Бронников Егор ПМ-1901
Рослая Ирина ПМ-1901

Оптимизация Python кода

0) Предварительная работа

0. Зависимости

In [210]:

```
import copy
import warnings
import numpy as np
import pandas as pd
from typing import Union
from random import random
from functools import wraps
from time import time
```

1. Создание данных

In [473]:

```
generate_data_A = lambda n: [[random() for _ in range(n)] for _ in range(n)]
```

In [474]:

```
generate_data_f = lambda n: [random() for _ in range(n)]
```

2. Любимые декораторы 😍 💖

In [157]:

```
def timeit_deco(iters = 1000000):
    def timeit_inner(function):
        @wraps(function)
        def inner(*args, **kwargs):
            start = time()
            result = function(*args, **kwargs)
            for _ in range(iters-1):
                function(*args, **kwargs)
            end = time()
            print(f"Time of {function.__name__}: {end-start} sec, {iters} loops")
            return result
        return inner
    return timeit_inner
```

1) Что мы будем оптимизировать

1. Метод Гаусса с частичным выбором ведущего элемента

Алгоритмы решения СЛАУ:

1. Метод Гаусса с частичным выбором ведущего элемента
2. Метод наискорейшего спуска

2) Метод Гаусса с частичным выбором ведущего элемента

0) Предварительная работа

Входные данные:

In [86]:

```
A = [[1.00, 0.17, -0.25, 0.54],
     [0.47, 1.00, 0.67, -0.32],
     [-0.11, 0.35, 1.00, -0.74],
     [0.55, 0.43, 0.36, 1.00]]
```

In [3]:

```
f = [0.3, 0.5, 0.7, 0.9]
```

In [233]:

```
rand_A = generate_data_A(100)
np_rand_A = np.matrix(rand_A, dtype=np.dtype(np.float64))
```

In [234]:

```
rand_f = generate_data_f(100)
np_rand_f = np.array(rand_f, dtype=np.dtype(np.float64))
```

1) Чистный Python

In [480]:

```
def gaussian_elimination_clear(A_arg: list, f_arg: list) -> list:
    A, f = copy.deepcopy(A_arg), copy.deepcopy(f_arg)
    for i in range(len(A)):
        column = list(map(abs, [row[i] for row in A][i:]))
        if max(column) == 0.:
            warnings.warn("Determinant equals 0")
            return
        max_row = max(range(len(column)), key=column.__getitem__)
        if max_row != 0:
            pos_max = max_row + i
            A[i], A[pos_max] = A[pos_max], A[i]
            f[i], f[pos_max] = f[pos_max], f[i]
        for j in range(i+1, len(A)):
            coef = -(A[j][i]/A[i][i])
            A[j] = [coef * A[i][k] + A[j][k] for k in range(len(A[i]))]
            f[j] = coef * f[i] + f[j]
    n = len(f)
    x = [0 for _ in range(len(f))]
    x[n-1] = f[n-1]/A[n-1][n-1]
    for i in range(n-2, -1, -1):
        sum_elem = sum(A[i][j] * x[j] for j in range(i+1, n))
        x[i] = (f[i] - sum_elem)/A[i][i]
    return x
```

In [481]:

```
gaussian_elimination_clear_verified = timeit_deco()(gaussian_elimination_c
lear)
gaussian_elimination_clear_verified(A, f)
```

```
Time of gaussian_elimination_clear: 26.323821783065796 sec, 1000
000 loops
```

Out[481]:

```
[0.10057872052525962,
 0.22566664974693562,
 0.2609990851979843,
 0.35010508759299724]
```

In [221]:

```
gaussian_elimination_clear_big_data = timeit_deco(1000)(gaussian_eliminati
on_clear)
gaussian_elimination_clear_big_data(rand_A, rand_f)
```

```
Time of gaussian_elimination_clear: 51.97058844566345 sec, 1000
loops
```

