**CYCLE -2 PART-2**

1. **Create a square matrix with random integer values(use randint()) and use appropriate functions to find:**

**i) inverse**

**ii) rank of matrix**

**iii)** **Determinant**

**iv) transform matrix into 1D array**

**v) eigen values and vectors**

import numpy as np

import numpy as nf

from numpy.linalg import eig

mat = np.random.randint(10, size=(3, 3))

array = nf.random.randint(10, size=(3, 3))

print("Square matrix \n",mat)

M\_inverse = np.linalg.inv(mat)

print("Inverse of the matrix\n",M\_inverse)

rank = np.linalg.matrix\_rank(mat)

print("Rank of the given Matrix \n",rank)

det= np.linalg.det(mat)

print("Determinant of the given Matrix \n",det)

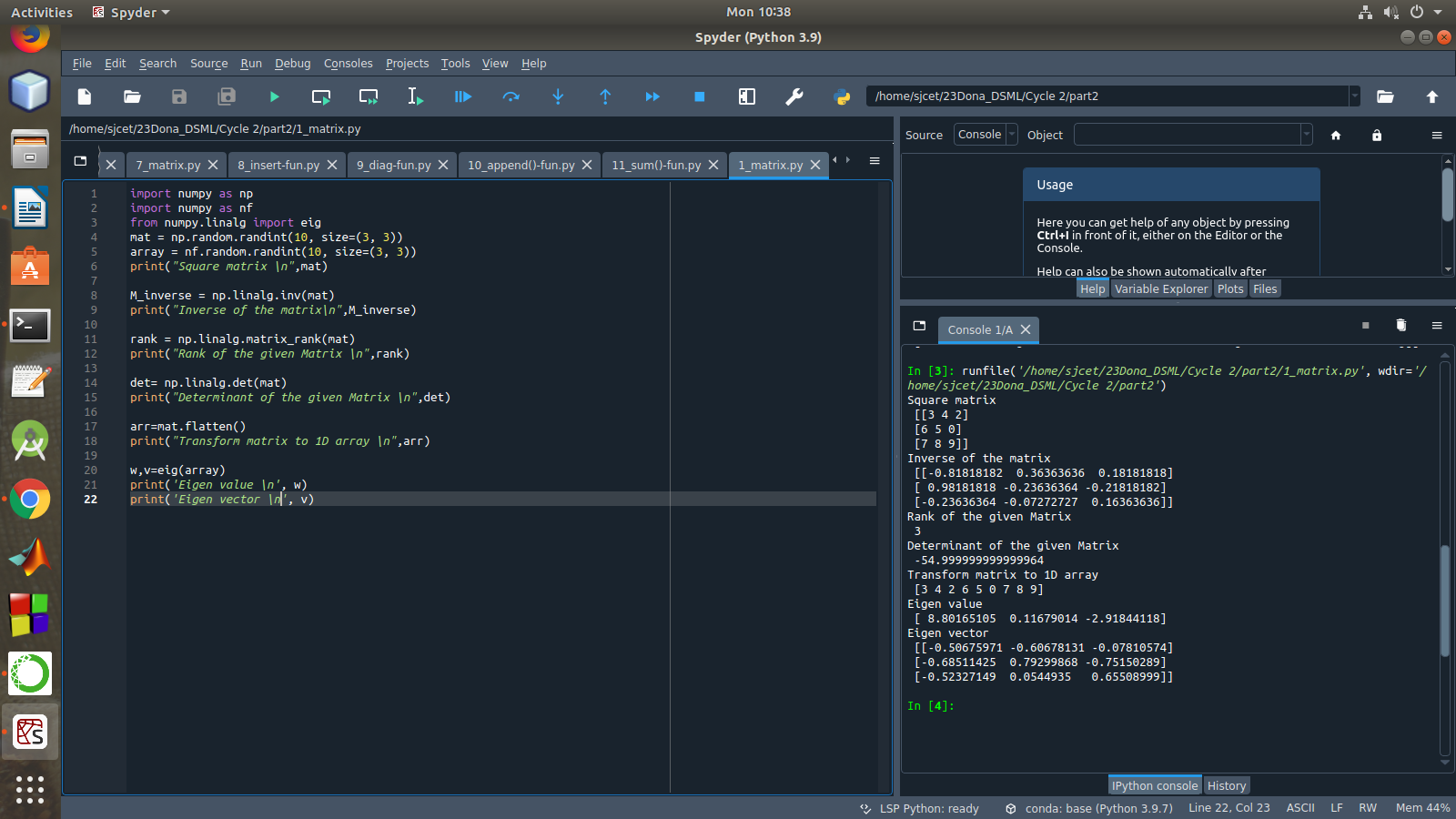
arr=mat.flatten()

print("Transform matrix to 1D array \n",arr)

w,v=eig(array)

print('Eigen value \n', w)

print('Eigen vector \n', v)



