Профилирование в Python для ускорения вычислений

Лунин Дмитрий

317 группа ВМК

27 октября 2015 г.

Что такое профилирование, и зачем оно нужно

The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.
Дональд Кнут

- Профилирование – сбор статистики во время работы программы.
- Далее будем рассматривать измерение времени работы, но можно измерять и использование памяти, попадания в кеши и т.д.

Простой способ

1. Несколько раз запустить и остановить программу.
2. Посмотреть на стек вызовов.
3. Если в них часто встречается какая-то функция – скорее всего проблема в ней.

Плюсы.

- Прост в использовании
- Работает лучше, чем вы думаете.
- Не требует никаких инструментов (кроме стека вызовов)

Минусы.

- Возможна ошибка.
- Работает только если есть 1-2 "узких места".

В IPython Notebook

```
%prun -s cumulative <expr>
```

```
 ncalls  tottime  percall  cumtime  percall filename:lineno(function
      1    0.008    0.008   16.944   16.944 <string>:1(<module>)
      1    0.000    0.000   16.936   16.936 <ipython-input-12-e618ba
      1    0.000    0.000   16.457   16.457 {map}
     10    0.009    0.001   16.457    1.646 <ipython-input-12-e618ba
     10    0.834    0.083   16.448    1.645 <ipython-input-13-892ce5
     10    4.977    0.498   10.763    1.076 <ipython-input-13-892ce5
     10    0.000    0.000    5.786    0.579 fromnumeric.py:1631(sum)
     10    0.000    0.000    5.786    0.579 _methods.py:31(_sum)
     10    5.786    0.579    5.786    0.579 {method 'reduce' of 'num
     10    3.725    0.373    3.725    0.373 {method 'argsort' of 'nu
```

cProfile

Профилирование всего скрипта

```
python -m cProfile [-o output_file] [-s sort_order]
    myscript.py
```

Профилирование одной команды

```
import cProfile
import re
cProfile.run('re.compile("foo|bar")')
```

Line profiler
В IPython Notebook

```
%load_ext line_profiler
%lprun -f <function> <expression>
%lprun -f my_func my_func(1, 2, 'text')
```

```
      Time  Per Hit   % Time  Line Contents
==============================================
                                @wraps(f)
                                def modified_f(*args, **kwargs):
         3      3.0      0.0        X = args[chunked_arg]
         1      1.0      0.0        pre_args = args[:chunked_arg]
         1      1.0      0.0        post_args = args[chunked_arg+1:]

         5      5.0      0.0        real_n_chunks = min(n_chunks, X.sh
       111    111.0      0.0        X_chunks = np.array_split(X, real_

         2      2.0      0.0        if not parallel:
  16251324 16251324.0     97.2            result_chunks = list(map(lam
                                    else:
```

Line profiler

В программах на Питоне можно использовать декоратор из line_profile:

```
@profile
def test():
    data = np.random.random((5000, 100))
    u, s, v = linalg.svd(data)
    pca = np.dot(u[:, :10], data)
    results = fastica(pca.T, whiten=False)
```

Вызываем профайлер:

```
kernprof.py -l -v demo.py
```

Ссылки

- Stackoverflow – о простом способе
  http://stackoverflow.com/questions/375913/what-can-i-use-to-profile-c-code-in-linux
- Документация к cProfile
  https://docs.python.org/3.5/library/profile.html
- Профилирование в IPython http://pynash.org/2013/03/06/timing-and-profiling.html
- Гайд по профилированию от scipy
  http://www.scipy-lectures.org/advanced/optimizing/