Out[221]:

```
[-0.34254770293648085,
 -0.4090914662877756,
 -1.2078727120286685,
 0.4861769955535396,
 -0.007577820719772972,
 -0.43446790366799587,
 -0.5014430016320889,
 0.44025060933862636,
 -0.08662577500040863,
 -0.506134318445409,
 0.03401775562770983,
 -0.033529251830273755,
 1.2292013559726096,
 0.6797586935311742
```

```
0.6797586955311742,
-1.1035810689558565,
-0.42959678440625965,
-0.6789154029728012,
0.07790126393375325,
0.05723676712724389,
0.002729254363397237,
-0.00986275683007534,
-0.8499926194643741,
0.2873363490649973,
0.33551499966598447,
0.6796454141763357,
-0.12376289330421134,
0.343117896709248,
0.14372495364941684,
0.6291132790035244,
0.5313384924405881,
0.5266293025467892,
-1.0396877329964838,
-0.16114840030619001,
-0.058164741646716964,
0.14960522671079973,
0.03460090380298253,
-0.6713101081457877,
-0.2994890951410481,
0.9608903722247057,
-0.393624319891645,
0.9758310181202099,
-1.022813164588696,
0.7989852744760086,
-0.34016752759125163,
-0.30280394481499867,
0.32812582561424425,
1.005253674811869,
-0.36243108806318464,
0.007424022418919655,
0.15551031645501792,
0.4005947477772976,
-0.07270458408938334,
0.6055909630750664,
-0.3235282648609819,
-0.19063656472336218,
-0.4096973107048753,
0.5007163770052809,
0.8610051045045208,
0.41340626498492977,
-1.143734716602805З,
-0.42975132922616416,
0.09258802257074306,
0.004909064504391222,
0.6344763422705886,
-0.3379394790789619,
0.3415776845443371,
-0.2796160773524794,
-1.0129971863622405,
-0.3200787548458613,
0.2647426313253064,
-0.3881165202316851,
0.31148887776736467,
0.03074674659704424,
-0.9621187074546367
```

```
 0.9621187074340307,
 1.2606288630109233,
 0.4480076360032881З,
 0.40941039593859463,
 -0.2244014408338056,
 -0.2133259917475554,
 -0.4400254536302573,
 0.04402941178843598,
 -0.02999593130554015,
 -0.15694137685151804,
 1.5636929883486994,
 -0.5543503967261902,
 -1.2282494036817797,
 -0.35882429675740063,
 0.6841834519371304,
 -0.7277546966086514,
 -0.012556068255794214,
 -0.25729881433380436,
 0.66979261222428,
 0.9258305522690313,
 0.3259177290799854,
 0.2524206182407445,
 0.9454695351039355,
 -0.6817373083961092,
 1.2507564751467068,
 -0.26256073365597116,
 -0.6042642805992405]
```

2) Щепото4ка numpy

Входные данные:

In [355]:

```
np_A = np.matrix(A, dtype=np.dtype(np.float64))
```

In [356]:

```
np_f = np.array(f, dtype=np.dtype(np.float64))
```

In [181]:

```
def gaussian_elimination_numpy(A_arg: np.matrix, f_arg: Union[np.matrix, n
p.array]) -> Union[np.matrix, np.array]:
    A, f = np.copy(A_arg), np.copy(f_arg)
    for i in range(len(A)):
        column = np.abs(A[i:, i])
        leading_elem = np.max(column)
        if leading_elem == 0.:
            warnings.warn("Determinant equals 0")
            return
        if np.where(column == leading_elem)[0][0] != 0:
            pos_max = column.argmax() + i
            A[[i, pos_max]] = A[[pos_max, i]]
            f[[i, pos_max]] = f[[pos_max, i]]
        for j in range(i+1, len(A)):
            coef = -(A[j, i]/A[i, i])
```

```
                A[j] = coef * A[i] + A[j]
                f[j] = coef * f[i] + f[j]
    n = f.shape[0]
    X = np.zeros(shape=f.shape)
    X[n-1] = f[n-1]/A[n-1, n-1]
    for i in range(n-2, -1, -1):
        sum_elem = sum(A[i, j] * X[j] for j in range(i+1, n))
        X[i] = (f[i] - sum_elem)/A[i, i]
    return X
```

