

Accelerated Python



Python

Быстро пишется, долго работает

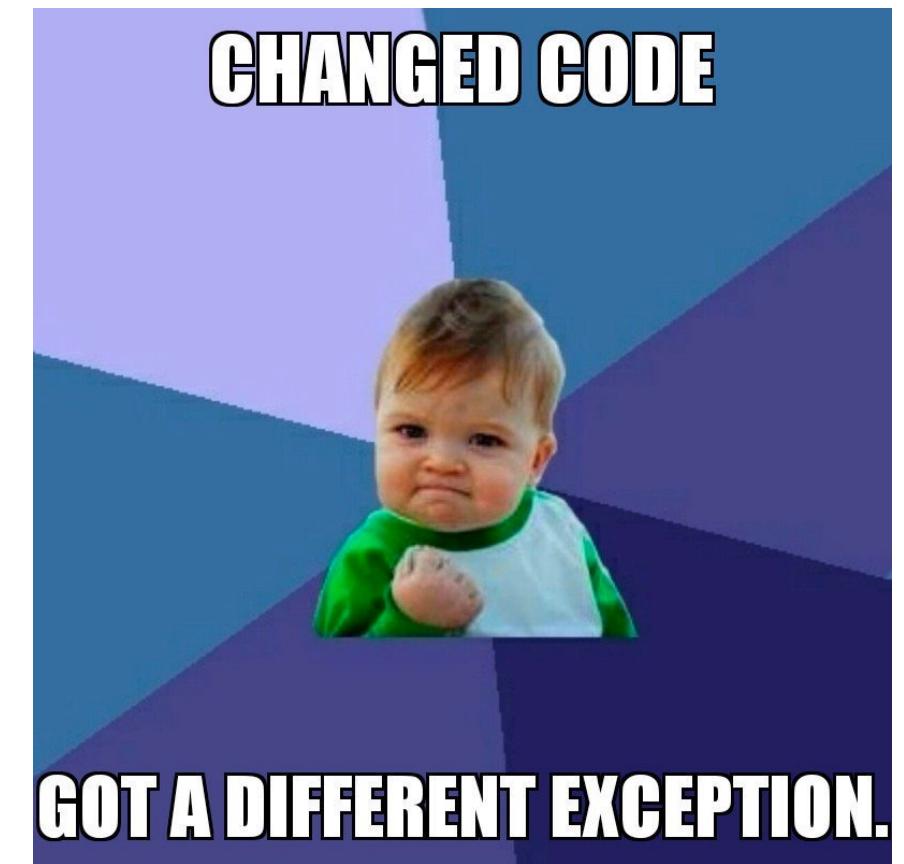


Python

Быстро пишется, долго работает



И не сложно учится



C/C++

Долго пишется, быстро работает



C/C++

Долго пишется, быстро работает

И...



Tries to learn C++



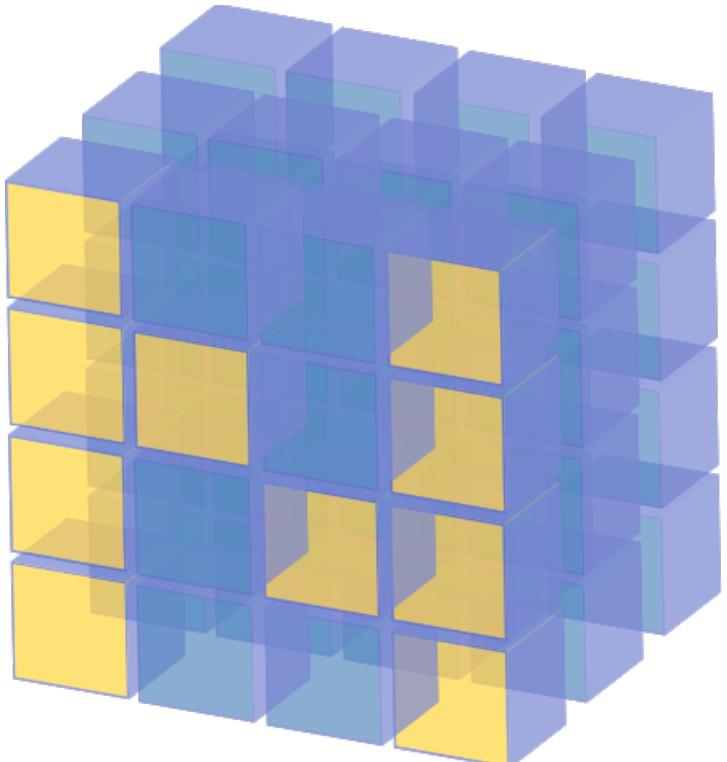
Тогда как объединить?

Скорость плюсов

Удобство питона



Использовать питону 😊



NumPy

При подсчёте фичей

```
%%timeit
stat1 = 0
stat2 = 0
stat3 = 0
for i, elem in enumerate(a_list):
    if elem % 17 == 0:
        stat1 += i
    if elem % 19 == 0:
        stat2 += i
    if elem % 28 == 0:
        stat3 += i
```

executed in 12.0s, finished 22:11:47 2019-04-12

14.7 ms ± 272 µs per loop (mean ± std.

