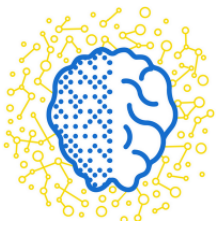
Школа машинного обучения



(https://mipt.ru/science/labs/laboratoriya-neyronnykh-sistem-i-glubokogo-obucheniya/)

Физтех-Школа Прикладной математики и информатики МФТИ ¶

Лаборатория нейронных сетей и глубокого обучения (DeepHackLab)

Домашнее задание 1

Основы Python и пакет NumPy

In [1]:
```
import numpy as np
import random
import scipy.stats as sps
```

Задача 1

В первой задаче вам предлагается перемножить две квадратные матрицы двумя способами -- без использования пакета ***numpy*** и с ним.

In [2]:
```
# Для генерации матриц используем фукнцию random -- она используется д
# функция sample создает случайную выборку. В качестве аргумента ей пе
# j -- число столбцов.
a = np.random.sample((10,10))
b = np.random.sample((10,10))
# выведите размерность (ранг) каждой матрицы с помощью функции ndim.
# Используйте функцию shape, что она вывела?
print(a.ndim, b.ndim)
print(a.shape)
print(a)
print(b)
```

```
2 2
(10, 10)
[[ 0.33790108  0.06816307  0.61695521  0.27534553  0.89298771  0.615
98249
   0.565858    0.7909646   0.43892302  0.66926666]
 [ 0.22933652  0.68415982  0.64447742  0.44621457  0.58485137  0.377
```

```
01965
   0.02281789  0.54583338  0.32602297  0.70160143]
 [ 0.14481165  0.16856471  0.89449731  0.38396727  0.00220337  0.222
74688
   0.2087503   0.64941996  0.0544724   0.32040479]
 [ 0.8989965   0.87344429  0.52549747  0.02895221  0.92955496  0.686
75364
   0.14713093  0.01650334  0.64327871  0.42266111]
 [ 0.30214697  0.15916595  0.60155549  0.65923151  0.492974    0.895
41905
   0.71388045  0.07550848  0.65706871  0.95859022]
 [ 0.01019631  0.23138435  0.39136443  0.89127071  0.78404961  0.858
29139
   0.38192854  0.01653532  0.01540367  0.79897027]
 [ 0.67171373  0.69506427  0.60047581  0.2742897   0.0885571   0.099
34099
   0.97511301  0.07465765  0.75014785  0.43062253]
 [ 0.40727846  0.81900615  0.18028406  0.07035022  0.72100209  0.961
0883
   0.84092582  0.47243448  0.44455317  0.49500255]
 [ 0.45645811  0.73795721  0.49480419  0.02639901  0.63202405  0.990
0247
   0.64215502  0.36043774  0.28842647  0.25711265]
 [ 0.19070368  0.35744964  0.29951034  0.5175269   0.86632314  0.455
54246
   0.39284969  0.49852826  0.15674109  0.00712162]]
[[ 0.4776652   0.33001078  0.97602209  0.31332461  0.69345258  0.174
743
   0.06311247  0.97172278  0.54851532  0.90075045]
 [ 0.66977238  0.62324754  0.39820291  0.04655874  0.25979894  0.065
48909
   0.17909429  0.32449723  0.50478918  0.8040335 ]
 [ 0.1385158   0.2676978   0.58697409  0.31647299  0.57098481  0.817
74176
   0.9606258   0.72098162  0.60860208  0.23239673]
 [ 0.11906318  0.30738309  0.53700844  0.07939507  0.37605143  0.679
30392
   0.29347841  0.61083667  0.00180204  0.65646103]
 [ 0.14809581  0.73345386  0.39835399  0.14065782  0.06334674  0.765
62447
   0.15401125  0.86624816  0.78929209  0.43279518]
 [ 0.95186725  0.89463021  0.30600748  0.34208804  0.89923923  0.333
39772
   0.71053804  0.96219827  0.23463552  0.35718574]
 [ 0.16181507  0.05546226  0.10071493  0.94911705  0.2646791   0.650
0759
   0.97604322  0.33075193  0.44488752  0.5714193 ]
 [ 0.03171794  0.86512546  0.41206402  0.43222382  0.65763278  0.604
5783
   0.15830018  0.24811462  0.57429877  0.85060777]
 [ 0.1898324   0.93013975  0.16157618  0.06119685  0.67744261  0.600
97559
   0.7840696   0.7125241   0.08616063  0.15991355]
 [ 0.83197376  0.65955473  0.42854675  0.80942882  0.39500782  0.467
```

```
55747
   0.12399606  0.08439576  0.37660836  0.7154411 ]]
```