1. **Create a matrix X with suitable rows and columns**

**i) Display the cube of each element of the matrix using different methods**

**(use multiply(), \*, power(),\*\*)**

**ii) Display identity matrix of the given square matrix**

**iii) Display each element of the matrix to different powers.**

**iv) Create a matrix Y with same dimension as X and perform the operation X2+2Y**

import numpy as np

mat =np.array([[1, 2, 3],[3,2,4],[2,2,1]])

print("Matrix is....\n",mat)

print("Cubes using \*")

print(mat\*mat\*mat)

print("Cubes using \*\*")

print(mat\*\*3)

print("Cubes using multiply()")

print(np.multiply(mat,(mat\*mat)))

print("Cubes using power()")

print(np.power(mat,3))

print(pow(mat, 3))

b = np.identity(3, dtype = int)

print("Identity matrix:\n", b)

out = np.power(mat, mat)

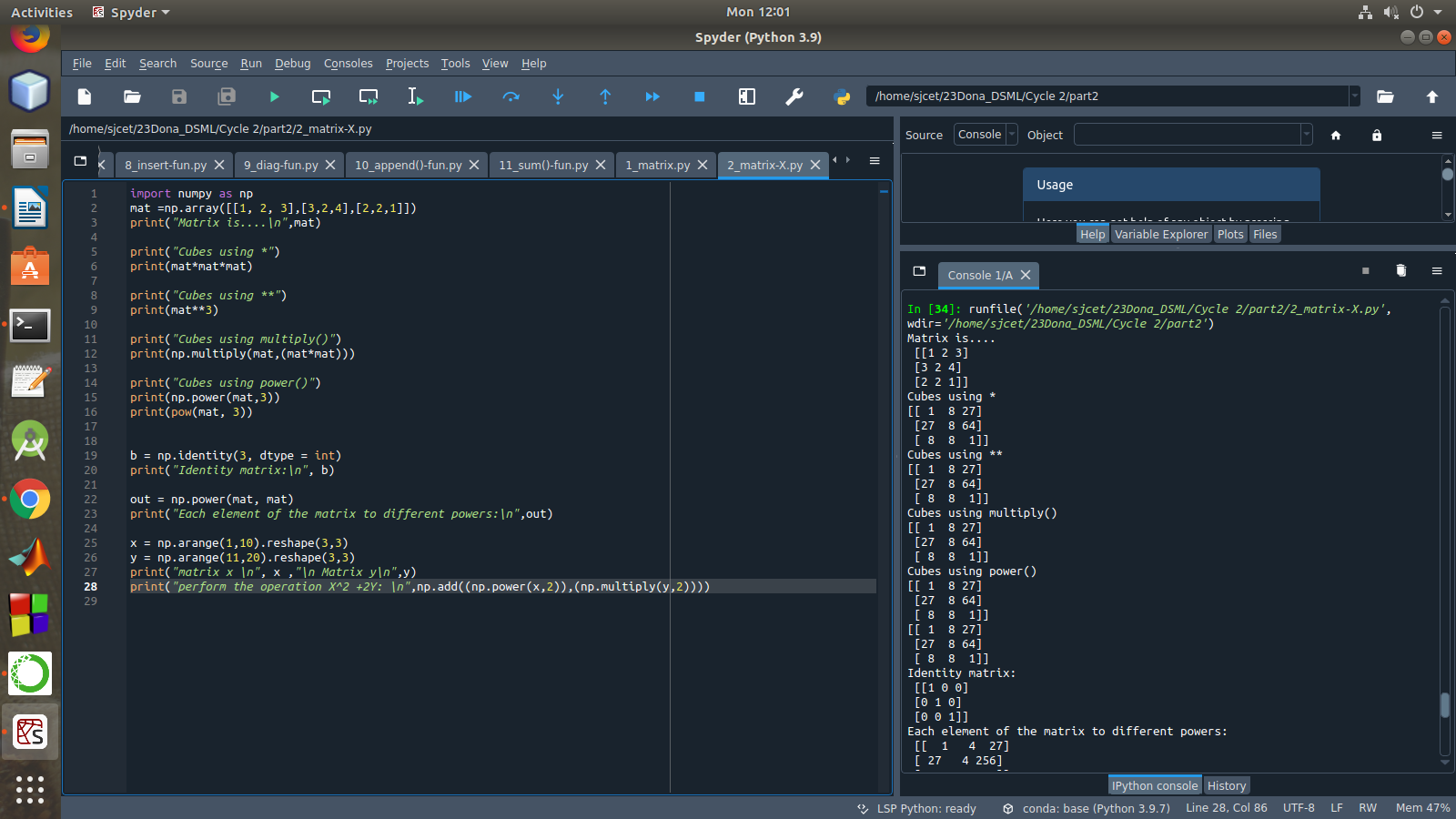
print("Each element of the matrix to different powers:\n",out)

