UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Programa de Pós-Graduação em Engenharia Elétrica UFJF

Directional-iDBSCAN

a proposal to CYGNO

Igor Pains

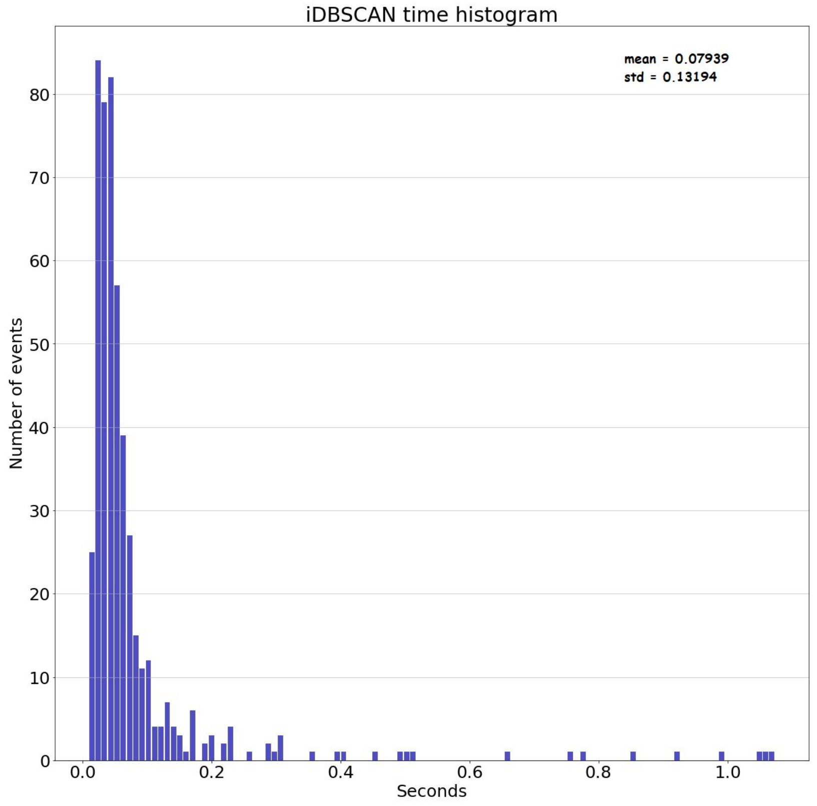
Igor Abritta and Rafael A Nobrega

Last presentation

- An implementation of the iDDBSCAN with its iteration part written in cython was made aiming to reduce the duration of the algorithm.

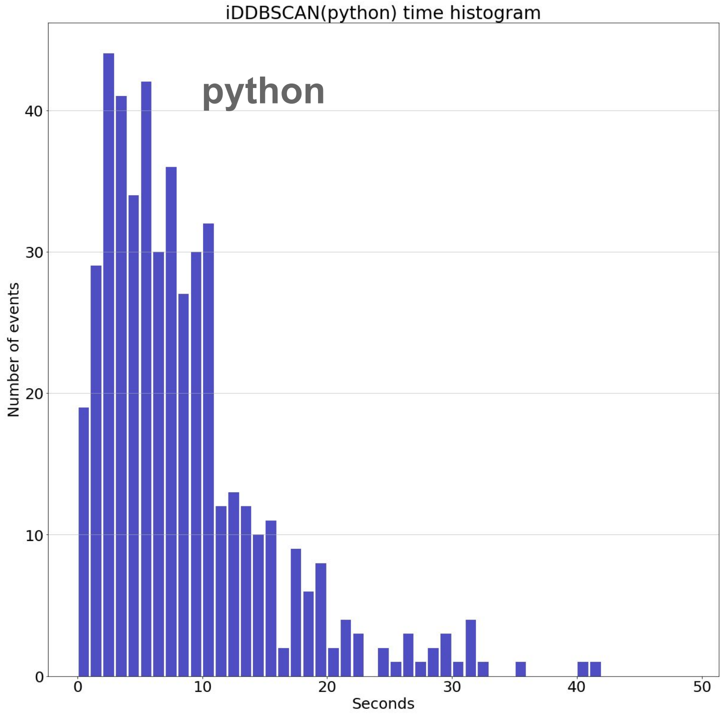
- The speed boost was not so high as other implementations found in the literature.