При подсчёте фичей

```
%%timeit
stat1 = 0
stat2 = 0
stat3 = 0
for i, elem in enumerate(a_list):
    if elem % 17 == 0:
        stat1 += i
    if elem % 19 == 0:
        stat2 += i
    if elem % 28 == 0:
        stat3 += i
```

executed in 12.0s, finished 22:11:47 2019-04-12

14.7 ms ± 272 µs per loop (mean ± std.

```
%%timeit
np.where(a_array % 17 == 0)[0].sum()
np.where(a_array % 19 == 0)[0].sum()
np.where(a_array % 28 == 0)[0].sum()
```

executed in 2.10s, finished 22:11:54 2019-04-12

2.56 ms ± 47.9 µs per loop (mean ± std. dev.

Когда нужны циклы

```
%%timeit
np.random.seed(42)
values = []
for _ in range(10000):
    values.append((np.random.randint(0, 2, size=10000) * 2 - 1).mean())
executed in 6.01s, finished 22:22:42 2019-04-12
```

750 ms ± 13.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

Когда нужны циклы

```
%%timeit
np.random.seed(42)
values = []
for _ in range(10000):
    values.append((np.random.randint(0, 2, size=10000) * 2 - 1).mean())
executed in 6.01s, finished 22:22:42 2019-04-12
```

750 ms ± 13.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

```
%%timeit
np.random.seed(42)
values = (np.random.randint(0, 2, size=(10000, 10000)) * 2 - 1).mean(axis=1)
executed in 8.49s, finished 22:20:06 2019-04-12
```

1.06 s ± 15 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

Когда нужны циклы

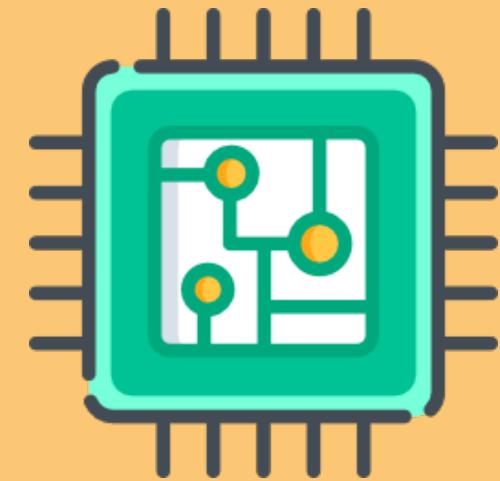
```
%%timeit
np.random.seed(42)
values = []
for _ in range(10):
    values.extend((np.random.randint(0, 2, size=(1000, 10000)) * 2 - 1).mean(axis=1))
executed in 6.30s, finished 22:22:17 2019-04-12
```

779 ms ± 14.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

```
%%timeit
np.random.seed(42)
values = []
for _ in range(100):
    values.extend((np.random.randint(0, 2, size=(100, 10000)) * 2 - 1).mean(axis=1))
executed in 5.11s, finished 22:22:22 2019-04-12
```

627 ms ± 10.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

2. C++ в Python



Python.h

Обычно что это такое знают только те, кто словил ошибки с этим файлом

Python.h

Обычно что это такое знают только те, кто словил ошибки с этим файлом

```
#include "Python.h"

static PyObject *
keywdarg_parrot(PyObject *self, PyObject *args, PyObject *keywds)
{
    int voltage;
    char *state = "a stiff";
    char *action = "voom";
    char *type = "Norwegian Blue";

    static char *kwlist[] = {"voltage", "state", "action", "type", NULL};

    if (!PyArg_ParseTupleAndKeywords(args, keywds, "i|sss",
                                    kwlist,
                                    &voltage, &state, &action, &type))
        return NULL;

    printf("-- This parrot wouldn't %s if you put %i Volts through it.\n",
           action, voltage);
    printf("-- Lovely plumage, the %s -- It's %s!\n", type, state);

    Py_INCREF(Py_None);

    return Py_None;
}

static PyMethodDef keywdarg_methods[] = {
    /* The cast of the function is necessary since PyCFunction values
     * only take two PyObject* parameters, and keywdarg_parrot() takes
     * three.
     */
    {"parrot", (PyCFunction)keywdarg_parrot, METH_VARARGS | METH_KEYWORDS,
     "Print a lovely skit to standard output."},
    {NULL, NULL, 0, NULL} /* sentinel */
};
```

Python.h

Писать такой код – сомнительное удовольствие, в этом случае спасёт...



