#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Nov 28 11:51:22 2019

@author: hym0509

"""

一 numpy基础

1，数组对象

1）数组的下标从0开始，最后一个元素的下标上数组长度减1.

2）numpy.arange(start,end,length)

3)numpy.array([element1,element2,..])

import os

import sys

import datetime as dt

import numpy as np

vec1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]

vec2 = np.array(vec1)

vec1 + vec1

vec2 + vec2

for i in range(len(vec1)):

vec1[i] += vec1[i]

vec1

1 Matrix multiplication

mat1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

mat2 = np.array(mat1)

np.dot(mat2, mat2)

mat3 = np.zeros([3, 3])

for i in range(3):

for k in range(3):

for j in range(3):

mat3[i][k] = mat3[i][k] + mat1[i][j] \* mat1[j][k]

mat3

2 Creating NumPy arrays

2.1 np.array(list): Converts python list into NumPy arrays.

2.2 array.ndim: Returns Dimension of the array.

2.3 array.shape: Returns shape of the array as a list.

arr1 = [4, 8, 2]

arr1 = np.array(arr1)

arr2 = np.array([24.3, 0., 8.9, 4.4, 1.65, 45])

arr3 = np.array([[4, 8, 5], [9, 3, 4], [1, 0, 6]])

arr1.ndim

arr3.shape

3，Array creation functions

3.1 Creation functions

3.1.1 np.arange(start, stop, step): Creates vector of values from start to stop with step width step.

np.arange(6)

3.1.2 np.zeros((rows, columns)): Creates array with all values set to 0.

np.zeros((4, 3))

3.1.3 np.identity(n): Creates identity matrix of dimension n.

np.identity(3)

3.1.4 np.linspace(start, stop, n): Creates vector of n evenly divided values from start to stop.

np.linspace(0, 80, 5)

3.1.5np.full((row, column), k): Creates array with all values set to k.

np.full((5, 4), 7)

3.2 Array of random numbers

3.2.1 np.random.rand(rows, columns): Creates array of random floats between zero and one.

np.random.rand(3, 3)

3.2.2 np.rondom.randint(k, size=(rows, columns)): Creates array of random integers between 0 and k-1.

np.random.randint(10, size=(5, 4))

3.3 Copy arrays

3.3.1 Reference

arr3

arr = arr3

arr[1, 1] = 777

arr3

3.3.2 array.copy(): Copies an array without reference (call-by-value).

arr3

arr = arr3.copy()

arr[1, 1] = 777

arr3

3.4 Array operations

3.4.1 Element-wise operations

arr3

arr3 + arr3

arr3\*\*2

3.4.2 Matrix multiplication

arr3 \* arr3

arr = np.ones((3, 2))

arr

4 Indexing with an integer：array[index]: Selects the value at position index from the data.

arr = np.arange(10)

arr

arr[4]

arr[-1]

5 Slicing：array[start : stop : step]: Selects a subset of the data.

5.1 Slicing in one dimension

arr = np.arange(10)

arr

arr[3:7]

arr[1:]

arr[:7]

arr[-3:]

arr[::-1]

arr[::2]

arr[:5:-1]

5.2 Slicing in higher dimensions

5.2.1 Indexing rows

arr3

vec = arr3[1]

vec

arr3[-1]

5.2.2 Slicing in two dimensions

arr3

arr3[0:2, 0:2]

arr3[2:, :]

6 Views on arrays

6.1 Creating views implicitly

column = arr3[:, 1]

column

column.base

column[1] = 100

arr3

6.2 Create a view by slicing

elem = column[1:2]

elem.base

elem[0] = 3

arr3

6.3 Obtaining views explicitly

arr3\_t = arr3.T

arr3\_t

arr3\_r = arr3.reshape(1, 9)

arr3\_r

7 Fancy indexing

7.1 Advanced and basic indexing

arr3

arr = arr3[[0, 2], [0, 2]]

arr

arr.base

arr = arr3[0:3:2, 0:3:2]

arr

arr.base

7.2 Boolean indexing

7.2.1 Boolean arrays

bool\_arr = (arr3 < 5)

bool\_arr

bool\_arr1 = (arr3 == 0)

bool\_arr1

7.2.2 Indexing with boolean arrays

arr3

y = arr3 % 2 == 0

y

arr3[y]

8 Conditional indexing

8.1 Find and replace values in arrays

a, b = arr3.copy(), arr3.copy()

for i in range(a.shape[0]):

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

if a[i, j] % 2 == 0:

a[i, j] = 5

b[b % 2 == 0] = 5

b

8.2 Find and replace values in arrays, condition: equal

arr3

arr = arr3.copy()

arr[arr == 4] = 100

arr

9 Reshape

9.1 array.reshape((rows, columns)): Reshapes an existing array.

arr = np.arange(15)

arr.reshape((3, 5))

