A Tutorial on Python for High-Performance Computing

Rafael Sarmiento
ETHZürich / CSCS
CSCS-USI Summer School 2019

ETH zürich
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Outline

- The problem: The Euclidean and Cityblock distance matrices
- Vectorized code with NumPy
- Delayed execution with Dask
- Just in time compilation with Numba
- Fortran90 subroutines within Python with F2PY
- C functions within Python with CFFI

The Problem: Euclidean distance matrix

$$d_{\mathrm{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean distance matrix

$$d_{\mathrm{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

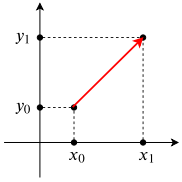
The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

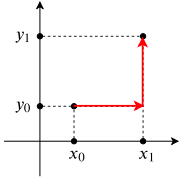
The Problem: Euclidean distance matrix

$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$

The Problem: Euclidean and Cityblock distance matrices

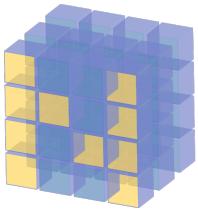


$$d_{\mathsf{e}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum (x_{1i} - y_{1i})^2 & \sum (x_{1i} - y_{2i})^2 & \dots & \sum (x_{1i} - y_{ni})^2 \\ \sum (x_{2i} - y_{1i})^2 & \sum (x_{2i} - y_{2i})^2 & \dots & \sum (x_{2i} - y_{ni})^2 \\ \dots & \dots & \dots & \dots \\ \sum (x_{ni} - y_{1i})^2 & \sum (x_{ni} - y_{2i})^2 & \dots & \sum (x_{ni} - y_{ni})^2 \end{bmatrix}$$



$$d_{\mathsf{cb}}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}, \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \dots & \dots & \dots \\ y_{n1} & y_{n2} & y_{n3} \end{bmatrix}\right) = \begin{bmatrix} \sum |x_{1i} - y_{1i}| & \sum |x_{1i} - y_{2i}| & \dots & \sum |x_{1i} - y_{ni}| \\ \sum |x_{2i} - y_{1i}| & \sum |x_{2i} - y_{2i}| & \dots & \sum |x_{2i} - y_{ni}| \\ \dots & \dots & \dots & \dots \\ \sum |x_{ni} - y_{1i}| & \sum |x_{ni} - y_{2i}| & \dots & \sum |x_{ni} - y_{ni}| \end{bmatrix}$$

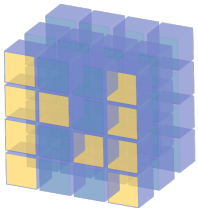
NumPy



NumPy is a Python library that adds support for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays[1].

[1] Homepage

NumPy



- `numpy.array`: a powerful N-dimensional array object
- Sophisticated functions often written in C
- Tools for integrating C/C++ and Fortran code (`Python.h` header file and F2PY)
- Linear algebra, Fourier transform, and random number capabilities

[1] Homepage

An inefficient implementation with NumPy

```
def euclidean_distance_matrix(x, y):
    num_samples = x.shape[0]
    dist_matrix = np.empty((num_samples, num_samples))
    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            diff = xi - yj
            dist_matrix[i][j] = np.dot(diff, diff)

    return dist_matrix
```

Using only vectorized operations

$$\sum_k (x_{ik} - y_{ik})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

Using only vectorized operations

$$\sum_k (x_{ik} - y_{ik})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

$\vec{x}_i \cdot \vec{y}_j \rightarrow$ `np.dot(x, y)` : Matrix product of $\{\vec{x}\}$ and $\{\vec{y}\}$

Using only vectorized operations

$$\sum_k (x_{ik} - y_{ik})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

$\vec{x}_i \cdot \vec{y}_j \rightarrow$ `np.dot(x, y)` : Matrix product of $\{\vec{x}\}$ and $\{\vec{y}\}$

$\vec{x}_i \cdot \vec{x}_i \rightarrow$ `np.einsum('ij,ij->i', x, x)` : A vector of elements $\sum_j x_{ij} x_{ij} \equiv \sum_j x_{ij}^2$

$\vec{y}_j \cdot \vec{y}_j \rightarrow$ `np.einsum('ij,ij->i', y, y)` : A vector of elements $\sum_j y_{ij} y_{ij} \equiv \sum_j y_{ij}^2$

Using only vectorized operations

$$\sum_k (x_{ik} - y_{ik})^2 = (\vec{x}_i - \vec{y}_j) \cdot (\vec{x}_i - \vec{y}_j) = \vec{x}_i \cdot \vec{x}_i + \vec{y}_j \cdot \vec{y}_j - 2\vec{x}_i \cdot \vec{y}_j$$

