

# Матрицы, питру и классы

Егор Суворов

Курс «Парадигмы и языки программирования», подгруппа 3

Среда, 21 сентября 2016 года

# Организационное

- Я студент третьего курса бакалавриата СПб АУ (программист).
- Имена: «извините, пожалуйста», «Егор», «Егор Фёдорович».  
E-mail: `egor_suvorov@mail.ru`
- Тема e-mail: `[parad]...`
- VK: `vk.com/egor.suvorov`
- Telegram: `@yeputons`
- Домашние задания общие с остальными подгруппами.
- Решения надо присылать мне. Если уже прислали кому-то ещё — перенаправлять не надо.
- Вопросы по текущей теме или смежным можно задавать в ходе рассказа на занятии.
- Вопросы по остальным темам и предметам лучше в оффлайне.
- А хотите Office Hours с 10 до 11?
- Слайды — на SEWiki.

## Зачёт и проверка

- Надо набрать хотя бы половину баллов от максимума.
- Надо набрать строго положительные баллы по каждой теме.

# Кто такие матрицы

*Матрица* — это квадратная<sup>1</sup> табличка<sup>2</sup> с чиселками<sup>3</sup>.

---

1

2

3

# Кто такие матрицы

*Матрица* — это квадратная<sup>1</sup> табличка<sup>2</sup> с чиселками<sup>3</sup>.

---

<sup>1</sup>прямоугольная

<sup>2</sup>

<sup>3</sup>

# Кто такие матрицы

*Матрица* — это квадратная<sup>1</sup> табличка<sup>2</sup> с чиселками<sup>3</sup>.

---

<sup>1</sup>прямоугольная

<sup>2</sup>алгебраический объект

<sup>3</sup>

# Кто такие матрицы

*Матрица* — это квадратная<sup>1</sup> табличка<sup>2</sup> с чиселками<sup>3</sup>.

---

<sup>1</sup>прямоугольная

<sup>2</sup>алгебраический объект

<sup>3</sup>необязательно числа

# Кто такие матрицы

*Матрица* — это квадратная<sup>1</sup> табличка<sup>2</sup> с чиселками<sup>3</sup>.

Полезны при изучении:

- Почти чего угодно со словом «линейное»:
  - Систем линейных уравнений.
  - Линейных преобразований (плоскости, пространства...).
  - Линейных рекуррент ( $F_n = F_{n-1} + F_{n-2}$ ).
- Графов (матрица смежности/инцидентности).
- Машинного обучения (привет от линейной алгебры).

---

<sup>1</sup>прямоугольная

<sup>2</sup>алгебраический объект

<sup>3</sup>необязательно числа



## Операции с матрицами

Запись матрицы  $A$  размера  $2 \times 3$ :


$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \end{pmatrix}$$

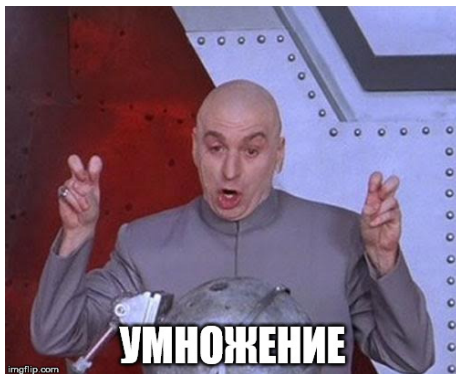
Сложение матриц одного размера:

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ 0 & A_{3,2} \end{pmatrix} + \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & 0 \\ B_{3,1} & B_{3,2} \end{pmatrix} = \begin{pmatrix} A_{1,1} + B_{1,1} & A_{1,2} + B_{1,2} \\ A_{2,1} + B_{2,1} & A_{2,2} \\ B_{3,1} & A_{3,2} + B_{3,2} \end{pmatrix}$$

Умножение<sup>4</sup> матрицы с  $k$  столбцами на матрицу с  $k$  строками:

$$\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} \\ B_{2,1} \\ B_{3,1} \end{pmatrix} = \begin{pmatrix} A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} + A_{1,3} \cdot B_{3,1} \\ A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} + A_{2,3} \cdot B_{3,1} \end{pmatrix}$$

<sup>4</sup> интуитивно можно думать «композиция отображений» 