In [250]:

```
gaussian_elimination_numpy_verified = timeit_deco()(gaussian_elimination_n
umpy)
gaussian_elimination_numpy_verified(np_A, np_f)
```

```
Time of gaussian_elimination_numpy: 62.90274977684021 sec, 10000
00 loops
```

Out[250]:

```
array([ 0.44088855, -0.36303099,  1.16679833,  0.39356722])
```

In [242]:

```
gaussian_elimination_numpy_big_data = timeit_deco(1000)(gaussian_eliminati
on_numpy)
gaussian_elimination_numpy_big_data(np_rand_A, np_rand_f)
```

```
Time of gaussian_elimination_numpy: 16.054409503936768 sec, 1000
loops
```

Out[242]:

```
array([ -6.71934594,   4.86748184,   4.74851857,  -4.55670187,
        -9.26257797,   5.70422765,   9.75789351,  -0.36814302,
        -2.31680848,  -2.07590735,   0.53913773,   4.21163372,
         1.79567533,   0.81628661,   5.03272543,  -0.74871559,
        -3.59662896,  -3.57160479,   6.95928885,  -1.12440183,
        -7.69842843,   0.65383318,   2.31393969,  -6.22804152,
        -6.98756728,  -3.28159678,   3.78432265,  -1.41930681,
         0.81708834,   0.38541865,  -0.03044053,  -2.98377972,
         5.04834318,  -2.37470309,  -0.8756495 ,   4.89024304,
         4.18296896,   2.27176869,  -0.4241246 ,  -6.21783257,
        -0.77512913,   6.22064111,   2.58895972,  -1.77014225,
        -0.73372351,  -7.30187689,  -1.66223131,  -5.50675426,
         2.5221906 ,  -4.49054631,  -2.31933879,  -1.26746577,
         1.84213797,   3.74180786,   1.63875098,   7.15595634,
        -4.82883895,  -1.40693299,  -0.61363231,  -2.32234089,
        -1.21860884,   0.85656897,   2.39290447,  -1.15844858,
         1.90983769,  -4.38752418,  -0.9054872 ,   1.21453081,
         1.01554976,   0.09492481,  -1.31765631,  -1.14094686,
         1.31422363,   6.57181202,   0.94442572,   2.66560377,
        -0.16744522,   6.35694704,  -0.29176067,   6.1245141 ,
       -10.25400986,   4.53359295,   1.27364564,  -2.19681137,
        -1.55093503,   5.66537775,  -3.8360484 ,  -3.05828103,
         3.98764015,   3.05101524,   1.38816828, -11.21079643,
         1.5998147 ,   0.82869714,   2.56626117,   6.3465645 ,
         2.62223904,  -0.97872403,  -5.27942814,  -4.00921628])
```

3) Чистый Python + Cython

In [16]:

```
%load_ext cython
```

In [202]:

```
%%cython -a
import copy
import warnings


def gaussian_elimination_clear_cython(A_arg: list, f_arg: list) -> list:
    A, f = copy.deepcopy(A_arg), copy.deepcopy(f_arg)
    for i in range(len(A)):
        column = [row[i] for row in A][i:]
        if max(column) == 0.:
            warnings.warn("Determinant equals 0")
            return
        max_row = max(range(len(column)), key=column.__getitem__)
        if max_row != 0:
            pos_max = max_row + i
            A[i], A[pos_max] = A[pos_max], A[i]
            f[i], f[pos_max] = f[pos_max], f[i]
        for j in range(i+1, len(A)):
            coef = -(A[j][i]/A[i][i])
            A[j] = [coef * A[i][k] + A[j][k] for k in range(len(A[i]))]
            f[j] = coef * f[i] + f[j]
    n = len(f)
    x = [0 for _ in range(len(f))]
    x[n-1] = f[n-1]/A[n-1][n-1]
    for i in range(n-2, -1, -1):
        sum_elem = sum(A[i][j] * x[j] for j in range(i+1, n))
        x[i] = (f[i] - sum_elem)/A[i][i]
    return x
```

Out[202]:

Generated by Cython 0.29.22

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

```
+01: import copy
+02: import warnings
 03: 
 04: 
+05: def gaussian_elimination_clear_cython(A_arg: list, f_arg: list) -> list:
+06:     A, f = copy.deepcopy(A_arg), copy.deepcopy(f_arg)
+07:     for i in range(len(A)):
+08:         column = [row[i] for row in A][i:]
+09:         if max(column) == 0.:
+10:             warnings.warn("Determinant equals 0")
+11:             return
```

```
+11:             return
+12:         max_row = max(range(len(column)), key=column.__getitem__)
+13:         if max_row != 0:
+14:             pos_max = max_row + i
+15:             A[i], A[pos_max] = A[pos_max], A[i]
+16:             f[i], f[pos_max] = f[pos_max], f[i]
+17:         for j in range(i+1, len(A)):
+18:             coef = -(A[j][i]/A[i][i])
+19:             A[j] = [coef * A[i][k] + A[j][k] for k in range(len(A[i]))]
+20:             f[j] = coef * f[i] + f[j]
+21:     n = len(f)
+22:     x = [0 for _ in range(len(f))]
+23:     x[n-1] = f[n-1]/A[n-1][n-1]
+24:     for i in range(n-2, -1, -1):
+25:         sum_elem = sum(A[i][j] * x[j] for j in range(i+1, n))
+26:         x[i] = (f[i] - sum_elem)/A[i][i]
+27:     return x
```

In [203]:

```
gaussian_elimination_clear_cython_verified = timeit_deco()(gaussian_elimination_clear_cython)
gaussian_elimination_clear_cython_verified(A, f)
```

```
Time of gaussian_elimination_clear_cython: 18.776877403259277 sec, 1000000 loops
```

Out[203]:

```
[0.4408885508918321,
 -0.36303099013644724,
 1.166798332275979,
 0.3935672231488123]
```

In [223]:

```
gaussian_elimination_clear_cython_big_data = timeit_deco(1000)(gaussian_elimination_clear_cython)
gaussian_elimination_clear_cython_big_data(rand_A, rand_f)
```

```
Time of gaussian_elimination_clear_cython: 22.52452850341797 sec, 1000 loops
```

Out[223]:

```
[-0.34254770293648085,
 -0.4090914662877756,
 -1.2078727120286685,
 0.4861769955535396,
 -0.007577820719772972,
 -0.4344679036679958 7,
 -0.5014430016320889,
 0.4402506093386263 6,
 -0.08662577500040863,
-0.506134318445409,
0.03401775562770983,
-0.033529251830273755,
1.2292013559726096,
0.6797586935311742,
-1.1035810689558565,
-0.42959678440625965,
-0.6789154029728012,
0.07790126393375325,
0.05723676712724389,
0.002729254363397237,
-0.00986275683007534,
-0.8499926194643741,
0.2873363490649973,
0.33551499966598447,
0.6796454141763357,
-0.12376289330421134,
0.343117896709248,
0.14372495364941684,
0.6291132790035244,
0.5313384924405881,
0.5266293025467892,
-1.0396877329964838,
-0.16114840030619001,
-0.058164741646716964,
0.14960522671079973,
0.03460090380298253,
-0.6713101081457877,
-0.2994890951410481,
0.9608903722247057,
-0.393624319891645,
0.9758310181202099,
-1.022813164588696,
0.7989852744760086,
-0.34016752759125163,
-0.30280394481499867,
0.32812582561424425,
1.005253674811869,
-0.36243108806318464,
0.007424022418919655,
0.15551031645501792,
0.4005947477772976,
-0.07270458408938334,
0.6055909630750664,
-0.3235282648609819,
-0.19063656472336218,
-0.4096973107048753,
0.5007163770052809,
0.8610051045045208,
0.41340626498492977,
-1.143734716602805З,
-0.42975132922616416,
0.09258802257074306,
0.004909064504391222,
0.6344763422705886,
-0.3379394790789619,
0.3415776845443371,
-0.2796160773524794,
-1.0129971863622405,
-0.3200787548458613
```

```
 -0.3200787548458015,
 0.2647426313253064,
 -0.3881165202316851,
 0.3114888777673646 7,
 0.0307467465970442 4,
 -0.9621187074546367,
 1.2606288630109233,
 0.4480076360032881 3,
 0.4094103959385946 3,
 -0.2244014408338056,
 -0.2133259917475554,
 -0.440025453630257 3,
 0.0440294117884359 8,
 -0.0299959313055401 5,
 -0.1569413768515180 4,
 1.5636929883486994,
 -0.5543503967261902,
 -1.2282494036817797,
 -0.3588242967574006 3,
 0.6841834519371304,
 -0.7277546966086514,
 -0.0125560682557942 14,
 -0.2572988143338043 6,
 0.66979261222428,
 0.9258305522690313,
 0.3259177290799854,
 0.2524206182407445,
 0.9454695351039355,
 -0.6817373083961092,
 1.2507564751467068,
 -0.2625607336559711 6,
 -0.6042642805992405]
```