В чём суть обёрток

1. Пишем код на удобном языке
2. Переводим код в **C/C++**
3. Компилируем
4. ...
5. PROFIT

Boost.Python

[Ссылка на проект](#)

Following C/C++ tradition, let's start with the "hello, world". A C++ Function:

```
char const* greet()
{
    return "hello, world";
}
```

can be exposed to Python by writing a Boost.Python wrapper:

```
#include <boost/python.hpp>

BOOST_PYTHON_MODULE(hello_ext)
{
    using namespace boost::python;
    def("greet", greet);
}
```

That's it. We're done. We can now build this as a shared library. The resulting DLL is now visible to Python. Here's a sample Python session:

```
>>> import hello_ext
>>> print hello_ext.greet()
hello, world
```

Pybind11

[Ссылка на проект](#)

```
#include <pybind11/pybind11.h>

int add(int i, int j) {
    return i + j;
}

PYBIND11_MODULE(example, m) {
    m.doc() = "pybind11 example plugin"; // optional module docstring

    m.def("add", &add, "A function which adds two numbers");
}
```

А в чём разница?



А в чём разница?



Да ни в чём! Просто у
Boost-а зависимость от
boost, а ещё Pybind11 более
лаконично реализован

CMake

```
CMakeLists.txt ×

1 cmake_minimum_required(VERSION 3.5)
2 project(taxi_ml_cxx)
3
4 set(CMAKE_CXX_STANDARD 11)
5
6 if(NOT DEFINED CMAKE_LIBRARY_OUTPUT_DIRECTORY)
7     set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_SOURCE_DIR})
8 endif()
9
10 set(THIRD_PARTY_DIRS ${PROJECT_SOURCE_DIR}/third_party)
11
12 add_compile_options ("-fexceptions" "-g")
13 # build type specific
14 if (CMAKE_BUILD_TYPE MATCHES "Debug" OR CMAKE_BUILD_TYPE MATCHES "Test")
15     add_compile_options ("-O0")
16     add_definitions ("-D_CONSOLE_LOGGER -DTEST_MOCK")
17 else ()
18     add_compile_options ("-O3")
19     add_definitions ("-DNDEBUG")
20 endif ()
21
22 find_package(PythonInterp 2.7 REQUIRED)
23 find_package(PythonLibs 2.7 REQUIRED)
24
25 set(CMAKE_POSITION_INDEPENDENT_CODE ON)
```