9.2 array.resize((rows, columns)): Changes array shape to rows x columns and fills new values with 0.

arr = np.arange(15)

arr.resize((3, 7))

arr

10 Adding and removing elements of arrays

np.append(array, value): Appends value to the end of array.

np.insert(array, index, value): Inserts values before index.

np.delete(array, index, axis): Deletes row or column on index.

a = np.arange(5)

a = np.append(a, 8)

a = np.insert(a, 3, 77)

print(a)

a.resize((3, 3))

np.delete(a, 1, axis=0)

11 Combining and splitting

np.concatenate((arr1, arr2), axis): Joins a sequence of arrays along an existing axis.

np.split(array, n): Splits an array into multiple sub-arrays.

np.hsplit(array, n): Splits an array into multiple sub-arrays hori- zontally.

np.concatenate((a, np.arange(6).reshape(2, 3)), axis=0)

np.split(np.arange(8), 4)

12 Transposing array:array.T

arr3

arr3.T

13 Matrix multiplication:np.dot(arr1, arr2)

res = np.dot(arr3, np.arange(18).reshape((3, 6)))

res

res2 = arr3 @ np.arange(18).reshape((3, 6))

res2

14 Array functions

arr3

np.sqrt(arr3)

np.exp(arr3)

15 Binary functions

x = np.array([3, -6, 8, 4, 3, 5])

y = np.array([3, 5, 7, 3, 5, 9])

np.maximum(x, y)

np.greater\_equal(x, y)

np.add(x, y)

np.mod(x, y)

16 Data processing:np.meshgrid(array1, array2)

p = np.arange(-5, 5, 0.01)

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

x

import matplotlib.pyplot as plt

val = np.sqrt(x\*\*2 + y\*\*2)

plt.figure(figsize=(2, 2))

plt.imshow(val, cmap="hot")

plt.colorbar()

17 Conditional logic:np.where(condition, a, b)

a = np.array([4, 7, 5, -7, 9, 0])

b = np.array([-1, 9, 8, 3, 3, 3])

cond = np.array([True, True, False, True, False, False])

res = np.where(cond, a, b)

res

res = np.where(a <= b, b, a)

res

18 Sorting

18.1 Sorting one-dimensional arrays

arr2

arr2.sort()

arr2

18.2 Sorting two-dimensional arrays

arr3

arr3.sort()

arr3

arr3.sort(axis=0)

arr3

##exapmle

def python\_sum(n):

a=[i\*\*2 for i in range(n)]

b=[i\*\*3 for i in range(n)]

c=[]

for i in range(n):

c.append(a[i]+b[i])

return c

def numpy\_sum(n):

np.arange(n)\*\*2 +np.arange(n)\*\*3

def main(argc,argv,envp):

n=100000

start=dt.datetime.now()

c=python\_sum(n)

print ((dt.datetime.now()-start).microseconds)

start=dt.datetime.now()

c=numpy\_sum(n)

print ((dt.datetime.now()-start).microseconds)

return 0

if \_\_name\_\_=='\_\_main\_\_':

sys.exit(main(len(sys.argv),sys.argv,os.environ))

def main(argc,argv,envp):

one\_dim=np.arange(1,5)

print (one\_dim)

print(type(one\_dim))

print(one\_dim.dtype)

print(one\_dim.shape)

two\_dim=np.array([[1,2,3],

[4,5,6],

[7,8,9]])

print(two\_dim)

print(type(two\_dim))

print(two\_dim.dtype)

print(two\_dim.shape)

three\_dim=np.arange(1,25).reshape(2,3,4)

print(three\_dim)

print(type(three\_dim))

print(three\_dim.dtype)

print(three\_dim.shape)

print(three\_dim[1,0,0])

for i in range(three\_dim.shape[0]):

for j in range(three\_dim.shape[1]):

for k in range(three\_dim.shape[2]):

print('{:4}'.format(three\_dim[i,j,k]),end='')

print()

print()

return 0

if \_\_name\_\_=='\_\_main\_\_':

sys.exit(main(len(sys.argv),sys.argv,os.environ))

二 Linear algebra

import numpy.linalg as nplin

1 Inverse

1.1 nplin.inv(array): Computes the inverse matrix.

inv = nplin.inv(arr3)

inv

1.2 np.allclose(array1, array2): Returns True if two arrays are ele- ment-wise equal within a tolerance.

np.allclose(np.identity(3), np.dot(inv, arr3))

2 Matrix functions

2.1 nplin.det(array): Computes the determinant.

nplin.det(arr3)

2.2 np.trace(array): Computes the trace.

np.trace(arr3)

2.3 np.diag(array): Returns the diagonal elements as an array.

np.diag(arr3)

3 nplin.eig(array): Returns the array of eigenvalues and the array of eigenvectors as a list.