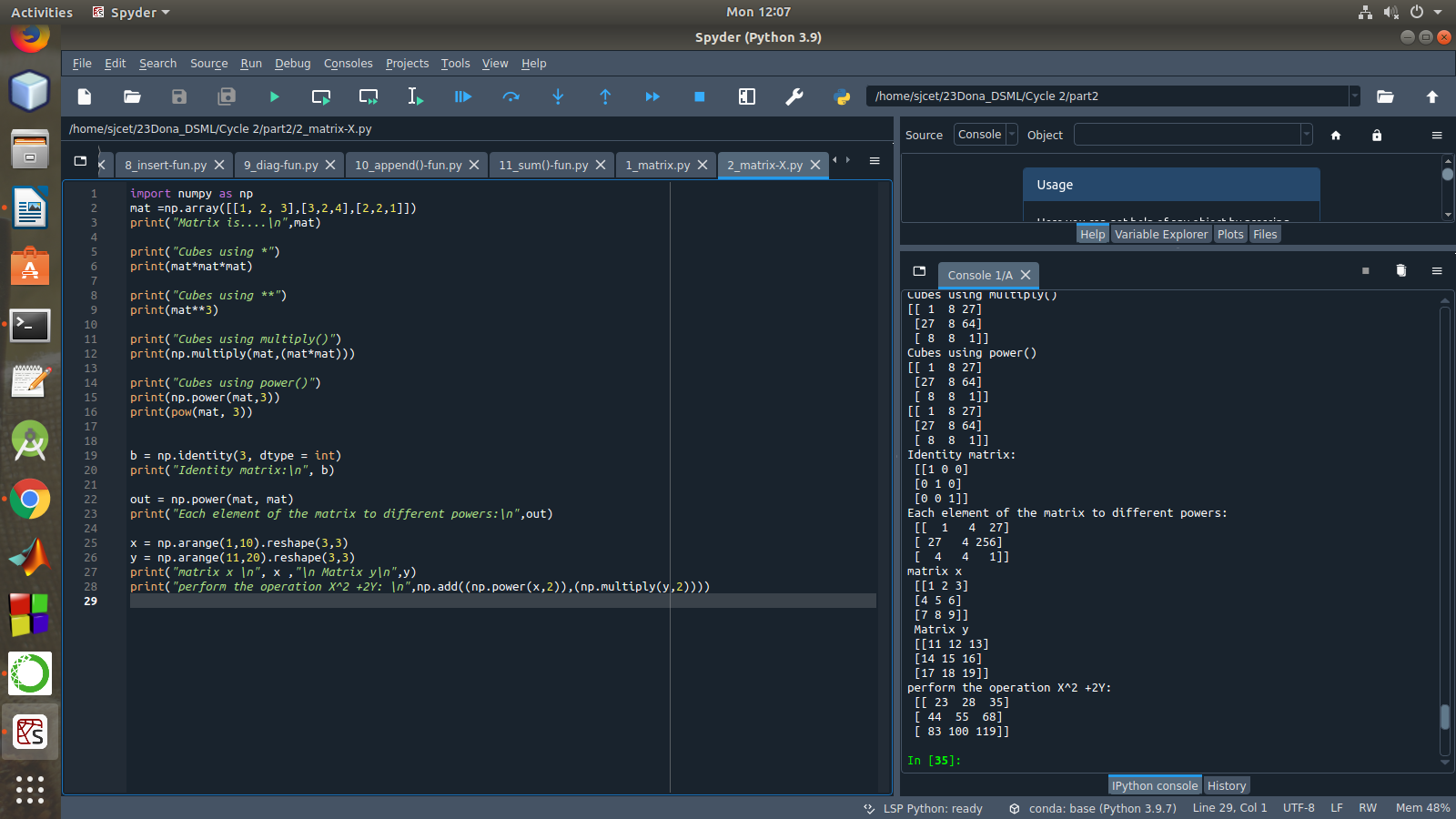
x = np.arange(1,10).reshape(3,3)