## Wait But Why

Например, если матрица описывает преобразование координат:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} ax_1 + by_1 \\ cx_1 + dy_1 \end{pmatrix} = \underbrace{\begin{pmatrix} a & b \\ c & d \end{pmatrix}}_A \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} a'x_2 + b'y_2 \\ c'x_2 + d'y_2 \end{pmatrix} = \underbrace{\begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix}}_{A'} \cdot \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$$

$$\begin{aligned} \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} &= \begin{pmatrix} a'(ax_1 + by_1) + b'(cx_1 + dy_1) \\ c'(ax_1 + by_1) + d'(cx_1 + dy_1) \end{pmatrix} = \\ &= \begin{pmatrix} (a'a + b'c)x_1 + (a'b + b'd)y_1 \\ (c'a + d'c)x_1 + (c'b + d'd)y_1 \end{pmatrix} = (A' \cdot A) \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \end{aligned}$$

## Wait But Why

Например, если матрица описывает преобразование координат:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} ax_1 + by_1 \\ cx_1 + dy_1 \end{pmatrix} = \underbrace{\begin{pmatrix} a & b \\ c & d \end{pmatrix}}_A \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} a'x_2 + b'y_2 \\ c'x_2 + d'y_2 \end{pmatrix} = \underbrace{\begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix}}_{A'} \cdot \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = A' \cdot \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = A' \cdot \left( A \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \right)$$

$$\begin{aligned} \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} &= \begin{pmatrix} a'(ax_1 + by_1) + b'(cx_1 + dy_1) \\ c'(ax_1 + by_1) + d'(cx_1 + dy_1) \end{pmatrix} = \\ &= \begin{pmatrix} (a'a + b'c)x_1 + (a'b + b'd)y_1 \\ (c'a + d'c)x_1 + (c'b + d'd)y_1 \end{pmatrix} = (A' \cdot A) \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \end{aligned}$$

## Умножение матриц-1

```
a=[[1,2,3],
   [4,5,6]]
b=[[5,4],
   [3,2],
   [1,0]]
c=[[0, 0], [0,0]]
for i in range(2):
    for j in range(2):
        c[i][j] = sum(a[i][k] * b[k][j] for k in range(3))
print(c) # [[14, 8],
           # [41, 26]]
```

Получаем  $O(n \cdot m \cdot k)$ . Если матрицы квадратные — куб.

Python медленный, так что у меня работает 0.65 секунд на матрицах  $150 \times 150$ .

## Умножение матриц-2

1987 год — алгоритм Копперсмита-Винограда,  $O(n^{2,375477})$ .

2010–2014 года — серия улучшений, аж до  $O(n^{2,3728639})$ .

Практика — алгоритм Штрассена,  $O(n^{2,807355})$ . Вам надо будет его реализовать.

# Алгоритм Штрассена

Разделяй-и-властвуй (как алгоритм Карацубы):

- 1 Разделили каждую матрицу на четыре одинаковых.
- 2 ???
- 3 Profit

Цель — чтобы на втором шаге потребовалось строго меньше восьми умножений матриц:

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \\ = \begin{pmatrix} A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} & A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} & A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{pmatrix}$$

Волшебные формулы смотрим в Википедии.

## Велосипед уже изобретён

NumPy — библиотека для работы с многомерными массивами, в том числе с матрицами.

- Написано на C, незаметно «встраивается» в Python.
- Может даже использовать GPU, если хочет.
- Массивы гарантированно хранятся более эффективно.
- Ускоряются только встроенные в библиотеку операции (не циклы).

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = np.array([[7,8,9],[10,11,12]])
print(a.shape)
print(a+b)
b[1][1] += 100
print(a.dot(b.transpose()))
print(b[:, 1:])
print(b[np.array([0, 0])])
```



## Форма и копии

- Можно считать, что все элементы массива расположены в памяти в лексикографическом порядке друг за другом.
- Форма массива — это лишь числа, которые легко меняются:

```
print(np.arange(24))  
print(np.arange(24).reshape(3, 8))  
print(np.arange(24).reshape(3, 8).reshape(2, 3, 4))
```
- Все операции по умолчанию поэлементные (сложение, `np.exp...`).
- Конструкция `a = b` ничего не копирует (только ссылку).
- Можно получить отдельный объект на том же массиве (view):  
`b = a.view()`, `b = a[...]` (`...` — это именно многоточие).
- Можно скопировать целиком (долго): `b = a.copy()`.
- Можно конкатенировать массивы: `vstack`, `hstack`.

## Распространение поэлементных операций

```
import numpy as np
a = np.arange(10).reshape(2, 5)
print(a * 100)
print(a + [5, 4, 3, 2, 1])
print(a + [[100], [1000]])
print(a.reshape(1, 10) * a.reshape(10, 1))
```

- К размерности дописываются единицы:  
 $(3) \rightarrow (1, 3) \rightarrow (1, 1, 3) \rightarrow \dots$
- Если какая-то размерность равна единице, то её можно расширить до максимума:  $(1, 10) \rightarrow (10, 10)$ .

# Матрицы

Матрица — это двухмерный массив.

