Использование Cython для ускорения вычислений в Python

Николаев Сергей Владимирович

ВМК МГУ

7 ноября 2016 г.

Что такое Cython?

- Компилятор Cython переводит программу, написанную на языках Python или Cython, в более эффективный код на C.
- Язык Cython является надстройкой Python, поддерживающей вызовы функций C и присвоение переменным типов данных языка C.

Плюсы Cython:

- Совмещает читабельность Python кода с быстротой языка C.
- Позволяет напрямую вызывать библиотечные функции C.
- Эффективно работает с большими данными, в том числе используя numpy массивы.
- Позволяет писать расширения на C для Python также легко, как и на самом Python.

Путь кода к импортируемому модулю

Код на Cython должен быть скомпилирован перед запуском.
Это происходит в два этапа:

1. Сначала Cython компилирует .pyx файл (содержащий нашу программу) в .c файл.
2. Затем C компилирует .c файл в модуль, который мы можем импортировать.

Порядок действий следующий:

1. Написать Cython код, сохранив его с расширением .pyx
2. Написать скрипт setup.py
3. Выполнить команду
   `python setup.py build_ext --inplace`

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
  name = 'Hello world app',
  ext_modules = cythonize("hello.pyx"),
)
```

С помощью IPython Notebook можно писать и вызывать Cython код «на месте». Для этого нужно загрузить расширение Cython в IPython:

```
%load_ext Cython
```

Затем нужно пометить клетку маркером.

```
%%cython

cdef int a = 0
for i in range(10):
    a += 1
print a
```

Пример использования Cython

Рассмотрим использование Cython на простом примере.
«Чистый» Python:

```
def f(x):
    return x**2 - x

def integrate_f(a, b, N):
    s = 0
    dx = (b - a) / N
    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```

Вычисление $f(0, 10, 1000000)$ заняло 1.41s.

Пример использования Cython

Скомпилируем это в Cython:

```
%%cython

def f(x):
    return x**2 - x

def integrate_f(a, b, N):
    s = 0
    dx = (b - a) / N
    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```

Вычисление $f(0, 10, 1000000)$ заняло 769ms.

Применим статическую типизацию к переменным:

```
%%cython

def f(double x):
    return x**2 - x

def integrate_f(double a, double b, int N):
    cdef int i
    cdef double s, dx
    s = 0
    dx = (b - a) / N
    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```

Вычисление $f(0, 10, 1000000)$ заняло 149ms. Это связано с тем, что цикл был скомпилирован в чистый C код.

Пример использования Cython

Вызов функций в Python может быть дорогим. Например, в прошлом примере постоянно происходила конвертация из C double в Python float. Изменим прототип функции **f**:

```
%%cython

cdef double f(double x):
    return x**2 - x
```

Вычисление $f(0, 10, 1000000)$ заняло 9ms.

Пример использования Cython

Посмотрим, как справляется с той же задачей numpy.

```
x = np.linspace(0, 10, 1000000)
(x**2 - x).sum() / 100000
```

Данное выражение вычисляется за 35ms. Таким образом, с помощью Cython можно написать более эффективный код по сравнению с numpy.

Классы в Cython

- Cython поддерживает объектно-ориентированное программирование, например классы Python.
- Кроме того, Cython позволяет создавать свои классы. Они как правило быстрее и эффективнее по памяти в сравнении с классами Python.
- Главное различие состоит в том, что классы Cython используют структуры C для хранения своих полей и методов, в то время как в Python используются словари.

Классы в Cython

```
%%cython

cdef class Function:
    cpdef double evaluate(self, double x):
        return 0

cdef class myFunction(Function):
    cpdef double evaluate(self, double x):
        return x**2 - x

def integrate_f(Function f, double a, double b, int N):
    cdef int i
    cdef double s, dx
    s = 0
    dx = (b - a) / N
    for i in range(N):
        s += f.evaluate(a + i * dx)
    return s * dx
```

Литература

- Сайт Cython
  http://cython.org/
- Документация Cython
  http://cython.readthedocs.io/en/latest/