4) Щепото4ка numpy + Cython

In [470]:

```
%%cython -a
import warnings
import numpy as np
from typing import Union


def gaussian_elimination_numpy_cython(A_arg: np.matrix, f_arg: Union[np.ma
trix, np.array]) -> Union[np.matrix, np.array]:
    A, f = np.copy(A_arg), np.copy(f_arg)
    for i in range(len(A)):
        column = np.abs(A[i:, i])
        leading_elem = np.max(column)
        if leading_elem == 0.:
            warnings.warn("Determinant equals 0")
            return
        if np.where(column == leading_elem)[0][0] != 0:
            pos_max = column.argmax() + i
            A[[i, pos_max]] = A[[pos_max, i]]
            f[[i, pos_max]] = f[[pos_max, i]]
        for j in range(i+1, len(A)):
            coef = -(A[j, i]/A[i, i])
            A[j] = coef * A[i] + A[j]
            f[j] = coef * f[i] + f[j]
    n = f.shape[0]
    X = np.zeros(shape=f.shape)
    X[n-1] = f[n-1]/A[n-1, n-1]
    for i in range(n-2, -1, -1):
        sum_elem = sum(A[i, j] * X[j] for j in range(i+1, n))
        X[i] = (f[i] - sum_elem)/A[i, i]
    return X
```

Out[470]:

Generated by Cython 0.29.22

Yellow lines hint at Python interaction.
Click on a line that starts with a " + " to see the C code that Cython generated for it.

```
+01: import warnings
+02: import numpy as np
+03: from typing import Union
 04: 
 05: 
+06: def gaussian_elimination_numpy_cython(A_arg: np.matrix, f_arg: Union[np.matrix, np.array]) -> Union[np.matrix, np.array]:
+07:     A, f = np.copy(A_arg), np.copy(f_arg)
+08:     for i in range(len(A)):
+09:         column = np.abs(A[i:, i])
+10:         leading_elem = np.max(column)
+11:         if leading_elem == 0.:
+12:             warnings.warn("Determinant equals 0")
+13:             return
+14:         if np.where(column == leading_elem)[0][0] != 0:
+15:             pos_max = column.argmax() + i
+16:             A[[i, pos_max]] = A[[pos_max, i]]
+17:             f[[i, pos_max]] = f[[pos_max, i]]
+18:         for j in range(i+1, len(A)):
+19:             coef = -(A[j, i]/A[i, i])
+20:             A[j] = coef * A[i] + A[j]
+21:             f[j] = coef * f[i] + f[j]
+22:     n = f.shape[0]
+23:     X = np.zeros(shape=f.shape)
+24:     X[n-1] = f[n-1]/A[n-1, n-1]
+25:     for i in range(n-2, -1, -1):
+26:         sum_elem = sum(A[i, j] * X[j] for j in range(i+1, n))
+27:         X[i] = (f[i] - sum_elem)/A[i, i]
+28:     return X
```