А как это сделать частью пакета?

```
class CMakeBuild(build_ext):
    def run(self):
        try:
            subprocess.check_output(['cmake', '--version'])
        except OSError:
            raise RuntimeError(
                "CMake must be installed to build the following extensions: " +
                ", ".join(e.name for e in self.extensions))

        for ext in self.extensions:
            self.build_extension(ext)

    def build_extension(self, ext):
        ext_dir = os.path.abspath(
            os.path.dirname(self.get_ext_fullpath(ext.name)))
        cfg = 'Debug' if self.debug else 'Release'
        cmake_args = ['-DCMAKE_LIBRARY_OUTPUT_DIRECTORY=' + ext_dir,
                      '-DPYTHON_EXECUTABLE=' + sys.executable,
                      '-DCMAKE_BUILD_TYPE=' + cfg]
        build_args = ['--config', cfg, '--', '-j2']
        env = os.environ.copy()
        env['CXXFLAGS'] = '{} -DVERSION_INFO=\"{}\"'.format(
            env.get('CXXFLAGS', ''),
            self.distribution.get_version())
        )

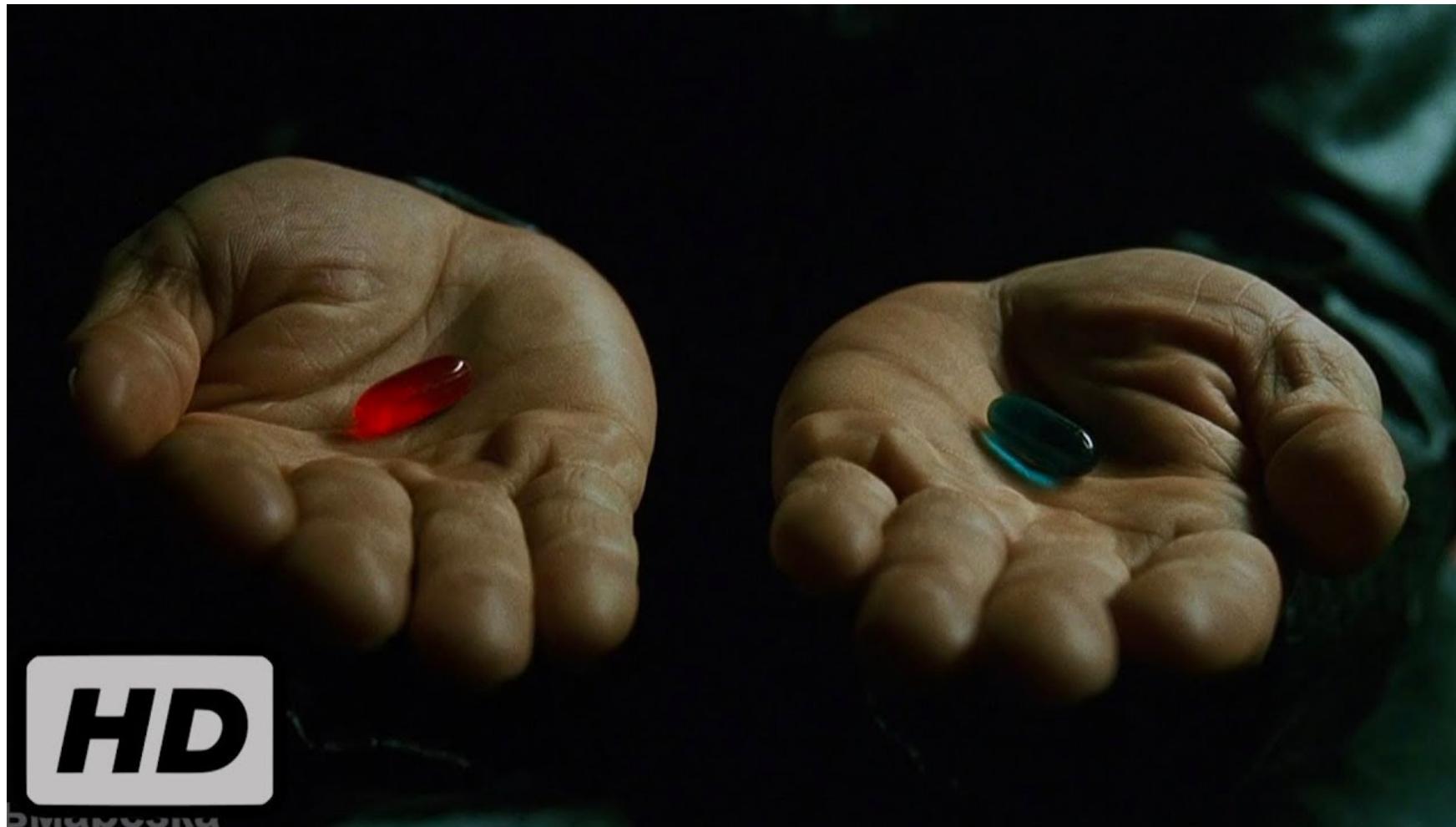
        if not os.path.exists(self.build_temp):
            os.makedirs(self.build_temp)
        subprocess.check_call(['cmake', ext.source_dir] + cmake_args,
                             cwd=self.build_temp, env=env)
        subprocess.check_call(['cmake', '--build', '.'] + build_args,
                             cwd=self.build_temp)

        for dir_path, _, _ in os.walk(os.path.join(ext_dir, ext.name)):
            path = os.path.join(dir_path, '__init__.py')
            if not os.path.exists(path):
                open(path, 'w').close()
print() # Add an empty line for cleaner output
```

```
common_setup_kwargs = dict(
    name='yandex_taxi_ml_cxx',
    version=taxi_ml_cxx.__version__,
    description='C++ wrappers',
    packages=find_packages(include=('taxi_ml_cxx*',)))

if os.environ.get('TAXI_ML_CXX_USE_BUILT'):
    setup(
        package_data={'': ['*.so']},
        **common_setup_kwargs
    )
else:
    setup(
        ext_modules=[CMakeExtension('taxi_ml_cxx')],
        cmdclass=dict(build_ext=CMakeBuild),
        **common_setup_kwargs
    )
```

Альтернативный путь



Cython



В чём суть cython

1. Пишем код на языке, похожем на питон
2. Он переводится на С
3. Компилируем
4. ...
5. PROFIT

Cython

Ссылка на проект

```
1 # python_functions.py
2 # -----
3 import math
4
5 def f(x):
6     return math.exp(-(x ** 2))
7
8 def integrate_f(a, b, N):
9     s = 0
10    dx = (b - a) / N
11    for i in range(N):
12        s += f(a + i * dx)
13    return s * dx
14
```

```
1 # cython_functions.pyx
2 # -----
3 from libc cimport math
4
5 cdef double f(double x):
6     return math.exp(-(x ** 2))
7
8 def integrate_f(double a, double b, int N):
9     cdef double s = 0
10    cdef double dx = (b - a) / N
11    cdef int i
12    for i in range(N):
13        s += f(a + i * dx)
14    return s * dx
15
```

Cython: до

```
import numpy as np

cimport numpy as np
cimport cython

@cython.boundscheck(False)
@cython.wraparound(False)
def memory_efficient_inner1d(
    cython.floating[:, :] fst_arr,
    cython.integral[:] fst_indices,
    cython.floating[:, :] snd_arr,
    cython.integral[:] snd_indices
):
    cdef int size = fst_indices.shape[0]
    cdef int width = fst_arr.shape[1]
    cdef np.ndarray[np.float64_t] result = np.zeros(size, dtype=np.float64)
    cdef int i, j, fst_index, snd_index
    cdef double sum
    for i in range(size):
        sum = 0
        fst_index = fst_indices[i]
        snd_index = snd_indices[i]
        for j in range(width):
            sum += fst_arr[fst_index, j] * snd_arr[snd_index, j]
        result[i] = sum

    return result
```

