyze and visualize the harmonic components
plt.figure(figsize=(12, 6))

# Plot the original signal
plt.subplot(2, 1, 1)
plt.plot(t, signal, label='Original Signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.title('Original Signal in Time Domain')
plt.grid(True)
plt.legend()

# Plot the FFT result
plt.subplot(2, 1, 2)
plt.stem(xf, yf, use_line_collection=True, label='FFT Amplitude Spectrum')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude')
plt.title('FFT Amplitude Spectrum')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```



[ ]: