Matrix

* Data Dimensions using Numpy
* Data are classified into three different categories based on their dimension
  + Scalar Values
  + Vectors
  + Matrics
  + Tensors
* Scalar Values
  + Scaler values have no dimensions at all or are often called zero-dimensional data
  + single numerical value
  + 1, 3, 7.6, etc.
* **Vectors:**
* list of scalar values are of two types:
  + Row Vector and Column Vector.
  + The basic difference between these two is they store data in a horizontal and vertical manner respectively.
  + Background pattern

    Description automatically generated
  + Vectors are one dimensional

Length can vary depending on the number of elements they have

* **Matrics**
  + collection of values arranged in rows and columns order. Matrices are two-dimensional data types represented by **mxn**
  + where m is the number of rows and n is the number of columns.

A picture containing shape

Description automatically generated

* + matrix of dimension 2×3 that has 2 rows and 3 columns.
* **Tensors:**
  + n-dimensional values,
  + anything above a two-dimensional matrix is called a tensor.
  + these are hard to visualize so we consider them as the collection of vectors depending on their dimensions.
* Visualize a 3\*3 matrix with indices

Table

Description automatically generated

A picture containing text, scoreboard

Description automatically generated

Sparse Matrices and SciPy,common Scipy methods,samplecodes

**Sparse Matrix and its representations**

A [matrix](https://www.geeksforgeeks.org/data-structures/#Matrix) is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have **0 value**, then it is called a sparse matrix.

**Why to use Sparse Matrix instead of simple matrix ?**

* **Storage:**There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
* **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..

**Example:**

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with **triples- (Row, Column, value).**

Sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation
2. Linked list representation

**Method 1: Using Arrays:**

2D array is used to represent a sparse matrix in which there are three rows named as

* **Row:**Index of row, where non-zero element is located
* **Column:**Index of column, where non-zero element is located
* **Value:**Value of the non zero element located at index – (row,column)

Table

Description automatically generated

# Python program for Sparse Matrix Representation

# using arrays

# assume a sparse matrix of order 4\*5

# let assume another matrix compactMatrix

# now store the value,row,column of arr1 in sparse matrix compactMatrix

sparseMatrix **=** [[0,0,3,0,4],[0,0,5,7,0],[0,0,0,0,0],[0,2,6,0,0]]

# initialize size as 0

size **=** 0

**for** i **in** range(4):

**for** j **in** range(5):

**if** (sparseMatrix[i][j] !**=** 0):

            size **+=** 1

# number of columns in compactMatrix(size) should

# be equal to number of non-zero elements in sparseMatrix

rows, cols **=** (3, size)

compactMatrix **=** [[0 **for** i **in** range(cols)] **for** j **in** range(rows)]

k **=** 0

**for** i **in** range(4):

**for** j **in** range(5):

**if** (sparseMatrix[i][j] !**=** 0):

            compactMatrix[0][k] **=** i

            compactMatrix[1][k] **=** j

            compactMatrix[2][k] **=** sparseMatrix[i][j]

            k **+=** 1

**for** i **in** compactMatrix:

    print(i)

**Output**

0 0 1 1 3 3

2 4 2 3 1 2

3 4 5 7 2 6

Sparce matrix using scipy:

import numpy as np

from scipy.sparse import csr\_matrix

arr = np.array([[0,0,3,0,4],[0,0,5,7,0],[0,0,0,0,0],[0,2,6,0,0]])

print(csr\_matrix(arr).data)

#o/p

[3 4 5 7 2 6]

[3 4 5 7 2 6]

**SciPy**

SciPy is a [Python](https://www.mygreatlearning.com/blog/python-tutorial-for-beginners-a-complete-guide/) library used for scientific computing

SciPy is built in top of the NumPy

SciPy module in Python is a fully-featured version of Linear Algebra while Numpy contains only a few features.

Most new Data Science features are available in Scipy rather than Numpy.

Here is a sampling of the packages included:

Preliminaries

scipy.integrate Numerical integration routines and differential equation solvers

scipy.linalg Linear algebra routines and matrix decompositions extending beyond those pro‐ vided in numpy.linalg

scipy.optimize Function optimizers (minimizers) and root finding algorithms

scipy.signal Signal processing tools scipy.sparse Sparse matrices and sparse linear system solvers

scipy.special Wrapper around SPECFUN, a Fortran library implementing many common mathematical functions, such as the gamma function

scipy.stats Standard continuous and discrete probability distributions (density functions, samplers, continuous distribution functions), various statistical tests, and more descriptive statistics Together NumPy and SciPy form a reasonably complete and mature computational foundation for many traditional scientific computing applications.

**scipy.integrate:**

Methods for Integrating Functions given function object.

   quad          -- General purpose integration.

   dblquad       -- General purpose double integration.

   tplquad       -- General purpose triple integration.

   fixed\_quad    -- Integrate func(x) using Gaussian quadrature of order n.

   quadrature    -- Integrate with given tolerance using Gaussian quadrature.

   romberg       -- Integrate func using Romberg integration.

 Methods for Integrating Functions given fixed samples.

   trapz         -- Use trapezoidal rule to compute integral from samples.

   cumtrapz      -- Use trapezoidal rule to cumulatively compute integral.

