STTP on Python Programming for Students, Engineers & Researchers

Samridh Khaneja
khanejasamridh@gmail.com

Project engineer, Wipro technologies
M.Sc. Mathematics
Sardar Vallabhbhai National Institute of Technology, Surat

$27^{th} December, 2022$

Outline

- 1-D Arrays
- 2-D Arrays(basics of matrices)
- Matplotlib
- Numerical linear algebra
- Scipy

1-D Arrays

- Create a numpy array of even integers from 20 to 40 using various functions and store it as ar1
    - '*array*' function.
    - '*arange*' function.
    - '*linspace*' function.
- Set the second element(index $= 1$) of ar1 to 0 and print the array.
- Print the data type of the elements of ar1 using the '*dtype*' command.
- Use the '*any*' function to check if any of the elements in ar1 are zero or not.
- Print the length and dimension of ar1 using built-in functions.

1-D Arrays

- Statistics using numpy:
  - Print the largest and smallest value from ar1 using built-in functions.
  - Print mean and median of ar1 using '*mean*' and '*median*' functions respectively.
  - Print standard deviation and variance of ar1 using built-in functions.
  - Print the sum and product of all elements of ar1 using the built-in functions.
- Other operations:
  - Sort ar1 in ascending order using the '*sort*' function.
  - Reverse and print the ar1 using the '*flip*' function.
  - Count the number of non-zero elements in ar1 using '*count_nonzero*' function.

1-D Arrays

- Vectorization
  - Create and store a vector F = [32, 38, 40, 28, 56, 65, 70] which contains temperatures(Fahrenheit). Print F.
  - Use the formula $\frac{C}{5} = \frac{F-32}{9}$ to create an array C of temperatures(Centigrade) corresponding to F. Print C.
  - Square every element of F and print the array.
  - Find *Sin*, *Cos* and *Tan* of each element in F. Print the results.

1-D Arrays

- Operations on a vector using relational and logical operators:
  - Print elements of array F which are greater than 50.
  - Print elements of array F which are divisible by both 8 and 4.
  - Print elements of array F which are divisible by 8 or 4.
  - Print elements of array F which are not divisible by 4.

Take home exercises

- Use loops to complete vectorization problems. Compute the timings in both the cases. You should observe that, vectorization method is faster as compared to use of loops.
- Take a 3 X 3 X 3 array and do various manipulations on them.

2-D Arrays(basics of matrices)

Create the following matrices:

$$\begin{bmatrix} 5 & 0 & 4 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 3 & 1 & 1 \end{bmatrix} \; and \; \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \; using$$

- '*matrix*' function in numpy. Store these matrices as A_1, B_1 and C_1 respectively. Print these matrices.
- '*array*' function in numpy. Store these matrices as A, B and C respectively. Print these matrices.
- Print A * B and $A_1 * B_1$. Did you notice the difference in outputs ?

2-D Arrays(basics of matrices)

- Print the dimension of A, B and C using the '*ndim*' attribute.
- Print the number of rows and columns in A, B and C using the '*shape*' attribute.
- Print the transpose of A, B and C using the '*transpose*' function.
- Print the diagonal of A and B using the '*diagonal*' function.
- Print $A + B$ and $A - B$.
- Create an identity matrix of order 3 X 3 and verify that AI = A. Use '*eye*' function.
- Print the matrix products AB, BA, AC and BC.
- Verify that matrix products are computed using '*np.dot*' function.

2-D Arrays(basics of matrices)

- Print greatest and least elements of A, B and C using '*min*' and '*max*' functions respectively.
- Print the sum of all elements in matrices A, B and C using the '*sum*' function.
- Print the traces of matrices A, B and C using the '*trace*' function.
- Print the flattened one-dimensional array for A, B and C using the '*flatten*' function.
- Print the sum of rows and sum of columns for all matrices A, B and C using the '*sum*' function. [Hint: Use axis = 0 and axis = 1]

Take home exercises

- Read about different matrix products and implement them using numpy functions.
- Compute inner product and outer product of matrices using numpy functions.
- Implement all the operations using loops and compare execution time in both the cases. Library functions execute a little faster.

Matplotlib

- Generate 100 evenly spaced points between 0 and $\frac{\pi}{4}$. Store them in a 1-D array x.
- Compute 1-D arrays:
  - $y_1 = Sin(x)$
  - $y_2 = Cos(x)$
  - $y_3 = Tan(x)$
- Plot the above functions on separate plots.

Matplotlib

- In each of the plots:
    - Add an appropriate x-label.
    - Add an appropriate y-label.
    - Add an appropriate label to the plot.
    - Display a grid.
    - Set x limit from 0 to $\frac{\pi}{4}$.
    - Set y limit from 0 to 1.1.

