Математическая статистика

Практическое задание 0

В данном задании предлагается решить 4 простых задачи на использование функций библиотеки `numpy`. Хоть само задание и не относится к курсу статистики, оно является важным в условиях отсутствия курса по Питону. Решение этих задач поможет научить писать простой и понятный код, работающий при этом в десятки или даже в сотни раз быстрее. Нам же это облегчит процесс проверки.

Правила:

- Задание считается выполненным, если решено *не менее трех задач*.
- Успешное выполнение задание является допуском для выполнения следующих практических заданий.
- В случае неуспешного выполнения задания допускаются две попытки повторной сдачи. Мы будем стараться отвечать в течении трех дней.
- Выполненную работу нужно отправить на почту `probability.diht@yandex.ru`, указав тему письма `"[номер группы] Фамилия Имя - Задание 0"`. Квадратные скобки обязательны.
- Прислать нужно ноутбук и его pdf-версию. Названия файлов должны быть такими: `0.N.ipynb` и `0.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- В данном задании весь присылаемый код должен корректно работать на `Python 3.5`.

Во всех заданиях предполагается, что все аргументы функций, которые нужно реализовать, имеют тип `numpy.array` либо являются числами. Возвращать нужно также либо `numpy.array`, либо число. Кроме того, предполагается, что все аргументы корректны, и проверять их на корректность не нужно.

При реализации запрещается пользоваться любыми циклами, в том числе стандартными функциями языка, которые заменяют циклы. Можно использовать любые функции библиотек `numpy` или `scipy`, кроме функции `numpy.fromfunction` и декоратора `numpy.vectorize`.

In [128]:
```
import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt

%matplotlib inline
```

Задача 1. Напишите функцию, реализующую матричное умножение. При вычислении разрешается создавать объекты размерности три. Запрещается пользоваться функциями, реализующими матричное умножение (`numpy.dot`, операция `@`, операция умножения в классе `numpy.matrix`). *Авторское решение занимает одну строчку.*

In [129]:
```
def matrix_multiplication(A, B):
    return <Тут ваш код>
```

```
  File "<ipython-input-129-5d523404e449>", line 2
    return <Тут ваш код>
           ^
SyntaxError: invalid syntax
```

Проверьте правильность реализации на случайных матрицах. Должен получится ноль.

In [130]:
```
A = sps.uniform.rvs(size=(10, 20))
B = sps.uniform.rvs(size=(20, 30))
np.abs(matrix_multiplication(A, B) - A @ B).sum()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-130-2312eaf828a2> in <module>()
      1 A = sps.uniform.rvs(size=(10, 20))
      2 B = sps.uniform.rvs(size=(20, 30))
----> 3 np.abs(matrix_multiplication(A, B) - A @ B).sum()

NameError: name 'matrix_multiplication' is not defined
```

А вот в таком стиле вы присылали бы нам свои работы, если не стали бы делать это задание.

In [131]:
```
def stupid_matrix_multiplication(A, B):
    C = [[0 for j in range(len(B[0]))] for i in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

Проверьте, насколько быстрее работает ваш код по сравнению с неэффективной реализацией `stupid_matrix_multiplication`. Эффективный код должен работать почти в 200 раз быстрее. Для примера посмотрите также, насколько быстрее работают встроенные `numpy`-функции.

In [132]:
```
A = sps.uniform.rvs(size=(400, 200))
B = sps.uniform.rvs(size=(200, 300))

%time C1 = matrix_multiplication(A, B)
%time C2 = A @ B  # python 3.5
%time C3 = np.matrix(A) * np.matrix(B)
%time C4 = stupid_matrix_multiplication(A, B)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-132-49ebb7c90123> in <module>()
      2 B = sps.uniform.rvs(size=(200, 300))
      3
----> 4 get_ipython().magic('time C1 = matrix_multiplication(A, B)')
      5 get_ipython().magic('time C2 = A @ B  # python 3.5')
      6 get_ipython().magic('time C3 = np.matrix(A) * np.matrix(B)')

/opt/anaconda/lib/python3.6/site-packages/IPython/core/interactiveshell.py in magic(self, arg_s)
   2156         magic_name, _, magic_arg_s = arg_s.partition(' ')
   2157         magic_name = magic_name.lstrip(prefilter.ESC_MAGIC)
-> 2158         return self.run_line_magic(magic_name, magic_arg_s)
   2159
   2160     #-------------------------------------------------------------------------