$\vec{x}_i \cdot \vec{y}_j \rightarrow$ `np.dot(x, y)`

$\vec{x}_i \cdot \vec{x}_i \rightarrow$ `np.einsum('ij,ij->i', x, x)[:,np.newaxis]`

$\vec{y}_j \cdot \vec{y}_j \rightarrow$ `np.einsum('ij,ij->i', y, y)[:,np.newaxis].T`

Using only vectorized operations

```
def euclidean_distance_matrix(x, y):
    x2 = np.einsum('ij,ij->i', x, x)[:,np.newaxis]
    y2 = np.einsum('ij,ij->i', y, y)[:,np.newaxis].T
    xy = np.dot(x, y.T)

    return np.abs(x2 + y2 - 2. * xy)
```

[lab] Euclidean distance matrix with NumPy

- Let's open the notebook `euclidean-distance-matrix-numpy.ipynb` and check step by step what the function `euclidean_numpy` does:
  - What's the shape of the array resulting from the `np.einsum` operation? Why?
  - What's the effect of adding a new axis to an array with `[:,np.newaxis]`?
  - What's the effect of adding a new axis to an array with `[np.newaxis,:]`?
  - What's the effect of the sum `x2 + y2` in the `euclidean_numpy` function?
  - Why is necessary to add a new axis?
- Run all cells and compare the execution times of the different approaches.
- While running the `%timeit` function calls, you may open a terminal and check the load with the command `top`.

Cityblock distance matrix

$$\sum_k |x_{ik} - y_{jk}|$$

The trick we used for the Euclidean distance matrix doesn't work here!

SciPy

The SciPy library provides many user-friendly and efficient numerical routines for operations such as numerical integration, interpolation, optimization, linear algebra and statistics. SciPy builds on the `numpy.array` *object and expands the set of mathematical functions included in NumPy*[1]*.*

[1] Homepage

SciPy

```
scipy.spatial.distance.cdist(x, y, 'cityblock')
```

SciPy

```
scipy.spatial.distance.cdist(x, y, 'cityblock')
```

`cdist` is fast but doesn't use OpenMP threads. We can easily write a distributed Cityblock distance matrix function based on `cdist` with the help of Dask.

Dask

Dask is a flexible library for parallel computing in Python. It provides dynamic task scheduling optimized for computation as well as big data collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy and Pandas to larger-than-memory or distributed environments[1].

[1] Homepage

Dask

- `dask.array` implements a subset of the NumPy array interface using blocked algorithms, cutting up the large array into chunks of small arrays.
- `dask.bag` parallelizes computations across a large collection of generic Python objects.
- `dask.dataframe` is a large parallel DataFrame composed of many smaller Pandas DataFrames which may live on disk for larger-than-memory computing on a single machine or a cluster.

[1] Homepage

Dask

- `dask.delayed` can be used to parallelize custom algorithms. It allows to create graphs with a light annotation of normal python code:

```
x = dask.delayed(myfunction1)(<args>)
y = dask.delayed(myfunction2)(<args>)
z = dask.delayed(myfunction3)(x, y)
z.compute(scheduler='threads')
```

[1] Homepage

[lab] Cityblock distance matrix with SciPy and Dask

- Let's run the notebook `cityblock-distance-matrix-scipy.dask.ipynb`.
  - While timing `cdist` check with `top` that it runs on a single thread.
  - Why is it relevant for the implementation of such distributed function that `cdist` runs on a single thread?
  - Check that when we create the list of delayed functions the execution is deferred to when `compute` is called.
- Run all cells and compare the execution times of the different approaches.
- While running the `%timeit` function calls, you may open a terminal and check with `top` that the new distributed function is runnig in multiple threads.

Numba

Numba is an open source just-in-time (JIT) compiler that translates a subset of Python and NumPy code into fast machine code[1].

[1] Homepage

Numba

- Translation of python functions to machine code at runtime using the LLVM compiler library
- Designed to be used with NumPy arrays and functions
- Options to parallelize code for CPUs and GPUs and automatic SIMD Vectorization
- Support for both NVIDIA's CUDA and AMD's ROCm driver allowing to write parallel GPU code from Python.