y = np.arange(11,20).reshape(3,3)

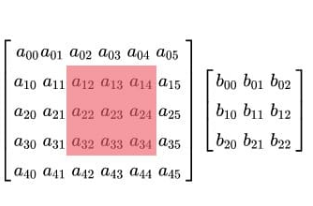
print("matrix x \n", x ,"\n Matrix y\n",y)

print("perform the operation X^2 +2Y: \n",np.add((np.power(x,2)),(np.multiply(y,2))))





1. **Multiply a matrix with a submatrix of another matrix and replace the same in larger matrix.**



import numpy as np

mat = np.array([[6, 1, 1, 4],

[1, 2, 5, 2],

[1, 5, 7, 3],

[3, 2, 4, 1]])

print("Original Matrix....\n",mat)

sub = mat[1:3, 1:3]

print("Sub matrix....\n",sub)

mat2 = np.array([[1, 4],

[3, 2]])

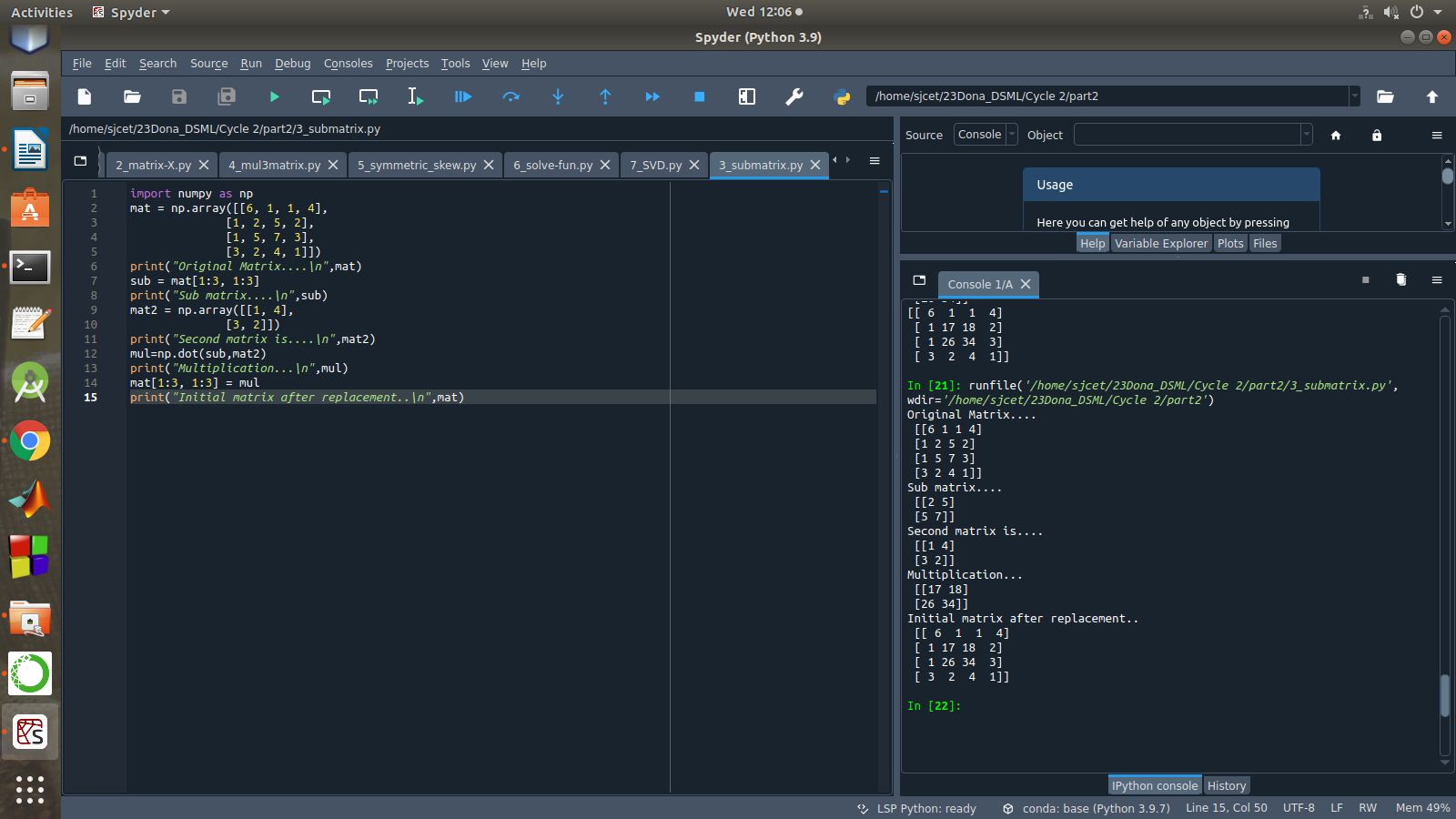
print("Second matrix is....\n",mat2)