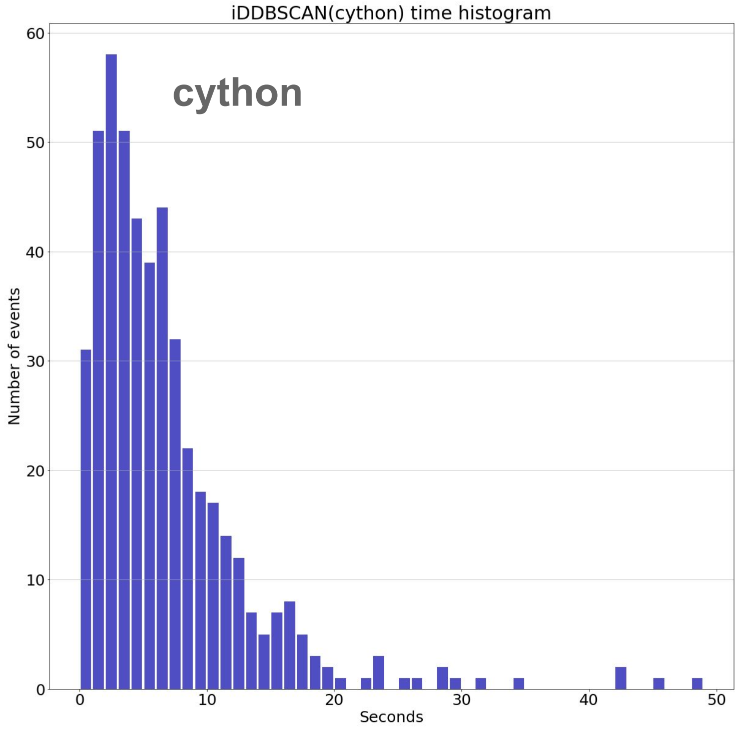
Time efficiency - iDBSCAN



- The iDBSCAN (one iteration) needed less than one second to do the clusterization in most part of the events.
- It took less than 0.15 seconds for most part of the events.
- The slowest event needed 1.073 seconds

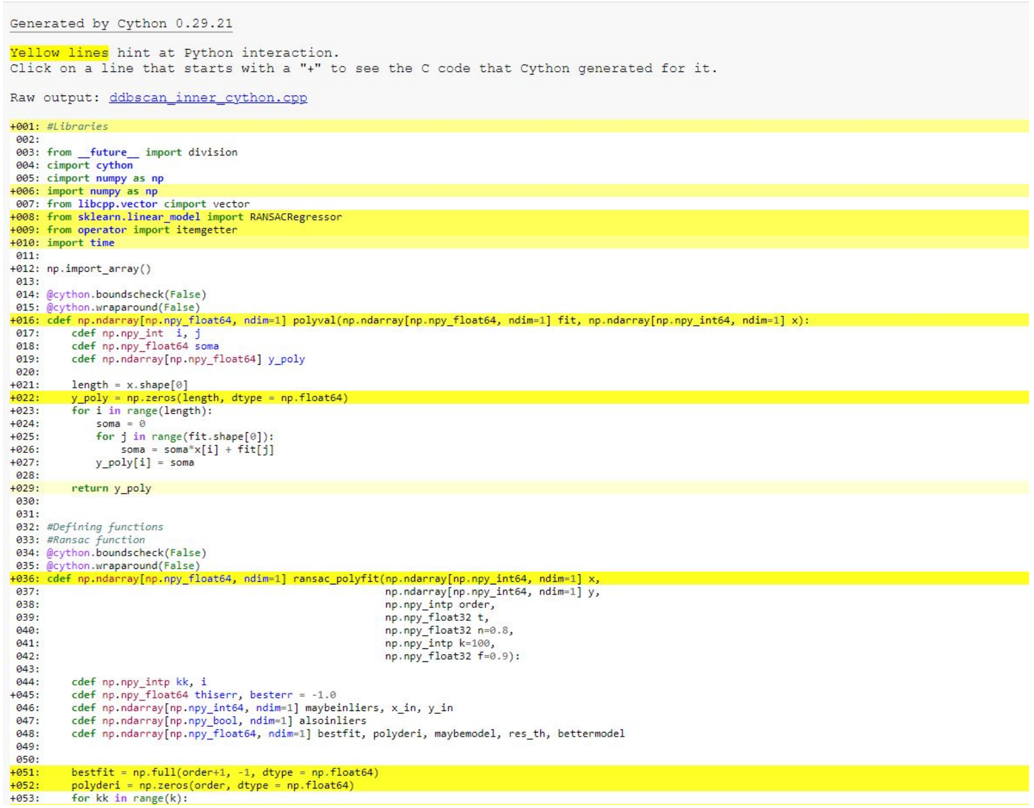
Time efficiency - iDDBSCAN (python vs cython)





The cython algorithm was 25% faster than the python version.