In [249]:

```
gaussian_elimination_numpy_cython_verified = timeit_deco()(gaussian_elimin
ation_numpy_cython)
gaussian_elimination_numpy_cython_verified(np_A, np_f)
```

```
Time of gaussian_elimination_numpy_cython: 64.43942546844482 se
c, 1000000 loops
```

Out[249]:

```
array([ 0.44088855, -0.36303099,  1.16679833,  0.39356722])
```

In [241]:

```
gaussian_elimination_numpy_cython_big_data = timeit_deco(1000)(gaussian_el
imination_numpy_cython)
gaussian_elimination_numpy_cython_big_data(np_rand_A, np_rand_f)
```

```
Time of gaussian_elimination_numpy_cython: 15.401480674743652 se
c, 1000 loops
```

Out[241]:

```
array([ -6.71934594,   4.86748184,   4.74851857,  -4.55670187,
        -9.26257797,   5.70422765,   9.75789351,  -0.36814302,
        -2.31680848,  -2.07590735,   0.53913773,   4.21163372,
         1.79567533,   0.81628661,   5.03272543,  -0.74871559,
        -3.59662896,  -3.57160479,   6.95928885,  -1.12440183,
        -7.69842843,   0.65383318,   2.31393969,  -6.22804152,
        -6.98756728,  -3.28159678,   3.78432265,  -1.41930681,
         0.81708834,   0.38541865,  -0.03044053,  -2.98377972,
         5.04834318,  -2.37470309,  -0.8756495 ,   4.89024304,
         4.18296896,   2.27176869,  -0.4241246 ,  -6.21783257,
        -0.77512913,   6.22064111,   2.58895972,  -1.77014225,
        -0.73372351,  -7.30187689,  -1.66223131,  -5.50675426,
         2.5221906 ,  -4.49054631,  -2.31933879,  -1.26746577,
         1.84213797,   3.74180786,   1.63875098,   7.15595634,
        -4.82883895,  -1.40693299,  -0.61363231,  -2.32234089,
        -1.21860884,   0.85656897,   2.39290447,  -1.15844858,
         1.90983769,  -4.38752418,  -0.9054872 ,   1.21453081,
         1.01554976,   0.09492481,  -1.31765631,  -1.14094686,
         1.31422363,   6.57181202,   0.94442572,   2.66560377,
        -0.16744522,   6.35694704,  -0.29176067,   6.1245141 ,
       -10.25400986,   4.53359295,   1.27364564,  -2.19681137,
        -1.55093503,   5.66537775,  -3.8360484 ,  -3.05828103,
         3.98764015,   3.05101524,   1.38816828, -11.21079643,
         1.5998147 ,   0.82869714,   2.56626117,   6.3465645 ,
         2.62223904,  -0.97872403,  -5.27942814,  -4.00921628])
```

5) Итоги

In [251]:

```
pd.read_json("gaussian_verified_data.json").sort_values(by="time (sec)").s
tyle.set_properties(**{'font-size': '16pt'})
```

Out[251]:

	name	loops	time (sec)
2	Clear Python + Cython	1000000	18.776877

0	Clear Python	1000000	25.075198
1	numpy	1000000	62.902750
3	numpy + Cython	1000000	64.439425

In [252]:

```
pd.read_json("gaussian_big_data.json").sort_values(by="time (sec)").style.
set_properties(**{'font-size': '16pt'})
```

Out[252]:

	name	loops	time (sec)
3	numpy + Cython	1000	15.401481
1	numpy	1000	16.054410
2	Clear Python + Cython	1000	22.524529
0	Clear Python	1000	51.970588

3) Метод наискорейшего спуска

0. Предварительная работа

Входные данные:

In [422]:

```
A = [[4.33, -1.12, -1.08, 1.14],
     [-1.12, 4.33, 0.24, -1.22],
     [-1.08, 0.24, 7.21, -3.22],
     [1.14, -1.22, -3.22, 5.43]]

np_A = np.matrix(A, dtype=np.dtype(np.float64))
```

In [466]:

```
f = [0.3, 0.5, 0.7, 0.9]

np_f = np.array(f, dtype=np.dtype(np.float64))
```

Умножение матриц

In [467]:

```
def matmul(mat1, mat2):
    return [[sum(mat1[i][k] * mat2[k][j] for k in range(len(mat1[0])))
             for j in range(len(mat2[0]))]
             for i in range(len(mat1))]
```

Скалярное произведение

In [454]:

```
def dot(mat1, mat2):
    return sum(x[0] * y[0] for x, y in zip(mat1, mat2))
```

1) Чистый Python

In [462]:

```
def steepest_descent_method_clear(A_arg: list, f_arg: list, K_max: int) ->
list:
    A, f = copy.deepcopy(A_arg), copy.deepcopy(f_arg)
    x = [[0] for _ in range(len(f))]
    for k in range(K_max):
        mul = matmul(A, x)
        r = [[f[i] - mul[i][0]] for i in range(len(f))]
        alpha = dot(r, r)/dot(matmul(A, r), r)
        x = [[x[i][0] + alpha * r[i][0]] for i in range(len(x))]
    return x
```

In [463]:

```
steepest_descent_method_clear_verified = timeit_deco()(steepest_descent_me
thod_clear)
steepest_descent_method_clear_verified(A, f, 10)
```

```
Time of steepest_descent_method_clear: 142.95242381095886 sec, 1
000000 loops
```

Out[463]:

```
[[0.10056807326359224],
 [0.2256523011705975],
 [0.2609940631555897],
 [0.3500720527873754]]
```

2) numpy

In [461]:

```
def steepest_descent_method_numpy(A_arg: np.matrix, f_arg: np.array, K_max
: int) -> np.array:
    A, f = np.copy(A_arg), np.copy(f_arg)
    if not np.all(np.linalg.eigvals(A) > 0):
        warnings.warn("Matrix is not positive definite")
        return
    elif not np.allclose(A, A.T):
        warnings.warn("Matrix is not symmetric")
        return
    elif K_max < 0:
        warnings.warn("The number of iterations cannot be negative")
        return
    x = np.zeros(f.shape, dtype=np.dtype(np.float64))
    for k in range(K_max):
        r = np.squeeze(np.asarray(f - np.matmul(A, x)))
        alpha = (np.dot(r, r)/np.dot(np.matmul(A, r), r)).item(0)
        x = x + alpha * r
    return x
```

In [443]:

```
steepest_descent_method_numpy_verified = timeit_deco()(steepest_descent_me
thod_numpy)
steepest_descent_method_numpy_verified(np_A, np_f, 10)
```

```
Time of steepest_descent_method_numpy: 154.923086643219 sec, 100
0000 loops
```

Out[443]:

```
array([0.10056807, 0.2256523 , 0.26099406, 0.35007205])
```

3) Чистый Python + Cython

In [465]:

```
%%cython -a
import copy
from __main__ import matmul
from __main__ import dot


def steepest_descent_method_clear_cython(A_arg: list, f_arg: list, K_max:
int) -> list:
    A, f = copy.deepcopy(A_arg), copy.deepcopy(f_arg)
    x = [[0] for _ in range(len(f))]
    for k in range(K_max):
        mul = matmul(A, x)
        r = [[f[i] - mul[i][0]] for i in range(len(f))]
        alpha = dot(r, r)/dot(matmul(A, r), r)
        x = [[x[i][0] + alpha * r[i][0]] for i in range(len(x))]
    return x
```

Out[465]:

Generated by Cython 0.29.22

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that
Cython generated for it.

```
+01: import copy
+02: from __main__ import matmul
+03: from __main__ import dot
 04:
 05:
+06: def steepest_descent_method_clear_cython(A_arg: list, f_arg
: list, K_max: int) -> list:
+07:     A, f = copy.deepcopy(A_arg), copy.deepcopy(f_arg)
+08:     x = [[0] for _ in range(len(f))]
+09:     for k in range(K_max):
+10:         mul = matmul(A, x)
+11:         r = [[f[i] - mul[i][0]] for i in range(len(f))]
+12:         alpha = dot(r, r)/dot(matmul(A, r), r)
+13:         x = [[x[i][0] + alpha * r[i][0]] for i in range(len
(x))]
+14:     return x
```

In [468]:

```
steepest_descent_method_clear_cython_verified = timeit_deco()(steepest_des
cent_method_clear_cython)
steepest_descent_method_clear_cython_verified(A, f, 10)
```

```
Time of steepest_descent_method_clear_cython: 128.13575959205627
sec, 1000000 loops
```

Out[468]:

```
[[0.10056807326359224],
 [0.2256523011705975],
 [0.2609940631555897],
 [0.3500720527873754]]
```

4) numpy + Cython

In [469]:

```
%%cython -a
import warnings
import numpy as np
from typing import Union


def steepest_descent_method_numpy_cython(A_arg: np.matrix, f_arg: np.array
, K_max: int) -> np.array:
    A, f = np.copy(A_arg), np.copy(f_arg)
    if not np.all(np.linalg.eigvals(A) > 0):
        warnings.warn("Matrix is not positive definite")
        return
    elif not np.allclose(A, A.T):
        warnings.warn("Matrix is not symmetric")
        return
    elif K_max < 0:
        warnings.warn("The number of iterations cannot be negative")
        return
    x = np.zeros(f.shape, dtype=np.dtype(np.float64))
    for k in range(K_max):
        r = np.squeeze(np.asarray(f - np.matmul(A, x)))
        alpha = (np.dot(r, r)/np.dot(np.matmul(A, r), r)).item(0)
        x = x + alpha * r
    return x
```

Out[469]:

Generated by Cython 0.29.22

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

```
+01: import warnings
+02: import numpy as np
+03: from typing import Union
 04: 
 05: 
+06: def steepest_descent_method_numpy_cython(A_arg: np.matrix, f_arg: np.array, K_max: int) -> np.array:
+07:     A, f = np.copy(A_arg), np.copy(f_arg)
+08:     if not np.all(np.linalg.eigvals(A) > 0):
+09:         warnings.warn("Matrix is not positive definite")
+10:         return
+11:     elif not np.allclose(A, A.T):
+12:         warnings.warn("Matrix is not symmetric")
+13:         return
+14:     elif K_max < 0:
+15:         warnings.warn("The number of iterations cannot be negative")
+16:         return
+17:     x = np.zeros(f.shape, dtype=np.dtype(np.float64))
+18:     for k in range(K_max):
+19:         r = np.squeeze(np.asarray(f - np.matmul(A, x)))
+20:         alpha = (np.dot(r, r)/np.dot(np.matmul(A, r), r)).item(0)
+21:         x = x + alpha * r
+22:     return x
```

In [472]:

```
steepest_descent_method_numpy_cython_verified = timeit_deco()(steepest_descent_method_numpy_cython)
steepest_descent_method_numpy_cython_verified(A, f, 10)
```

```
Time of steepest_descent_method_numpy_cython: 158.60338521003723 sec, 1000000 loops
```

Out[472]:

```
array([0.10056807, 0.2256523 , 0.26099406, 0.35007205])
```

5) Итоги

In [484]:

```
pd.read_json("steepest_descent_verified_data.json").sort_values(by="time (sec)").style.set_properties(**{'font-size': '16pt'})
```

Out[484]:

	name	loops	time (sec)
2	Clear Python + Cython	1000000	128.135760
0	Clear Python	1000000	142.952424
1	numpy	1000000	154.923087
3	numpy + Cython	1000000	158.603385