mul=np.dot(sub,mat2)

print("Multiplication...\n",mul)

mat[1:3, 1:3] = mul

print("Initial matrix after replacement..\n",mat)



1. **Given 3 Matrices A, B and C. Write a program to perform matrix multiplication of the 3 matrices.**

import numpy as np

M1 = np.array([[3, 6], [4, 2]])

M2 = np.array([[9, 2], [1, 2]])

M3=np.array([[2,4],[3,1]])

Mul = M1.dot(M2)

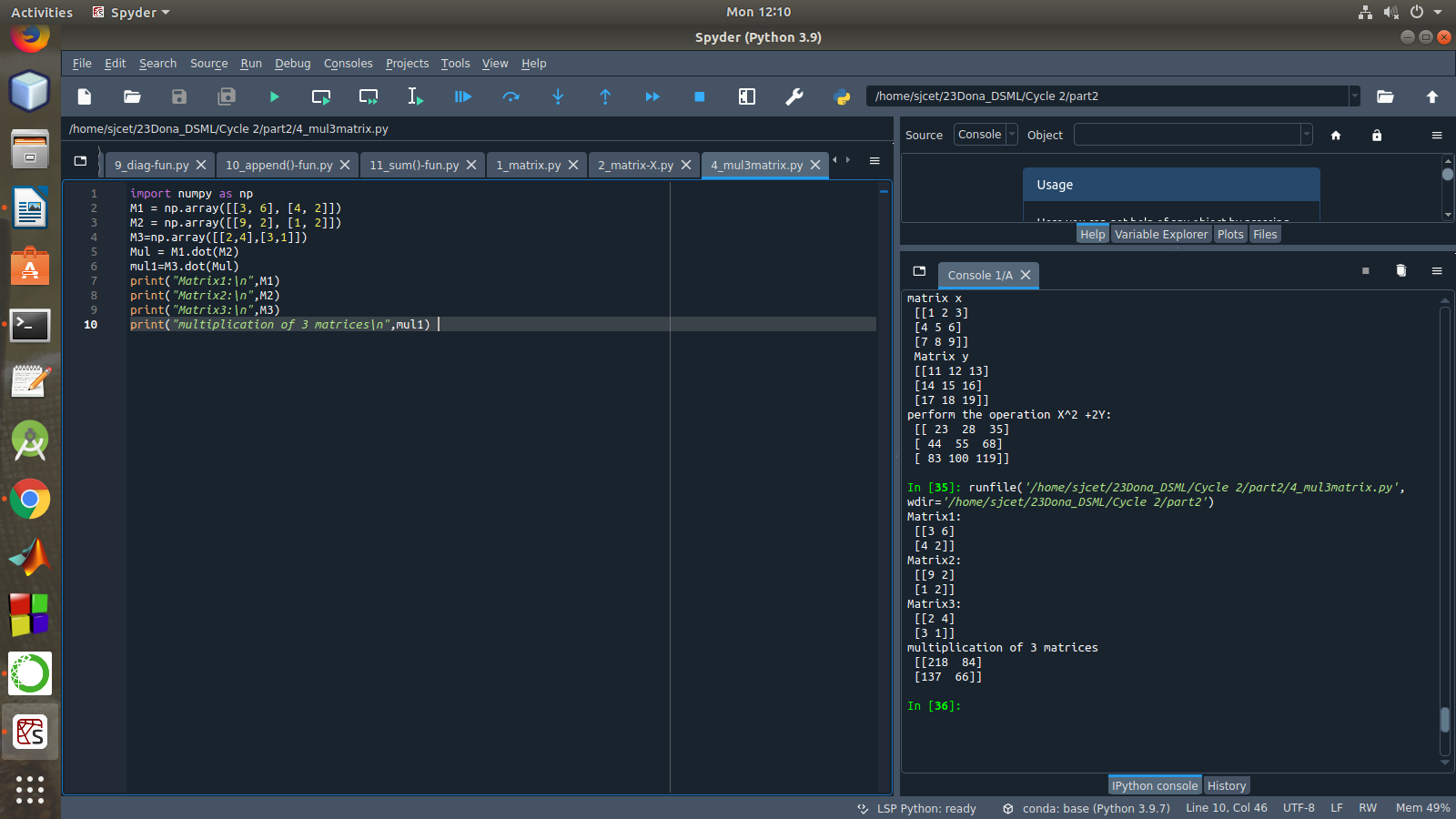
mul1=M3.dot(Mul)