Cython: после

```
28350 if (likely(PyLong_CheckExact(b))) {  
28351     #if CYTHON_USE_PYLONG_INTERNALS  
28352     const digit_t digits = ((PyLongObject*)b)->ob_digit;  
28353     const Py_ssize_t size = Py_SIZE(b);  
28354     if (likely(__Pyx_sst_abs(size) <= 1)) {  
28355         ival = likely(size) ? digits[0] : 0;  
28356         if (size == -1) ival = -ival;  
28357         return ival;  
28358     } else {  
28359         switch (size) {  
28360             case 2:  
28361                 if (8 * sizeof(Py_ssize_t) > 2 * PyLong_SHIFT) {  
28362                     return (Py_ssize_t) (((size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]);  
28363                     break;  
28364             case -2:  
28365                 if (8 * sizeof(Py_ssize_t) > 2 * PyLong_SHIFT) {  
28366                     return -(Py_ssize_t) (((size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]);  
28367                     break;  
28368             case 3:  
28369                 if (8 * sizeof(Py_ssize_t) > 3 * PyLong_SHIFT) {  
28370                     return (Py_ssize_t) (((((size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]);  
28371                     break;  
28372             case -3:  
28373                 if (8 * sizeof(Py_ssize_t) > 3 * PyLong_SHIFT) {  
28374                     return -(Py_ssize_t) (((((size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]) << PyLong_SHIFT) | (size_t)digits[0]);  
28375                     break;  
28376             case 4:  
28377                 if (8 * sizeof(Py_ssize_t) > 4 * PyLong_SHIFT) {  
28378                     return (Py_ssize_t) (((((((size_t)digits[3]) << PyLong_SHIFT) | (size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]) << PyLong_SHIFT);  
28379                     break;  
28380             case -4:  
28381                 if (8 * sizeof(Py_ssize_t) > 4 * PyLong_SHIFT) {  
28382                     return -(Py_ssize_t) (((((((size_t)digits[3]) << PyLong_SHIFT) | (size_t)digits[2]) << PyLong_SHIFT) | (size_t)digits[1]);  
28383                     break;  
28384         }  
28385     }  
28386     #endif  
28387     return PyLong_AsSsize_t(b);  
28388 }  
28389 x = PyNumber_Index(b);  
28390 if (!x) return -1;  
28391 ival = PyInt_AsSsize_t(x);  
28392 Py_DECREF(x);  
28393 return ival;  
28394  
28395 static CYTHON_INLINE PyObject * __Pyx_PyBool_FromLong(long b) {  
28396     return b ? __Pyx_NewRef(Py_True) : __Pyx_NewRef(Py_False);  
28397 }  
28398 static CYTHON_INLINE PyObject * __Pyx_PyInt_FromSize_t(size_t ival) {  
28399     return PyInt_FromSize_t(ival);  
28400 }
```

```
28381  
28382  
28383  
28384  
28385  
28386  
28387  
28388  
28389  
28390  
28391  
28392  
28393  
28394  
28395  
28396  
28397  
28398  
28399  
28400  
28401  
28402  
28403  
28404  
28405  
28406  
28407  
28408  
28409  
28410  
28411  
28412  
28413  
28414  
28415  
28416  
28417  
28418  
28419  
28420  
28421  
28422  
28423  
28424  
28425  
28426  
28427  
28428  
28429  
28430  
28431  
28432  
28433  
28434  
28435  
28436  
28437  
28438  
28439  
28440  
28441  
28442  
28443  
28444  
28445  
28446  
28447  
28448  
28449  
28450  
28451  
28452  
28453  
28454  
28455  
28456  
28457  
28458  
28459  
28460  
28461  
28462  
28463  
28464  
28465  
28466  
28467  
28468  
28469  
28470  
28471  
28472  
28473  
28474  
28475  
28476  
28477  
28478  
28479  
28480  
28481  
28482  
28483  
28484  
28485  
28486  
28487  
28488  
28489  
28490  
28491  
28492  
28493  
28494  
28495  
28496  
28497  
28498  
28499  
28500  
28501  
28502  
28503  
28504  
28505  
28506  
28507  
28508  
28509  
28510  
28511  
28512  
28513  
28514  
28515  
28516  
28517  
28518  
28519  
28520  
28521  
28522  
28523  
28524  
28525  
28526  
28527  
28528  
28529  
28530  
28531  
28532  
28533  
28534  
28535  
28536  
28537  
28538  
28539  
28540  
28541  
28542  
28543  
28544  
28545  
28546  
28547  
28548  
28549  
28550  
28551  
28552  
28553  
28554  
28555  
28556  
28557  
28558  
28559  
28560  
28561  
28562  
28563  
28564  
28565  
28566  
28567  
28568  
28569  
28570  
28571  
28572  
28573  
28574  
28575  
28576  
28577  
28578  
28579  
28580  
28581  
28582  
28583  
28584  
28585  
28586  
28587  
28588  
28589  
28590  
28591  
28592  
28593  
28594  
28595  
28596  
28597  
28598  
28599  
28600  
28601  
28602  
28603  
28604  
28605  
28606  
28607  
28608  
28609  
28610  
28611  
28612  
28613  
28614  
28615  
28616  
28617  
28618  
28619  
28620  
28621  
28622  
28623  
28624  
28625  
28626  
28627  
28628  
28629  
28630  
28631  
28632  
28633  
28634  