In [3]:
```
def mult(a, b):
    r = np.zeros((a.shape[0], b.shape[1]))
    for i in range(a.shape[0]):
        for j in range(b.shape[1]):
            for k in range(a.shape[1]):
                r[i][j] += a[i][k]*b[k][j]
    return r
```

In [4]:
```
def np_mult(a, b):
    return a @ b
```

In [5]:
```
%%time
# засечем время работы функции без NumPy
mult(a,b)
```

```
CPU times: user 1.78 ms, sys: 32 µs, total: 1.81 ms
Wall time: 1.84 ms
```

Out[5]:
```
array([[ 1.80066642,  3.1751745 ,  2.15181168,  2.11000664,  2.54997243,
         3.06687685,  2.38685793,  3.08235067,  2.44095028,  2.83495628],
       [ 1.82227084,  2.81749445,  2.04309615,  1.39976219,  2.11164097,
         2.35728859,  1.69631687,  2.48596188,  2.03020837,  2.61856489],
       [ 0.89532783,  1.54661805,  1.44347942,  1.18480118,  1.64558709,
         1.8154655 ,  1.55889287,  1.58713715,  1.35480132,  1.71605149],
       [ 2.38011609,  3.1863456 ,  2.43638044,  1.38488631,  2.4902445 ,
         2.29417211,  2.06059879,  3.56636412,  2.43825435,  2.80415481],
       [ 2.37823907,  3.07355202,  2.15575605,  2.24676299,  2.74106703,
         3.03172645,  2.87384792,  3.27418405,  1.99115844,  2.76892297],
       [ 1.98323956,  2.44600111,  1.77556781,  1.62977448,  1.88528439,
         2.46995475,  1.89675078,  2.62543843,  1.66405179,  2.32354957],
       [ 1.67072166,  2.15418558,  1.93253992,  1.85330717,  2.17280003,
         2.27237335,  2.51356495,  2.56292764,  1.8819583 ,  2.60606741],
       [ 2.44532346,  3.29870893,  2.01188638,  2.08878319,  2.56447444,
         2.52335849,  2.4656531 ,  3.13769191,  2.41141966,  3.07639042],
       [ 2.20395743,  2.88570121,  1.96869357,  1.75467601,  2.4349659 ,
         2.29352561,  2.38668616,  3.08500967,  2.26958754,  2.66758081],
       [ 1.11058333,  2.17148729,  1.54006127,  1.09365397,  1.5960858 ,
         2.12267719,  1.55889144,  2.38805217,  1.73618933,  2.08486716]])
```

In [6]:
```
%%time
# засечем время работы функции с NumPy
np_mult(a,b)
```

```
CPU times: user 93.5 ms, sys: 16.2 ms, total: 110 ms
Wall time: 133 ms
```

Out[6]:
```
array([[ 1.80066642,  3.1751745 ,  2.15181168,  2.11000664,  2.54997243,
         3.06687685,  2.38685793,  3.08235067,  2.44095028,  2.83495628],
       [ 1.82227084,  2.81749445,  2.04309615,  1.39976219,  2.11164097,
         2.35728859,  1.69631687,  2.48596188,  2.03020837,  2.61856489],
       [ 0.89532783,  1.54661805,  1.44347942,  1.18480118,  1.64558709,
         1.8154655 ,  1.55889287,  1.58713715,  1.35480132,  1.71605149],
       [ 2.38011609,  3.1863456 ,  2.43638044,  1.38488631,  2.4902445 ,
         2.29417211,  2.06059879,  3.56636412,  2.43825435,  2.80415481],
       [ 2.37823907,  3.07355202,  2.15575605,  2.24676299,  2.74106703,
         3.03172645,  2.87384792,  3.27418405,  1.99115844,  2.76892297],
       [ 1.98323956,  2.44600111,  1.77556781,  1.62977448,  1.88528439,
         2.46995475,  1.89675078,  2.62543843,  1.66405179,  2.32354957],
       [ 1.67072166,  2.15418558,  1.93253992,  1.85330717,  2.17280003,
         2.27237335,  2.51356495,  2.56292764,  1.8819583 ,  2.60606741],
       [ 2.44532346,  3.29870893,  2.01188638,  2.08878319,  2.56447444,
         2.52335849,  2.4656531 ,  3.13769191,  2.41141966,  3.07639042],
       [ 2.20395743,  2.88570121,  1.96869357,  1.75467601,  2.4349659 ,
         2.29352561,  2.38668616,  3.08500967,  2.26958754,  2.66758081],
       [ 1.11058333,  2.17148729,  1.54006127,  1.09365397,  1.5960858 ,
         2.12267719,  1.55889144,  2.38805217,  1.73618933,  2.08486716]])
```

Задача 2

Напишите функцию, которая по данной последовательности $\{A_i\}_{i=1}^n$ строит последовательность S_n, где $S_k = \frac{A_1+...+A_k}{k}$.
Аналогично -- с помощью библиотеки **NumPy** и без нее. Сравните скорость, объясните результат.