print("Matrix1:\n",M1)

print("Matrix2:\n",M2)

print("Matrix3:\n",M3)

print("multiplication of 3 matrices\n",mul1)



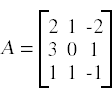
1. **Write a program to check whether given matrix is symmetric or Skew Symmetric.**

Solving systems of equations with numpy

One of the more common problems in linear algebra is solving a matrix-vector equation.

Here is an example. We seek the vector x that solves the equation

A X = b

Where  And X=A-1 b.

**Numpy provides a function called solve for solving such eauations.**

import numpy as np

A = np.array([[6, 1, 1],

[1, -2, 5],

[1, 5, 7]])

print("Original Matrix\n",A)

inv=np.transpose(A)

print ("Transpose matrix\n",inv)

neg=np.negative(A)

comparison = A == inv

comparison1 = inv== neg

equal\_arrays = comparison.all()

skew=comparison1.all()

if equal\_arrays :

print("Symmetric")

else:

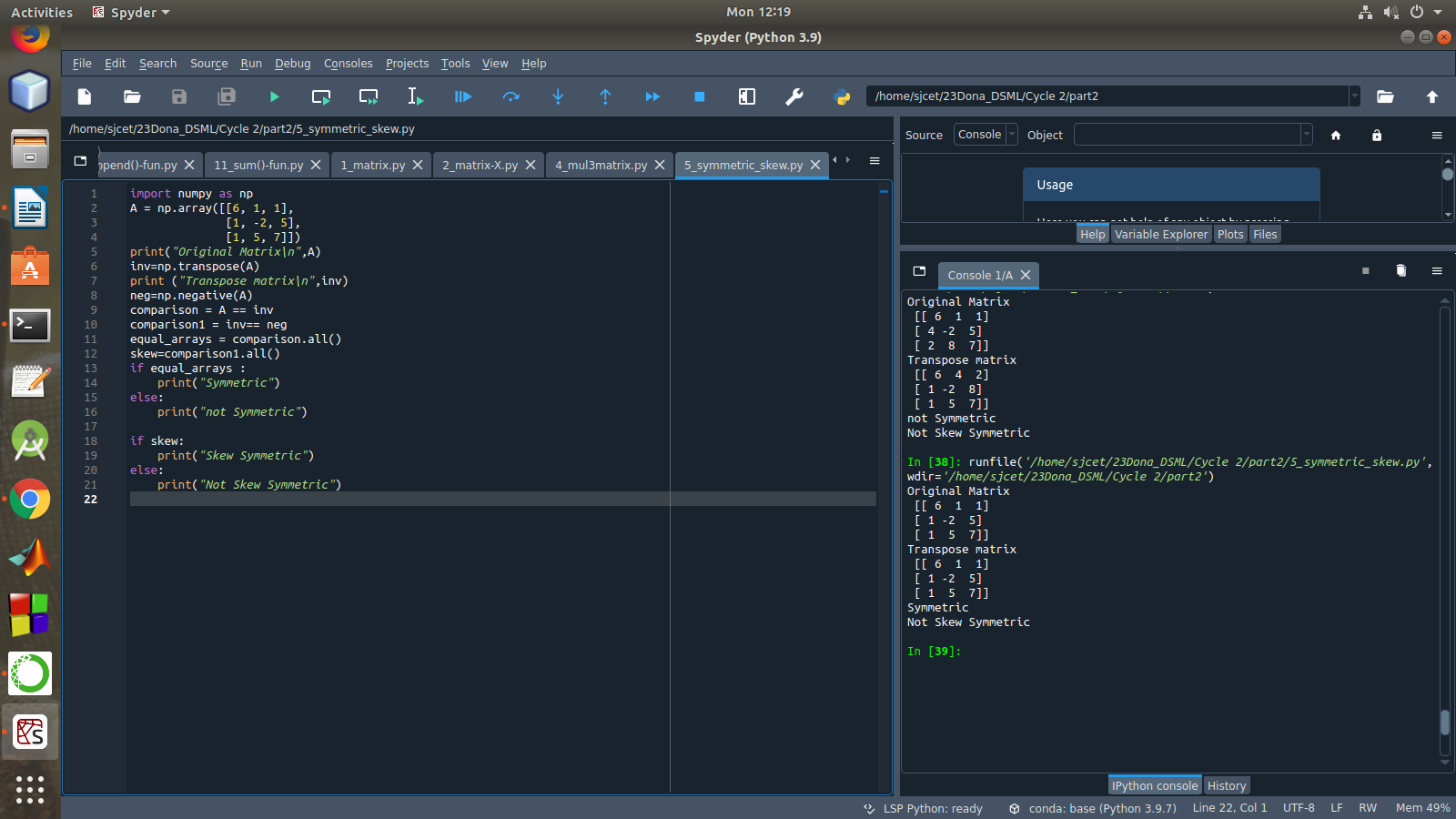
print("not Symmetric")

if skew:

print("Skew Symmetric")

else:

print("Not Skew Symmetric")



1. **Write a program to find out the value of X using solve(), given A and b as above**

import numpy as np

A = np.array([[2, 1, -2],

[3, 0, 1],

[1, 1, -1]])