```
import numpy as np
a = np.arange(4).reshape(2, 2)
print(a * a)  # wtf
```

# Матрицы

Матрица — это двухмерный массив.

```
import numpy as np
a = np.arange(4).reshape(2, 2)
print(a * a) # wtf
```

Потому что все операции поэлементные. Используйте `a.dot(a)` или `np.dot(a, a)`.

Альтернативно можно использовать `np.matrix` вместо `np.array`, но это не рекомендуется:

- Если использовать оба, то будет путаница в операциях (типизация-то динамическая).
- Некоторые функции сторонних библиотек могут вернуть `array`, даже если получили на вход `matrix`.
- Поэлементные операции не работают (хотя умножать на число можно).
- `*` и `/` ведут себя по-разному.

# Задание в классе №1

Обязательное:

- 1 Установите NumPy (в составе набора SciPy): [scipy.org/install.html](http://scipy.org/install.html).
  - Для Ubuntu всё лежит в репозиториях.
  - Для Mac надо поставить специальную сборку Python, в которую всё включено.
  - Для Windows надо сделать один из пунктов:
    - `pip install numpy`
    - Поставить собранные пакеты от Christoph Gohlke (обратите внимание на версию Python — 3.3, 3.4, 3.5).
    - Поставить специальную сборку Python.
- 2 Откройте руководство:  
[docs.scipy.org/doc/numpy-dev/user/quickstart.html](http://docs.scipy.org/doc/numpy-dev/user/quickstart.html).
- 3 Перемножьте две матрицы при помощи NumPy и покажите код.

## Задание в классе №1

Дополнительное:

- 4 Перемножьте две такие же матрицы при помощи циклов `for` без `NumPy`.
- 5 Уменьшите количество циклов `for` до двух.
- 6 Замерьте время работы двух функций при разных  $n$  при помощи модуля `timeit`: `print(timeit.Timer(some_func).timeit(1))`.
- 7 Поделитесь разницей в скорости с друзьями.

## Набираем классы

Если логическая сущность состоит из нескольких кусочков данных, они все лежат в одном объекте:

```
class BinaryExpression:
    def __init__(self, left, op, right):
        self.left = left
        self.op = op
        self.right = right

    def validate(self):
        self.left.validate()
        assert self.op == '+'
        self.right.validate()
```

```
expr = BinaryExpression(None, '+', None)
expr.validate() # fails
```

## Классы по-змеиному

- Первый параметр у методов объекта (передается неявно, в отличие от Java/C++) — это объект, над которым совершается операция.
- Обычно первый параметр называют `self`
- `__init__` — конструктор.
- Можно переопределить метод `__str__`, чтобы работал `print(expr)`.
- Также бывает `__cmp__`, `__eq__` и другие «magic methods»
- Всё публичное; приватное принято просто не вызывать. Можно начать метод с `_`, чтобы подчеркнуть приватность.
- Типизация утиная, никакого строгого «интерфейса» не задать.



## Стандартные проблемы

```
class Foo:
    x=[]
foo = Foo()
bar = Foo()
print(foo.x, bar.x)
foo.x.append(1)
print(foo.x, bar.x)
```

Как с аргументами по умолчанию в функциях:

```
def foo(x=[]):
    x.append(1)
    print(x)
foo()
foo()
```

Поле x было одно на все экземпляры Foo.

## Задание в классе №2

Напишите несколько программ, которые читают со стандартного входа строки по одной и после каждого чтения выводят текущую строку, если:

- 1 Она имела чётный номер.
- 2 Она была длины 3.
- 3 Она представляла собой целое число.
- 4 Она имела чётный номер среди строк длины 3, являющихся целыми числами.

Запрещается хранить предыдущие введённые строки.

После этого сохраните код в отдельный файл и выполните следующее задание.

## Задание в классе №3

- Создайте и используйте класс «Printer» с методом `consume(str)`, который печатает строку `str`.
- Создайте и используйте класс «FilterLengthIs3» с конструктором `__init__(consumer)` и методом `consume(str)`. При вызове `consumer(str)` на строке длины три должен быть вызван метод `consumer.consume(str)`.
- Создайте и используйте аналогичный класс «FilterIsNumber».
- Создайте и используйте аналогичный класс «DropOdd».
- Зацените чёткое разделение ответственности между написанными классами по сравнению с кодом без классов.

# Работаем и спрашиваем!