28635  
28636  
28637  
28638  
28639  
28640  
28641  
28642  
28643  
28644  
28645  
28646  
28647  
28648  
28649  
28650  
28651  
28652  
28653  
28654  
28655  
28656  
28657  
28658  
28659  
28660  
28661  
28662  
28663  
28664  
28665  
28666  
28667  
28668  
28669  
28670  
28671  
28672  
28673  
28674  
28675  
28676  
28677  
28678  
28679  
28680  
28681  
28682  
28683  
28684  
28685  
28686  
28687  
28688  
28689  
28690  
28691  
28692  
28693  
28694  
28695  
28696  
28697  
28698  
28699  
28700  
28701  
28702  
28703  
28704  
28705  
28706  
28707  
28708  
28709  
28710  
28711  
28712  
28713  
28714  
28715  
28716  
28717  
28718  
28719  
28720  
28721  
28722  
28723  
28724  
28725  
28726  
28727  
28728  
28729  
28730  
28731  
28732  
28733  
28734  
28735  
28736  
28737  
28738  
28739  
28740  
28741  
28742  
28743  
28744  
28745  
28746  
28747  
28748  
28749  
28750  
28751  
28752  
28753  
28754  
28755  
28756  
28757  
28758  
28759  
28760  
28761  
28762  
28763  
28764  
28765  
28766  
28767  
28768  
28769  
28770  
28771  
28772  
28773  
28774  
28775  
28776  
28777  
28778  
28779  
28780  
28781  
28782  
28783  
28784  
28785  
28786  
28787  
28788  
28789  
28790  
28791  
28792  
28793  
28794  
28795  
28796  
28797  
28798  
28799  
28800  
28801  
28802  
28803  
28804  
28805  
28806  
28807  
28808  
28809  
28810  
28811  
28812  
28813  
28814  
28815  
28816  
28817  
28818  
28819  
28820  
28821  
28822  
28823  
28824  
28825  
28826  
28827  
28828  
28829  
28830  
28831  
28832  
28833  
28834  
28835  
28836  
28837  
28838  
28839  
28840  
28841  
28842  
28843  
28844  
28845  
28846  
28847  
28848  
28849  
28850  
28851  
28852  
28853  
28854  
28855  
28856  
28857  
28858  
28859  
28860  
28861  
28862  
28863  
28864  
28865  
28866  
28867  
28868  
28869  
28870  
28871  
28872  
28873  
28874  
28875  
28876  
28877  
28878  
28879  
28880  
28881  
28882  
28883  
28884  
28885  
28886  
28887  
28888  
28889  
28890  
28891  
28892  
28893  
28894  
28895  
28896  
28897  
28898  
28899  
28900  
28901  
28902  
28903  
28904  
28905  
28906  
28907  
28908  
28909  
28910  
28911  
28912  
28913  
28914  
28915  
28916  
28917  
28918  
28919  
28920  
28921  
28922  
28923  
28924  
28925  
28926  
28927  
28928  
28929  
28930  
28931  
28932  
28933  
28934  
28935  
28936  
28937  
28938  
28939  
28940  
28941  
28942  
28943  
28944  
28945  
28946  
28947  
28948  
28949  
28950  
28951  
28952  
28953  
28954  
28955  
28956  
28957  
28958  
28959  
28960  
28961  
28962  
28963  
28964  
28965  
28966  
28967  
28968  
28969  
28970  
28971  
28972  
28973  
28974  
28975  
28976  
28977  
28978  
28979  
28980  
28981  
28982  
28983  
28984  
28985  
28986  
28987  
28988  
28989  
28990  
28991  
28992  
28993  
28994  
28995  
28996  
28997  
28998  
28999  
29000  
29001  
29002  
29003  
29004  
29005  
29006  
29007  
29008  
29009  
29010  
29011  
29012  
29013  
29014  
29015  
29016  
29017  
29018  
29019  
29020  
29021  
29022  
29023  
29024  
29025  
29026  
29027  
29028  
29029  
29030  
29031  
29032  
29033  
29034  
29035  
29036  
29037  
29038  
29039  
29040  
29041  
29042  
29043  
29044  
29045  
29046  
29047  
29048  
29049  
29050  
29051  
29052  
29053  
29054  
29055  
29056  
29057  
29058  
29059  
29060  
29061  
29062  
29063  
29064  
29065  
29066  
29067  
29068  
29069  
29070  
29071  
29072  
29073  
29074  
29075  
29076  
29077  
29078  
29079  
29080  
29081  
29082  
29083  
29084  
29085  
29086  
29087  
29088  
29089  
29090  
29091  
29092  
29093  
29094  
29095  
29096  
29097  
29098  
29099  
29100  
29101  
29102  
29103  
29104  
29105  
29106  
29107  
29108  
29109  
29110  
29111  
29112  
29113  
29114  
29115  
29116  
29117  
29118  
29119  
29120  
29121  
29122  
29123  
29124  
29125  
29126  
29127  
29128  
29129  
29130  
29131  
29132  
29133  
29134  
29135  
29136  
29137  
29138  
29139  
29140  
29141  
29142  
29143  
29144  
29145  
29146  
29147  
29148  
29149  
29150  
29151  
29152  
29153  
29154  
29155  
29156  
29157  
29158  
29159  
29160  
29161  
29162  
29163  
29164  
29165  
29166  
29167  
29168  
29169  
29170  
29171  
29172  
29173  
29174  
29175  
29176  
29177  
29178  
29179  
29180  
29181  
29182  
29183  
29184  
29185  
29186  
29187  
29188  
29189  
29190  
29191  
29192  
29193  
29194  
29195  
29196  
29197  
29198  
29199  
29200  
29201  
29202  
29203  
29204  
29205  
29206  
29207  
29208  
29209  
29210  
29211  
29212  
29213  
29214  
29215  