In [7]:
```
# функция, решающая задачу с помощью NumPy
def sec_av(A):
    A = A.cumsum() / np.arange(1, len(A) + 1)
    return A
```

In [8]:
```
# функция без NumPy
def stupid_sec_av(A):
    S = [0 for i in range(len(A))]
    v = 0
    for i in range(len(A)):
        v += A[i]
        S[i] = v
    for i in range(len(A)):
        S[i] /= (i + 1)
    return S

# зададим некоторую последовательность и проверим ее на ваших функциях.
# Первая функция должна работать ~ в 50 раз быстрее
A = sps.uniform.rvs(size=10 ** 7)

%time S1 = sec_av(A)
%time S2 = stupid_sec_av(A)
#проверим корректность:
print(np.abs(S1 - S2).sum())
```

```
CPU times: user 94 ms, sys: 38.9 ms, total: 133 ms
Wall time: 134 ms
CPU times: user 6.97 s, sys: 204 ms, total: 7.18 s
Wall time: 7.25 s
0.0
```

Задача 3

Пусть задан некоторый массив X. Надо построить новый массив, где все элементы с нечетными индексами требуется заменить на число a (если оно не указано, то на 1). Все четные элементы исходного массива нужно возвести в куб и записать в обратном порядке относительно позиций этих элементов. Массив X при этом должен остаться без изменений. В конце требуется слить массив X с преобразованным X и вывести в обратном порядке.

In [9]:
```
# функция, решающая задачу с помощью NumPy
def transformation(X, a=1):
    # Обнулим все четные позиции
    v = np.array([0, 1])
    d = np.outer(np.ones_like(X[::2]), v).ravel()
    n = X * d
    # Четные возводим в степень
    j = n ** 3
    j = j[1::]
    j = np.append(j, 0)
    j = j[::-1]
    # Из нечетных создаем массив [a 0 a 0 a 0 ... a 0]
    v = np.array([a, 0])
    k = np.outer(np.ones_like(X[::2]), v).ravel()
    # Складываем эта два чуда
    r = j + k
    # Ну и сливаем с изначальным
    Y = X + r
    return Y
```

In [10]:
```
# функция, решающая задачу  без NumPy
def stupid_transformation(X, a=1):
    j = []
    k = []
    for i in range(len(X)):
        if i % 2 == 0: # Индексация с нуля, поэтому это нечетные
            j.append(a)
        else:
            k.append(X[i] ** 3)
    k = k[::-1]
    S = []

    fst = True
    while len(j) > 0 or len(k) > 0:
        if fst:
            fst = False
            if len(j):
                S.append(j.pop(0))
        else:
            fst = True
            if len(k):
                S.append(k.pop(0))
    # Ну мы ж совсем без нампи)))0))0)
    Y = [0 for _ in range(len(X))]
    for i in range(len(S)):
        Y[i] = S[i] + X[i]
    return Y
```

In [11]:
```
X = sps.uniform.rvs(size=10 ** 5) # Я перестарался с неэффективностью,
# здесь код эффективнее примерно в 20 раз.
# если Вы вдруг соберетесь печатать массив без np -- лучше сначала пос
%time S1 = transformation(X, 5)
%time S2 = stupid_transformation(X, 5)
# проверим корректность:
np.abs(S1 - S2).sum()
```

```
CPU times: user 6.35 ms, sys: 10.6 ms, total: 17 ms
Wall time: 17.6 ms
CPU times: user 1.37 s, sys: 87.7 ms, total: 1.46 s
Wall time: 1.5 s
```

Out[11]:
```
0.0
```

Вопрос человека: Почему методы ***numpy*** оказываются эффективнее?

Ответ человека: А почему ActiveRecord медленнее нативных запросов к БД? Ответ прост, потому что долполнительные обертки (в данном случае сам Питон) расходуют доп. время и память.

Дополнительные задачи

Дополнительные задачи подразумевают, что Вы самостоятельно разберётесь в некоторых функциях ***numpy***, чтобы их сделать.

Эти задачи не являются обязательными, но могут повлиять на Ваш рейтинг в лучшую сторону (точные правила учёта доп. задач будут оглашены позже).