A = np.array([[3, -1, 0], [2, 0, 0], [-2, 2, -1]])

eigenval, eigenvec = nplin.eig(A)

eigenval

eigenvec

4 nplin.qr(array): Conducts a QR decomposition and returns Q and R as lists.

Q,R = nplin.qr(arr3)

Q

R

5 nplin.solve(A, b): Returns the solution of the linearsystem Ax = b.

b = np.array([7, 4, 8])

x = nplin.solve(A, b)

x

三 pandas

1, Series

1.1 Importing Pandas and creating a Series

import numpy as np

import pandas as pd

obj = pd.Series([2, -5, 9, 4])

obj

1.2 Series indexing

obj2 = pd.Series([2, -5, 9, 4], index=["a", "b", "c", "d"])

npobj = np.array([2, -5, 9, 4])

obj2

obj2["b"]

1.3 Series creation from Numpy arrays

npobj = np.array([2, -5, 9, 4])

obj2=pd.Series(npobj, index=["a", "b", "c", "d"])

obj2

1.4 Series from dicts

dictdata = {"Göttingen": 117665, "Northeim": 28920,

"Hannover": 532163, "Berlin": 3574830}

obj3 = pd.Series(dictdata)

obj3

1.5 Series properties

obj.values

obj.index

obj2.index

1.6 Series manipulation

obj2[["c", "d", "a"]]

obj2[obj2 < 0]

obj2 \* 2

np.exp(obj2)["a":"c"]

2 DataFrame

2.1 Creating a DataFrame:pd.DataFrame()

data = {"company": ["Daimler", "E.ON", "Siemens", "BASF", "BMW"],

"price": [69.2, 8.11, 110.92, 87.28, 87.81],

"volume": [4456290, 3667975, 3669487, 1778058, 1824582]}

frame = pd.DataFrame(data)

frame

2.2 Print DataFrame

frame2 = pd.DataFrame(data, columns=["company", "volume", "price", "change"])

frame2

2.3 Add data to DataFrame

frame2["change"] = [1.2, -3.2, 0.4, -0.12, 2.4]

frame2["change"]

2.4 Indexing DataFrames

frame2[["company", "change"]]

2.5 del DataFrame[column]: Deletes column from DataFrame.

del frame2["volume"]

frame2

2.6 Naming properties

frame2.index.name = "number:"

frame2.columns.name = "feature:"

frame2

2.7 DataFrame.reindex(): Creates new DataFrame with data conformed to a new index, while the initial DataFrame will not be changed.

frame3 = frame.reindex([0, 2, 3, 4])

frame3

2.8 Filling missing values

frame4 = frame.reindex(index=[0, 2, 3, 4, 5], fill\_value=0, columns=["company", "price", "market cap"])

frame4

2.9 DataFrame.fillna(value): Fills NaNs with value.

frame4[:3]

frame4.fillna(1000000, inplace=True)

frame4[:3]

2.10 Dropping column

frame4[:2]

frame4.drop("price", axis=1)[:3]

2.11 Hierarchical indexing

Multiindex

ind = [["a", "a", "a", "b", "b"], [1, 2, 3, 1, 2]]

frame6=pd.DataFrame(np.arange(15).reshape((5, 3)), index=ind,

columns=["first", "second", "third"])

frame6

Selecting of a multiindex

frame6.loc["a"]

frame6.loc["b", 1]

2.12 NumPy functions on DataFrames:DataFrame.apply(np.function, axis)

frame6.apply(np.mean)

frame6.apply(np.sqrt)[:2]

四 import and export data

1 Reading data

df = pd.read\_csv("data/ex2.csv", names=["a", "b", "c", "d", "hello"])

df

2 writing data

df.to\_csv("out/out3.csv", index=False,

header=["a", "b", "c", "d", "e"])

df

五 matplotlib

1， Import matplotlib and simple example

import matplotlib.pyplot as plt

import numpy as np

plt.plot(np.arange(10))

plt.savefig("out/list.pdf")

2， Figures

plt.figure(...): Creates new Figure object allowing for multiple parameters.

plt.gcf(): Returns the reference of the active figure.

fig = plt.figure(figsize=(16, 8))

print(plt.gcf())

3，fig.add\_subplot(): Adds subplot to the Figure fig.

ax1 = fig.add\_subplot(2, 2, 1)

ax2 = fig.add\_subplot(2, 2, 2)

ax3 = fig.add\_subplot(2, 2, 3)

ax4 = fig.add\_subplot(2, 2, 4)

fig.savefig("out/subplots.pdf")

4，Filling subplots with content

from numpy.random import randn

ax1.plot([5, 7, 4, 3, 1])

ax2.hist(randn(100), bins=20, color="r")

ax3.scatter(np.arange(30), np.arange(30) \* randn(30))

ax4.plot(randn(40), "k--")

fig.savefig("out/content.pdf")