[1] Homepage

Numba

```
def reduce(x):
    x_sum = 0.0
    for i in range(x.shape[0]):
        x_sum += x[i]

    return x_sum
```

[1] Homepage

Numba

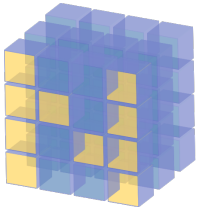
```python
import numba

@numba.jit(nopython=True)
def reduce(x):
    x_sum = 0.0
    for i in range(x.shape[0]):
        x_sum += x[i]

    return x_sum
```

[1] Homepage

[lab] Cityblock distance matrix with Numba's just-in-time compilation

- Let's run the notebook `cityblock-distance-matrix-numba.jit.ipynb`.
  - Notice that the function to be decorated with `@numba.jit` is not written in a pythonic style. Instead, with the loops it resembles more the C or Fortran styles.
  - What are the diferences between the two Numba implementations?
  - Notice that on the innermost loop we use `numba.prange` to run it in parallel.
- Run all cells and compare the execution times of the different approaches.
- While running the `%timeit` function calls, you may open a terminal and check with `top` that the decorated functions run in multiple threads.

F2PY



This is a Fortran to Python interface generator. F2PY is a part of NumPy and also is available as a standalone command line tool `f2py` *that's installed with NumPy. It facilitates creating and building Python C/API extension modules that provide a connection between Python and Fortran*[1].

[1] Homepage

[lab] Python binding with F2PY for a Cityblock distance matrix Fortran90 subroutine

- Let's go to the notebook `cityblock-distance-matrix-fortran.ipynb`.
  - Open a terminal and build the libraries. The same python executable that's used for the kernel must be used to build the libraries. You can find the path to it on the kernel's launcher script.
  - Notice that the empty array is created with `order='F'`. and that the `x` array containing the dataset is passed transposed.
  - You may try to build the libraries using the 'by hand' command described on the notebook. Make sure that you copy the `.so` files to the folder `metrics` to the directory level where the notebooks are running. Alternatively you could add the directory that contains the `.so` files to the `PYTHONPATH`.
- Run all cells and compare the execution times of the different approaches.
- While running the `%timeit` function calls, you may open a terminal and check with `top` that the function runs in multiple threads.

C Foreign Function Interface for Python

CFFI

CFFI enables calling C code from Python without learning a third language, to be used as interface. CFFI interacts with almost any C code from Python, based on C-like declarations[1].

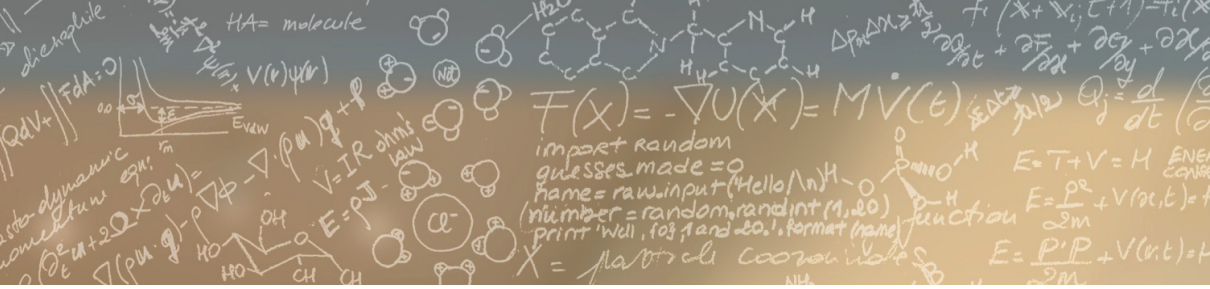
[1] Homepage

[lab-homework] Python binding with cffi for a Cityblock distance matrix C function

- Let's go to the folder `cityblock-cffi`.
  - Open a terminal and build the libraries following the instructions on the `README.md` file. The same python executable that was used for the kernel on the other tutorials must be used to build the libraries. You can find the path to it on the kernel's launcher script.
  - Notice that in this case two libraries are generated. A C library that contains the Cityblock distance matrix function and a python-importable library that's linked to it, which does the biding from C to Python.
  - You may create a notebook similar to the one with F2PY where you write a wrapper function for the `cbdm` C function. You can time it then and compare it to the NumPy implementation. While running the `%timeit` function calls, you may open a terminal and check with `top` that the function runs in multiple threads.

Conclusions

- When using NumPy, it pays off to find efficient ways to write vectorized code.
- Numba just-in-time compilations can be used to speed up python functions. There is much more in Numba, ex. `@numba.vectorize`, `@numba.cuda.jit`...
- Significant speedups can be obtained by implementing compute-intensive operations in C or Fortran90. Other solutions besides cffi are Cython and pybind11.
- Single-threaded operations can be run concurrently with Dask. Dask provides many other functionalities worth to have a look to.



Thank you for your attention!