/opt/anaconda/lib/python3.6/site-packages/IPython/core/interactiveshell.py in run_line_magic(self, ma
gic_name, line)
   2077                 kwargs['local_ns'] = sys._getframe(stack_depth).f_locals
   2078             with self.builtin_trap:
-> 2079                 result = fn(*args,**kwargs)
   2080             return result
   2081

<decorator-gen-59> in time(self, line, cell, local_ns)

/opt/anaconda/lib/python3.6/site-packages/IPython/core/magic.py in <lambda>(f, *a, **k)
    186     # but it's overkill for just that one bit of state.
    187     def magic_deco(arg):
--> 188         call = lambda f, *a, **k: f(*a, **k)
    189
    190         if callable(arg):

/opt/anaconda/lib/python3.6/site-packages/IPython/core/magics/execution.py in time(self, line, cell,
 local_ns)
```

```
   1178         else:
   1179             st = clock2()
-> 1180             exec(code, glob, local_ns)
   1181             end = clock2()
   1182             out = None

<timed exec> in <module>()

NameError: name 'matrix_multiplication' is not defined
```

Ниже для примера приведена полная реализация функции. Вас мы, конечно, не будем требовать проверять входные данные на корректность, но документации к функциям нужно писать.

In [133]:
```
def matrix_multiplication(A, B):
    '''Возвращает матрицу, которая является результатом
    матричного умножения матриц A и B.

    '''

    # Если A или B имеют другой тип, нужно выполнить преобразование типов
    A = np.array(A)
    B = np.array(B)

    # Проверка данных входных данных на корректность
    assert A.ndim == 2 and B.ndim == 2, 'Размер матриц не равен 2'
    assert A.shape[1] == B.shape[0], ('Матрицы размерностей '
                                      '{} и {} неперемножаемы'.format(A.shape,
                                                                      B.shape))


    C = <Тут ваш код>

    return C
```

```
  File "<ipython-input-133-4aabf17dcc81>", line 17
    C = <Тут ваш код>
        ^
SyntaxError: invalid syntax
```

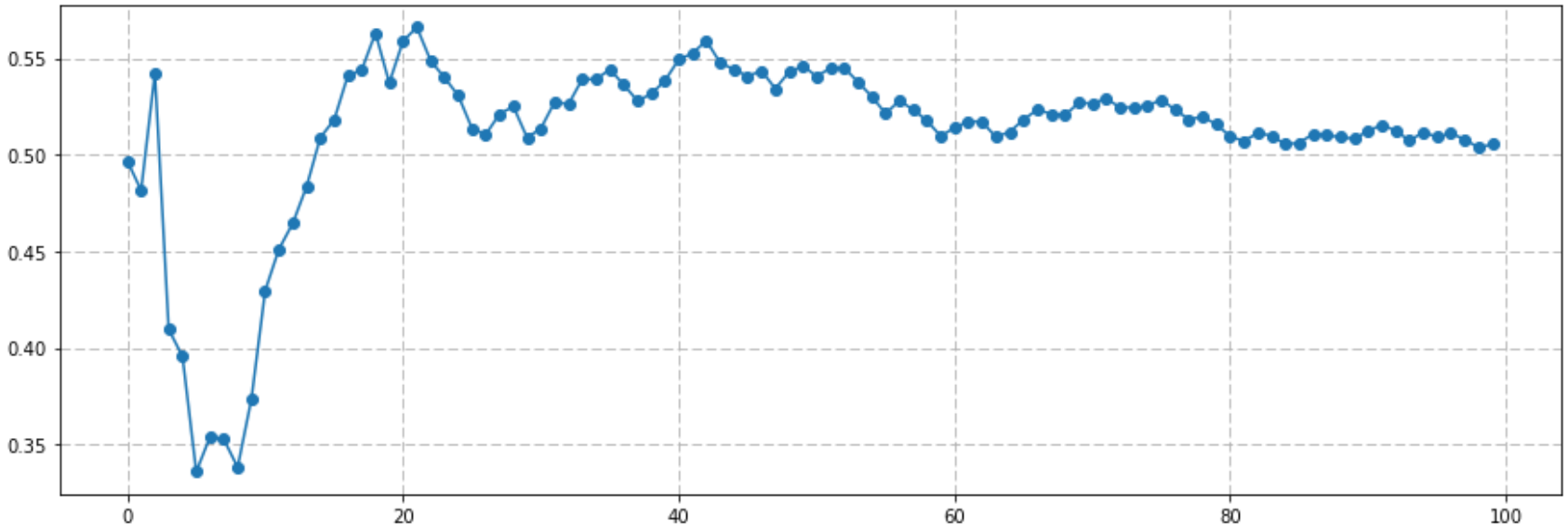
Задача 2. Напишите функцию, которая по входной последовательности $X = (X_1, \ldots, X_n)$ строит последовательность $S = (S_1, \ldots, S_n)$, где $S_k = \frac{X_1 + \ldots + X_k}{k}$. *Авторское решение занимает одну строчку.*

In [134]:
```
def cumavg(X):
    return X.cumsum()/(np.arange(1,len(X)+1))
```

Постройте график зависимости S_k от k. График должен быть в виде ломанной линии с достаточно крупными точками. Размер фигуры 15 на 5, сетка в виде пунктирной линии.

In [135]:
```
S = cumavg(sps.uniform.rvs(size=100))