b=np.array([[3],

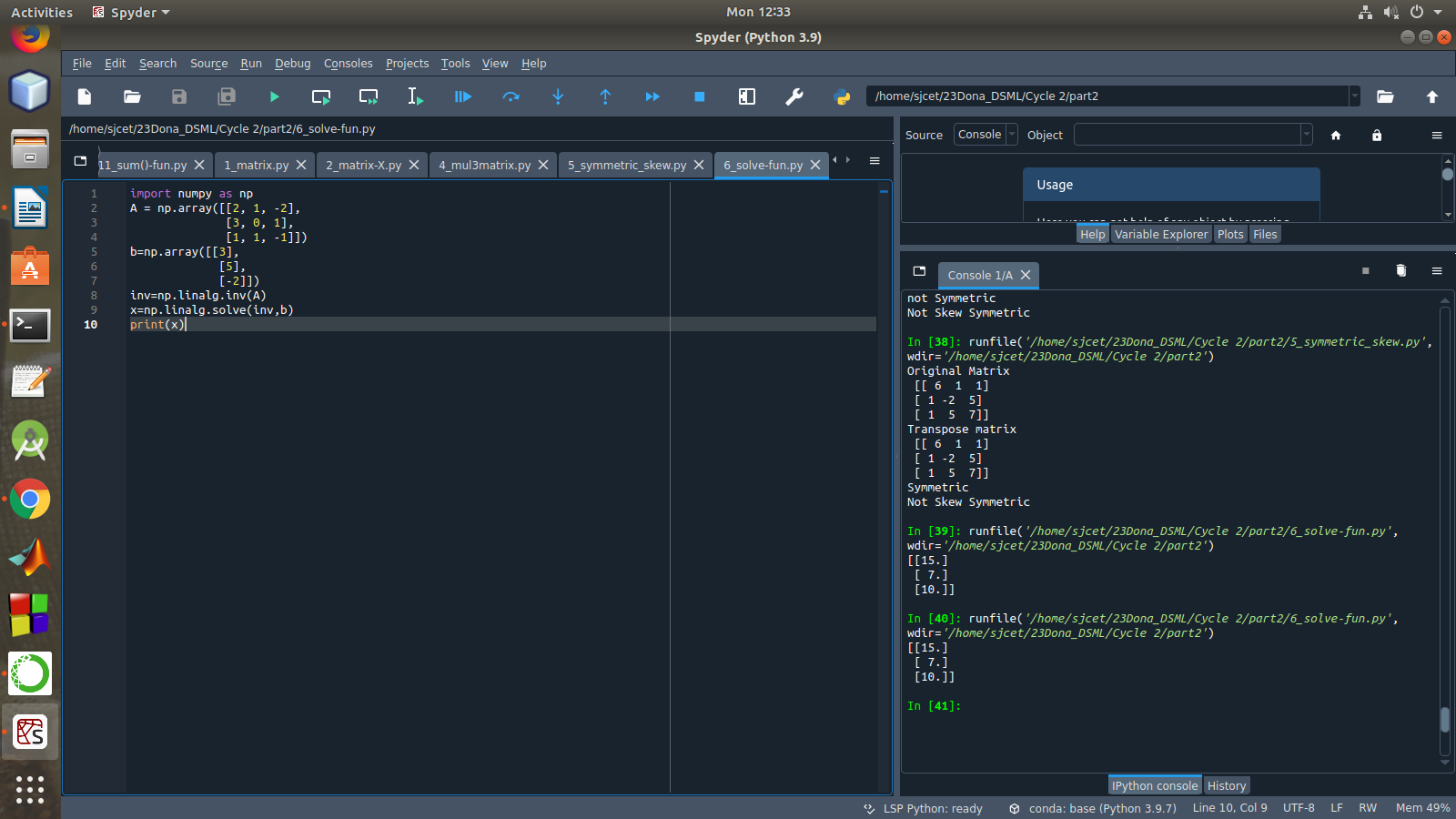
[5],

[-2]])

inv=np.linalg.inv(A)

x=np.linalg.solve(inv,b)

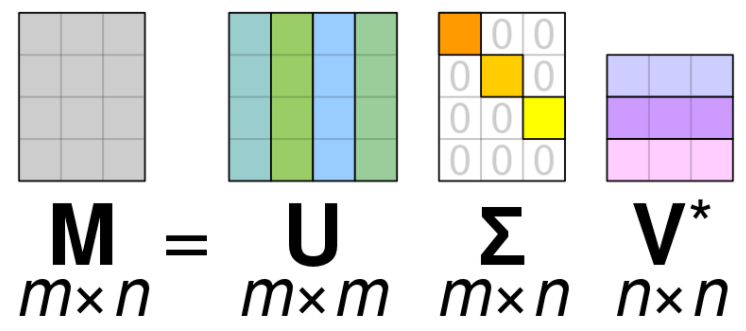
print(x)



Singular value Decomposition

Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. This approach is commonly used in reducing the no: of attributes in the given data set.

**M= U ∑V^T**



* **M**-is original matrix we want to decompose
* **U**-is left singular matrix (columns are left singular vectors). **U** columns contain eigenvectors of matrix **MM**ᵗ
* **Σ**-is a diagonal matrix containing singular (eigen) values.
* **V**-is right singular matrix (columns are right singular vectors). **V** columns contain eigenvectors of matrix **M**ᵗ**M**

**Numpy** provides a function for performing svd, which decomposes the given matrix into 3 matrices.

1. **Write a program to perform the SVD of a given matrix. Also reconstruct the given matrix from the 3 matrices obtained after performing SVD.**

from numpy import array

from scipy.linalg import svd

from numpy import diag

from numpy import dot

from numpy import zeros

# define a matrix

A = array([[1, 2], [3, 4], [5, 6]])

print(A)

# SVD

U, s, VT = svd(A)

print("first" ,U)

print("second",s)

print("3rd" ,VT)

Sigma = zeros((A.shape[0], A.shape[1]))

# populate Sigma with n x n diagonal matrix

Sigma[:A.shape[1], :A.shape[1]] = diag(s)

# reconstruct matrix

B = U.dot(Sigma.dot(VT))

print(B)