Задача 4*

Дана функция двух переменных: $f(x, y) = sin(x)cos(y)$ (это просто такой красивый 3D-график), а также дана функция для отрисовки $f(x, y)$ (`draw_f()`), которая принимает на вход двумерную сетку, на которой будет вычисляться функция.

Вам нужно разобраться в том, как строить такие сетки (подсказка - это одна конкретная функция ***numpy***), и подать такую сетку на вход функции отрисовки.

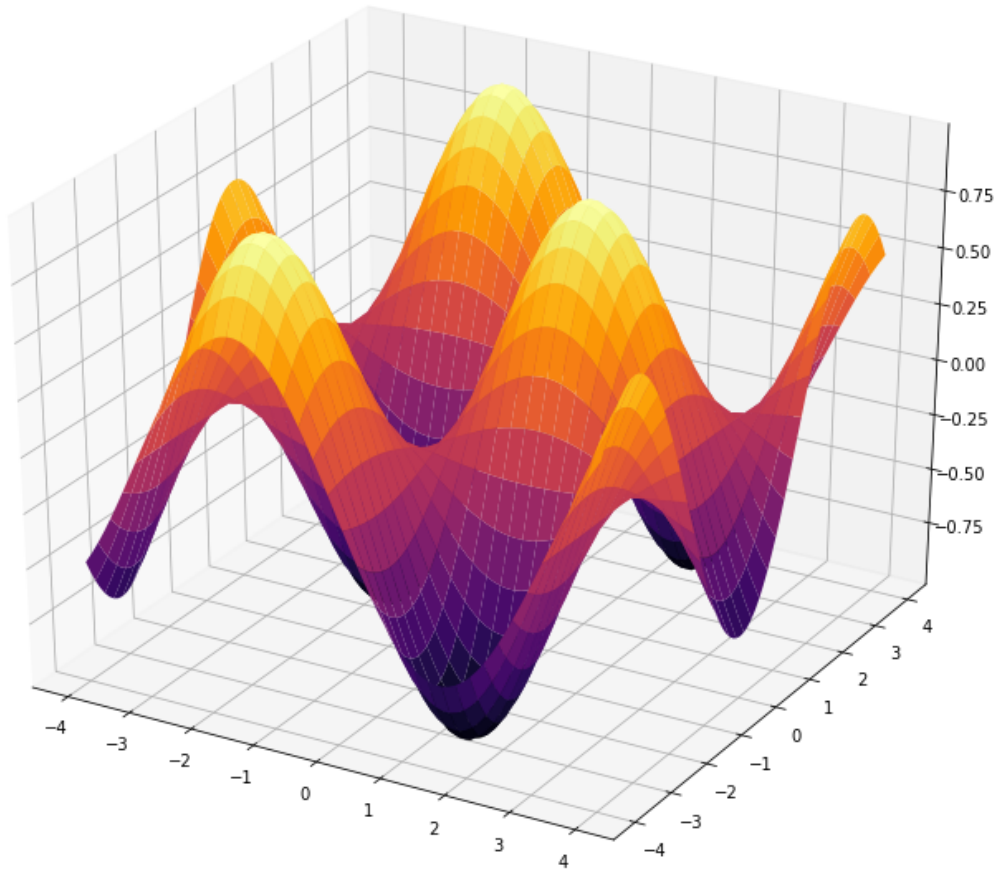
In [12]:
```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

def f(x, y):
    '''Функция двух переменных'''
    return np.sin(x) * np.cos(y)

def draw_f(grid_x, grid_y):
    '''Функция отрисовки функции f(x, y)'''
    fig = plt.figure(figsize=(10, 8))
    ax = Axes3D(fig)
    ax.plot_surface(grid_x, grid_y, f(grid_x, grid_y), cmap='inferno')
    plt.show()
```

In [13]:
```
delailed = 75
grid_x, grid_y = np.meshgrid(np.linspace(-4, 4, delailed), np.linspace

draw_f(grid_x, grid_y)
```



Задача 5*

Вам дана картинка. При загрузке её размерность равна 3: **(w, h, num_channels)**, где **w** - ширина картинки в пикселях, **h** - высота картинки в пикселях, **num_channels** - количество каналов *(R, G, B, alpha)*.

Вам нужно "развернуть" картинку в одномерный массив размера w * h * num_channels, написав **одну строку кода**.

In [14]:
```
from matplotlib import pyplot as plt
%matplotlib inline
```

In [15]:
```
path_to_image = './image.png'
image_array = plt.imread(path_to_image)
plt.imshow(image_array);
```

In [16]:
```
flat_image_array = np.ravel(image_array)
```

In [17]:
```
flat_image_array.size
```

Out[17]:
```
42336
```