ВМК

Задание 2. Векторизация вычислений с помощью библиотеки Numpy

Практикум 317 группы 2017-2018, осенний семестр

Начало выполнения задания: 13 сентября 2017 года.
Срок сдачи: **19 сентября 2017 года, 23:59.**

Решение каждой задачи должно быть описано в модуле `task_<номер задачи>.py`.

В систему anytask необходимо сдать `zip` архив, содержащий решения задач, называющийся `contest2_<фамилия студента>_<имя студента>.zip`.

Задания проверяются с помощью автоматических тестов. Количество баллов за задачу зависит от количества пройденных тестов. Задание считается засчитанным, если хотя бы одна задача проходит больше 25% тестов. Под матрицей во всех задачах понимается двумерный `numpy.array`, под вектором одномерный.

Решения всех задач должны быть полностью векторизованными (т.е. не содержать циклов и подобных конструкций)!

Замечание. В некоторых задачах полезно написать сначала невекторизованный вариант реализации, чтобы использовать его для тестирования векторизованного. Сравните по скорости невекторизованный и векторизованный варианты решения (для замерения скорости используйте модуль `time` или magic-command в jyputer notebook `%timeit`). Какой оказался быстрее?

1. Написать функцию `get_nonzero_diag_product(X)`, которая подсчитывает произведение ненулевых элементов на диагонали прямоугольной матрицы. Если все элементы на диагонали нулевые, функция должна вернуть None.

   Пример:

   ```
   >>> get_nonzero_diag_product(np.array([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]))
   3
   ```

2. Написать функцию `get_elements_by_indexes(X, i_indexes, j_indexes)`, принимающую матрицу `X` и два вектора длины `N`: `i_indexes` и `j_indexes`. Функция должна возвращать вектор `np.array([X[i_indexes[0], j_indexes[0]], ..., X[i_indexes[N-1], j_indexes[N-1]])`

   Пример:

   ```
   >>> get_elements_by_indexes(np.array([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]),
                               np.array([1, 2]), np.array([2, 2]))
   array([2, 3])
   ```

3. Написать функцию `replace_nan_to_means(X)`, принимающую матрицу `X`. Функция должна вернуть копию матрицы `X`, в которой все значения `nan` в каждом столбце заменены на среднее арифметическое остальных элементов столбца. В случае столбца из одних `nan` необходимо заменить все элементы столбца на нули. Исходная матрица `X` должна остаться неизменной!

   Пример:

   ```
   >>> replace_nan_to_means(np.array([[1, 0, 1], [np.nan, 2, 5], [2, 0, np.nan], [3, 1, 3]]))
   array([[ 1.,  0.,  1.],
          [ 2.,  2.,  5.],
          [ 2.,  0.,  3.],
          [ 3.,  1.,  3.]])
   ```

4. Написать функцию `get_max_before_zero(x)`, возвращающую максимальный элемент в векторе `x` среди элементов, перед которыми стоит нулевой. Если подходящих элементов нет, функция должна возвращать `None`.

   Пример:

   ```
   get_max_before_zero(np.array([6, 2, 0, 3, 0, 0, 5, 7, 0]))
   5
   ```

5. Написать функцию `encode_rle(x)`, реализующую кодирование длин серий (Run-length encoding). По входному вектору `x` необходимо вернуть кортеж из двух векторов одинаковой длины. Первый содержит числа, а второй — сколько раз их нужно повторить.

   Пример:

```
>>> encode_rle(np.array([2, 2, 2, 3, 3, 3, 5, 3]))
(np.array([2, 3, 5, 3]), np.array([3, 3, 1, 1]))
```