plt.figure(figsize=(15, 5))
plt.grid(linestyle="--")  # Сетка пунктиром
plt.plot(S, '-o')  # График ломаной с достаточно большими точками
plt.show()
```



Проверьте корректность работы реализации, а также ее эффективность. Эффективный код должен работать в 50 раз быстрее.

In [136]:
```
def stupid_cumavg(X):
    S = [0 for i in range(len(X))]
    for i in range(len(X)):
        S[i] = X[i] + S[i - 1]
    for i in range(len(X)):
        S[i] /= i + 1
    return S

X = sps.uniform.rvs(size=10 ** 7)

%time S1 = cumavg(X)
%time S2 = stupid_cumavg(X)

np.abs(S1 - S2).sum()
```

```
CPU times: user 80 ms, sys: 16.7 ms, total: 96.7 ms
Wall time: 96.2 ms
CPU times: user 5.85 s, sys: 70 ms, total: 5.92 s
Wall time: 5.93 s
```

Out[136]:
```
0.0
```

Задача 3. Дана матрица $A = (a_{ij})$ размера $n \times m$. Вычислите величину

$$\frac{1}{m} \sum_{j=1}^{m} \min_{i=1,\dots,n} a_{ij},$$

то есть средний минимум по столбцам. *Авторское решение занимает одну строчку.*

In [137]:
```
def avgmin(A):
    return  (A.min(axis=0).sum())/A.shape[1]
```

Проверьте корректность работы реализации, а также ее эффективность. Эффективный код должен работать почти в 200 раз быстрее. Обратите внимание, что разность чисел может быть не равна нулю из-за ошибок округления, но должна иметь малый порядок.

In [138]:
```
def stupid_avgmin(A):
    N, M = len(A), len(A[0])
    min_col = [min([A[i][j] for i in range(N)]) for j in range(M)]
    return sum(min_col) / M

N, M = 5000, 10000
A = sps.uniform.rvs(size=(N, M))

%time S1 = avgmin(A)
%time S2 = stupid_avgmin(A)

print(np.abs(S1 - S2))
```

```
CPU times: user 56.7 ms, sys: 0 ns, total: 56.7 ms
Wall time: 56.1 ms
CPU times: user 17.3 s, sys: 26.7 ms, total: 17.3 s
Wall time: 17.3 s
1.62630325873e-19
```

Задача 4. Дан массив X. Требуется построить новый массив, в котором все четные элементы X заменить на число v (если оно не указано, то на ноль). Все нечетные элементы исходного массива нужно возвести в квадрат и записать в обратном порядке относительно позиций этих элементов. Массив X при этом должен остаться без изменений.

In [139]:
```
def func4(X, v=0):
    result = X.copy()  # Копируем массив
    even = result % 2 == 0
    odd = np.logical_not(even)
    result[even] = v
    result[odd] = (((result[odd])**2)[::-1])
    return result
```

Проверьте корректность работы реализации, а также ее эффективность. Эффективный код должен работать в 20 раз быстрее.

In [140]:
```
def stupid_func4(X, v=0):
    odd = [elem ** 2 for elem in X if elem % 2]

    new_X = []
    j = len(odd) - 1
    for i in range(len(X)):
        if X[i] % 2:
            new_X.append(odd[j])
            j -= 1
        else:
            new_X.append(v)

    return new_X

X = sps.randint.rvs(size=10 ** 7, low=0, high=100)

%time A1 = func4(X)
%time A2 = stupid_func4(X)

np.abs(A1 - A2).sum()
```

```
CPU times: user 323 ms, sys: 30 ms, total: 353 ms
Wall time: 355 ms
CPU times: user 7.27 s, sys: 60 ms, total: 7.33 s
Wall time: 7.33 s
```

Out[140]:
```
0
```

Вопрос: За счет чего достигается такая эффективность методов numpy?

Ответ: Потому что на питоне фактически осуществляется лишь обращение к низкоуровневым библиотекам, написанным на C, например.