29216  
29217  
29218  
29219  
29220  
29221  
29222  
29223  
29224  
29225  
29226  
29227  
29228  
29229  
29230  
29231  
29232  
29233  
29234  
29235  
29236  
29237  
29238  
29239  
29240  
29241  
29242  
29243  
29244  
29245  
29246  
29247  
29248  
29249  
29250  
29251  
29252  
29253  
29254  
29255  
29256  
29257  
29258  
29259  
29260  
29261  
29262  
29263  
29264  
29265  
29266  
29267  
29268  
29269  
29270  
29271  
29272  
29273  
29274  
29275  
29276  
29277  
29278  
29279  
29280  
29281  
29282  
29283  
29284  
29285  
29286  
29287  
29288  
29289  
29290  
29291  
29292  
29293  
29294  
29295  
29296  
29297  
29298  
29299  
29300  
29301  
29302  
29303  
29304  
29305  
29306  
29307  
29308  
29309  
29310  
29311  
29312  
29313  
29314  
29315  
29316  
29317  
29318  
29319  
29320  
29321  
29322  
29323  
29324  
29325  
29326  
29327  
29328  
29329  
29330  
29331  
29332  
29333  
29334  
29335  
29336  
29337  
29338  
29339  
29340  
29341  
29342  
29343  
29344  
29345  
29346  
29347  
29348  
29349  
29350  
29351  
29352  
29353  
29354  
29355  
29356  
29357  
29358  
29359  
29360  
29361  
29362  
29363  
29364  
29365  
29366  
29367  
29368  
29369  
29370  
29371  
29372  
29373  
29374  
29375  
29376  
29377  
29378  
29379  
29380  
29381  
29382  
29383  
29384  
29385  
29386  
29387  
29388  
29389  
29390  
29391  
29392  
29393  
29394  
29395  
29396  
29397  
29398  
29399  
29400  
29401  
29402  
29403  
29404  
29405  
29406  
29407  
29408  
29409  
29410  
29411  
29412  
29413  
29414  
29415  
29416  
29417  
29418  
29419  
29420  
29421  
29422  
29423  
29424  
29425  
29426  
29427  
29428  
29429  
29430  
29431  
29432  
29433  
29434  
29435  
29436  
29437  
29438  
29439  
29440  
29441  
29442  
29443  
29444  
29445  
29446  
29447  
29448  
29449  
29450  
29451  
29452  
29453  
29454  
29455  
29456  
29457  
29458  
29459  
29460  
29461  
29462  
29463  
29464  
29465  
29466  
29467  
29468  
29469  
29470  
29471  
29472  
29473  
29474  
29475  
29476  
29477  
29478  
29479  
29480  
29481  
29482  
29483  
29484  
29485  
29486  
29487  
29488  
29489  
29490  
29491  
29492  
29493  
29494  
29495  
29496  
29497  
29498  
29499  
29500  
29501  
29502  
29503  
29504  
29505  
29506  
29507  
29508  
29509  
29510  
29511  
29512  
29513  
29514  
29515  
29516  
29517  
29518  
29519  
29520  
29521  
29522  
29523  
29524  
29525  
29526  
29527  
29528  
29529  
29530  
29531  
29532  
29533  
29534  
29535  
29536  
29537  
29538  
29539  
29540  
29541  
29542  
29543  
29544  
29545  
29546  
29547  
29548  
29549  
29550  
29551  
29552  
29553  
29554  
29555  
29556  
29557  
29558  
29559  
29560  
29561  
29562  
29563  
29564  
29565  
29566  
29567  
29568  
29569  
29570  
29571  
29572  
29573  
29574  
29575  
29576  
29577  
29578  
29579  
29580  
29581  
29582  
29583  
29584  
29585  
29586  
29587  
29588  
29589  
29590  
29591  
29592  
29593  
29594  
29595  
29596  
29597  
29598  
29599  
29600  
29601  
29602  
29603  
29604  
29605  
29606  
29607  
29608  
29609  
29610  
29611  
29612  
29613  
29614  
29615  
29616  
29617  
29618  
29619  
29620  
29621  
29622  
29623  
29624  
29625  
29626  
29627  
29628  
29629  
29630  
29631  
29632  
29633  
29634  
29635  
29636  
29637  
29638  
29639  
29640  
29641  
29642  
29643  
29644  
29645  
29646  
29647  
29648  
29649  
29650  
29651  
29652  
29653  
29654  
29655  
29656  
29657  
29658  
29659  
29660  
29661  
29662  
29663  
29664  
29665  
29666  
29667  
29668  
29669  
29670  
29671  
29672  
29673  
29674  
29675  
29676  
29677  
29678  
29679  
29680  
29681  
29682  
29683  
29684  
29685  
29686  
29687  
29688  
29689  
29690  
29691  
29692  
29693  
29694  
29695  
29696  
29697  
29698  
29699  
29700  
29701  
29702  
29703  
29704  
29705  
29706  
29707  
29708  
29709  
29710  
29711  
29712  
29713  
29714  
29715  
29716  
29717  
29718  
29719  
29720  
29721  
29722  
29723  
29724  
29725  
29726  
29727  
29728  
29729  
29730  
29731  
29732  
29733  
29734  
29735  
29736  
29737  
29738  
29739  
29740  
29741  
29742  
29743  
29744  
29745  
29746  
29747  
29748  
29749  
29750  
29751  
29752  
29753  
29754  
29755  
29756  
29757  
29758  
29759  
29760  
29761  
29762  
29763  
29764  
29765  
29766  
29767  
29768  
29769  
29770  
29771  
29772  
29773  
29774  
29775  
29776  
29777  
29778  
29779  
29780  
29781  
29782  
29783  
29784  
29785  
29786  
29787  
29788  
29789  
29790  
29791  
29792  
29793  
29794  
29795  
2979
```