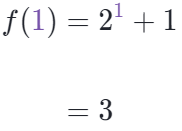
   simps         -- Use Simpson's rule to compute integral from samples.

   romb          -- Use Romberg Integration to compute integral from

                    (2\*\*k + 1) evenly-spaced samples.



if we want to find the y-value when x=1x, equals, 1, we can evaluate f(1),

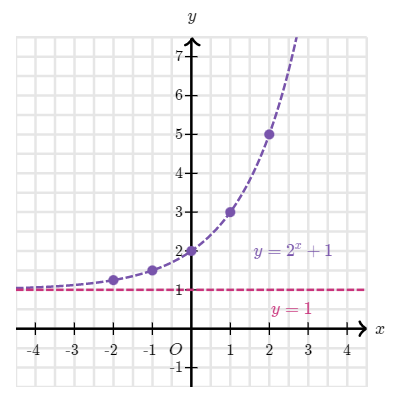


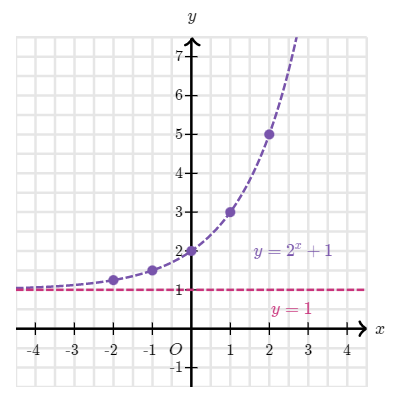
On the graph:

x=1, y=3



Finding the value of x between -4 to 4







Text

Description automatically generated



Find the area between limits 0 to 1

#single intergration

from scipy.integrate import quad

# function we want to integrate

def f(x):

return 2 \*\* x+1

res= quad(f, 1,2 ) #here 1 is lower limit and 2 is upper limit

print(res)

Res gives res and err as output:

(2.4426950408889634, 2.7119362765326152e-14)

#**Optimization** seeks to find the best (optimal) value of some function subject to constraints

**import** **numpy** **as** **np**

**import** **scipy.linalg** **as** **la**

**import** **matplotlib.pyplot** **as** **plt**

**import** **scipy.optimize** **as** **opt**

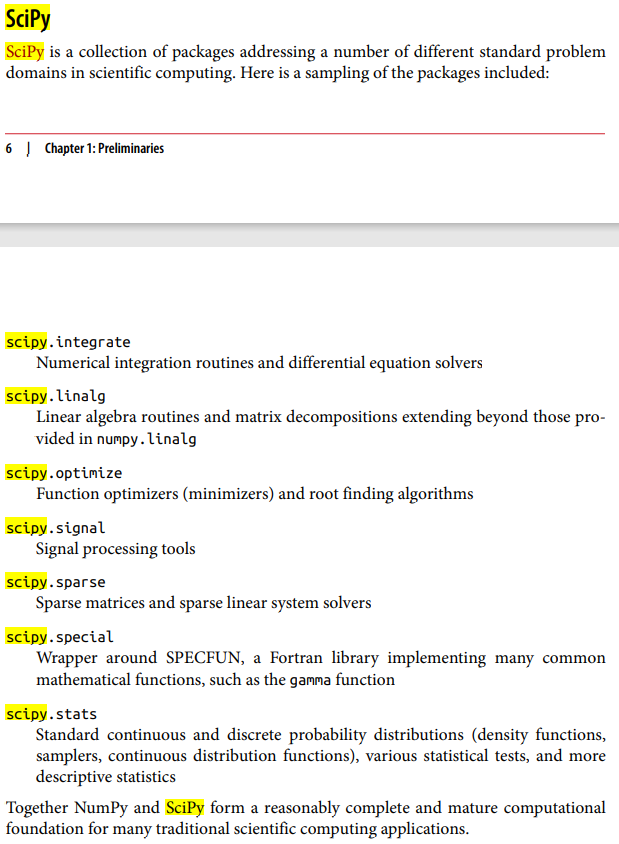
## **Functions of One Variable**

An easy example is to minimze a quadratic

f = **lambda** x: x\*\*2 + 1 *# a x^T x + 0\*x + 1*

x = np.linspace(-2,2)

print(x,f(x))



**Scipy.linalg**

**Text

Description automatically generated with medium confidence**

**import numpy as np**

**A = np.array([[8, 3, -2], [-4, 7, 5], [3, 4, -12]])**

**b = np.array([9, 15, 35])**

**x = np.linalg.solve(A, b)**

**print(x)**

**#[-0.58226371 3.22870478 -1.98599767]**

**op=8\*x[0] + 3\*x[1] - 2\*x[2]**

**print(op)**

**#9.0**

User-Defined Moving Window Functions The apply method on rolling and related methods provides a means to apply an array function of your own devising over a moving window. The only requirement is that the function produce a single value (a reduction) from each piece of the array. For example, while we can compute sample quantiles using rolling(...).quan tile(q), we might be interested in the percentile rank of a particular value over the sample. The scipy.stats.percentileofscore function does just this (see Figure 11-10 for the resulting plot): In [265]: from scipy.stats import percentileofscore In [266]: score\_at\_2percent = lambda x: percentileofscore(x, 0.02) In [267]: result = returns.AAPL.rolling(250).apply(score\_at\_2percent) In [268]: result.plot()Graphical user interface, chart, line chart

Description automatically generated

Text

Description automatically generated