Matplotlib

- Plot all the three functions on the same plot using:
  - Red color and a solid line for Sin(x).
  - Blue color and a dotted line for Cos(x).
  - Green color and a dashed line for Tan(x).
  - Add an appropriate label, x-label, y-label and set limits for the graph.

Matplotlib(Scatter and Bar plots)

Consider the annual revenues of the four big companies(Amazon, Meta, Microsoft and Apple) for the years 2018-2021.

- Create several lists that include the names of these companies with yearly revenues(in billion dollars) such as:
  - company = [‘amazon’, ‘meta’, ‘microsoft’, ‘apple’]
  - revenue_2018 = [233, 56, 110, 265]
  - revenue_2019 = [280, 71, 123, 260]
  - revenue_2020 = [386, 86, 143, 274]
  - revenue_2021 = [471, 118, 168, 366]

Matplotlib(Scatter and Bar plots)

- Draw a scatter plot using the '*scatter*' function in matplotlib and color the revenue points.
- Draw a bar plot for each year separately.
- Draw four sub-plots corresponding to revenue for each year in one figure.
- Create a stacked bar chart for each company.

Take home exercises

- Take a relevant problem that you have encountered as an academic or industry professional and perform an exploratory data analysis using matplotlib.
- Lear more about creating informative histograms, contour plots, etc in matplotlib.
- Learn more about the Seaborn library for visualization in Python.

Numerical linear algebra

Consider:

$$\begin{bmatrix} 5 & 0 & 4 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \textit{ using}$$

- Using the '*array*' function in numpy, create and print the coefficient matrix A and the right hand side vector b.
- Print the determinant of A using the '*linalg.det*' function. If the determinant is nonzero hence a unique solution for the above equation exists.
- Print and store the solution x_1 found using the '*linalg.solve*' function. Verify using $np.dot(A, x_1)$ to confirm if x_1 is a solution.

.

Numerical linear algebra

- We can also compute the solution by computing the inverse of A.
  1. Print the inverse of A computed using '*linalg.inv*' function.
  2. Using the 'np.dot()' function multiply inverse of A and b to get the solution x_2. Verify using $np.dot(A, x_2)$ to confirm if x_2 is a solution.
- We can also compute the solution by computing the QR factorization of A.

if A = QR where Q is orthogonal $Q^T Q = T$ and R is upper triangular

$$Ax = b \tag{1}$$

$$QRx = B \tag{2}$$

$$Rx = Q^T b \tag{3}$$

x can be obtained by solving the system (3)

.

Numerical linear algebra

1. Using '*linalg.qr*' function compute the QR factorization of A.
2. Store $b_1 = Q^T b$ using the '*transpose*' function and then multiply Q^T and b using the np.dot function.
3. Use '*linalg.solve*' on R and b_1 to get the solution x_3.
4. Verify the solution.

- In this exercise we will see one of the reasons why it is a bad practice to compute the inverse of a matrix in general. Consider:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Numerical linear algebra

- Store the above matrix in a python.
- Compute the inverse of this matrix using '*linalg.inv*'.
- Print the inverse matrix. (The original matrix has so many zero entries(sparse) while the inverse has none. It implies that a sparse matrix can be stored with less memory as compared to its inverse.)

.

Numerical linear algebra

- Find the pseudo inverse for $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$. This can be done using '*linalg.pinv*' function.
- Print the 1,2 Frobenius norm of matrix A using an built-in function.
- Print the eigen values and eigen vectors of A using built-in function.
- Compute the singular value decomposition of A using built-in function.

Scipy

Stats sub-package(import stats)

- Create a list l1 of any 20 integers. Print the list.
- Print the general statistics of l1 using the '*describe*' function.
- Print the geometric and harmonic means of l1 using the '*gmean*' and '*hmean*' functions respectively.
- Print the inter quartile range of l1 using the '*iqr*' function.
- Print the standard error of mean of l1 using '*sem*' function.
- Print the cumulative frequency of l1 data using the '*cumfreq*' function.

Scipy

Random number generation, probability density function, cumulative density function and quartiles

- Print the cdf of a standard normal random variable X at x = -2, 1 and 1.5 using the function '*stats.norm.cdf*'.
- Print the pdf at the same points using the function '*stats.norm.pdf*'.
- Print the median using the quartile function '*stats.norm.ppf*'. [Hint: Use argument = [0.5]]
- Draw and print 10 random samples from a standard normal distribution using the function '*stats.norm.rvs*' with the size argument.

Take home exercises

- Read about using the optimization, interpolation and image packages in Scipy. Implement them on practical problems.