А как это сделать частью пакета?

```
1  #!/usr/bin/env python
2  from __future__ import print_function
3
4  try:
5      from Cython.Build import build_ext
6      import numpy as np
7      np_include_dirs = np.get_include()
8  except ImportError:
9      from setuptools.command.build_ext import build_ext
10     np_include_dirs = ''
11
12 import platform
13
14 from distutils.core import setup
15 from distutils.extension import Extension
16 from setuptools import find_packages
17
18 included_packages = ('pyartm*',)
19
20 is_windows = platform.system() == 'Windows'
21 with open('requirements.txt', 'r') as req_file:
22     requirements = [
23         line.strip()
24         for line in req_file
25         if line.strip() and (is_windows or 'mysqlclient' not in line)
26     ]
27
28 setup(
29     setup_requires=requirements[:2],
30     ext_modules=[Extension(
31         "package.calculations.inner_product.cy_impl",
32         sources=["package/calculations/inner_product/cy_impl.pyx"],
33         include_dirs=[np_include_dirs],
34     )],
35     cmdclass={'build_ext': build_ext},
36     install_requires=requirements,
37     packages=find_packages(include=included_packages)
38 )
39
```

Альтернативный путь 2



Numba



В чём суть патча

1. Код питона компилируется в байт код (*.pyc файлы)
2. Берём байт код функции
3. Перегоняем его в машинный код и заменяем реализацию
4. ...
5. PROFIT

Numba

- Ссылка на проект

```
from numba import jit
import numpy as np
import time

x = np.arange(100).reshape(10, 10)

@jit(nopython=True)
def go_fast(a): # Function is compiled and runs in machine code
    trace = 0
    for i in range(a.shape[0]):
        trace += np.tanh(a[i, i])
    return a + trace

# DO NOT REPORT THIS... COMPILATION TIME IS INCLUDED IN THE EXECUTION TIME!
start = time.time()
go_fast(x)
end = time.time()
print("Elapsed (with compilation) = %s" % (end - start))

# NOW THE FUNCTION IS COMPILED, RE-TIME IT EXECUTING FROM CACHE
start = time.time()
go_fast(x)
end = time.time()
print("Elapsed (after compilation) = %s" % (end - start))
```

This, for example prints:

```
Elapsed (with compilation) = 0.33030009269714355
Elapsed (after compilation) = 6.67572021484375e-06
```

В итоге

1. Почти всегда можно обойтись питру
2. Если надо одну простую функцию ускорить, и она работает с числами или массивами, то Numba в помощь
3. Если пишете библиотеку с большими встройками C/C++, то лучше использовать Cython
4. Если хотите писать на плюсах, ~~то пишите, питон тут причём~~, используя мощь проектов на новых стандартах, то это проще на pybind11 или Boost.Python