Cython.html



```
Generated by Cython 0.29.21

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

Raw output: ddbscan_inner_cython.cpp

+001: #Libraries
 002:
 003: from __future__ import division
 004: cimport cython
 005: cimport numpy as np
+006: import numpy as np
 007: from libcpp.vector cimport vector
+008: from sklearn.linear_model import RANSACRegressor
+009: from operator import itemgetter
+010: import time
 011:
+012: np.import_array()
 013:
 014: @cython.boundscheck(False)
 015: @cython.wraparound(False)
+016: cdef np.ndarray[np.npy_float64, ndim=1] polyval(np.ndarray[np.npy_float64, ndim=1] fit, np.ndarray[np.npy_int64, ndim=1] x):
 017:     cdef np.npy_int  i, j
 018:     cdef np.npy_float64 soma
 019:     cdef np.ndarray[np.npy_float64] y_poly
 020:
+021:     length = x.shape[0]
+022:     y_poly = np.zeros(length, dtype = np.float64)
+023:     for i in range(length):
+024:         soma = 0
+025:         for j in range(fit.shape[0]):
+026:             soma = soma*x[i] + fit[j]
+027:         y_poly[i] = soma
 028:
+029:     return y_poly
 030:
 031:
 032: #Defining functions
 033: #Ransac function
 034: @cython.boundscheck(False)
 035: @cython.wraparound(False)
+036: cdef np.ndarray[np.npy_float64, ndim=1] ransac_polyfit(np.ndarray[np.npy_int64, ndim=1] x,
 037:                                                        np.ndarray[np.npy_int64, ndim=1] y,
 038:                                                        np.npy_intp order,
 039:                                                        np.npy_float32 t,
 040:                                                        np.npy_float32 n=0.8,
 041:                                                        np.npy_intp k=100,
 042:                                                        np.npy_float32 f=0.9):
 043:
 044:     cdef np.npy_intp kk, i
+045:     cdef np.npy_float64 thiserr, besterr = -1.0
 046:     cdef np.ndarray[np.npy_int64, ndim=1] maybeinliers, x_in, y_in
 047:     cdef np.ndarray[np.npy_bool, ndim=1] alsoinliers
 048:     cdef np.ndarray[np.npy_float64, ndim=1] bestfit, polyderi, maybemodel, res_th, bettermodel
 049:
 050:
+051:     bestfit = np.full(order+1, -1, dtype = np.float64)
+052:     polyderi = np.zeros(order, dtype = np.float64)
+053:     for kk in range(k):
```

- By using “cython -a *cython_code.pyx*” an html page will be created to show how close to C the code is.

- The ideal is to have as many white lines as possible.

Cython updates

- Based on the article: *Cython for Speeding-up Genetic Algorithm*
  - DOI: 10.1109/ICEIT48248.2020.9113210
- There are some tips that may speed up the algorithm:
  - Specify the types of the variables, arrays and its elements. ✔
  - Loop through arrays using indexing.
  - Disable unnecessary features such as check array limits and negative indexes.
  - Use arrays and vectors instead of lists.
  - Create new arrays instead of access them if they are needed more than once.
  - Avoid type casts.

Results and conclusions

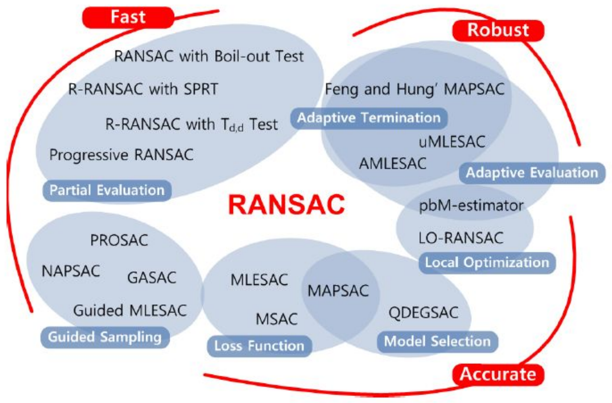
- The speed boost by using a cython version of the *ddbscan_inner* continued small, it was 30% faster than the python version after the updates.

- Almost 80% of the duration of the algorithm is on the directional search, mainly due to the *ransac_polyfit* function, made with a lot of numpy functions.

- Put the algorithm in cython is not always equivalent to have a high speed up, specially if the python algorithm has a lot of library calls.

Next steps

- Change the number of attempts in the *ransac_polyfit* in order to find the best parameter conciliating results and time efficiency.
- Modify the algorithm to just expand the neighborhoods of *core_points* at the directional search.
- Search faster implementations of RANSAC.


Fast
Robust
Accurate
RANSAC
RANSAC with Boil-out Test
R-RANSAC with SPRT
R-RANSAC with $T_{d,d}$ Test
Progressive RANSAC
Partial Evaluation
Feng and Hung' MAPSAC
Adaptive Termination
uMLESAC
AMLESAC
Adaptive Evaluation
pbM-estimator
LO-RANSAC
Local Optimization
PROSAC
NAPSAC
GASAC
Guided MLESAC
Guided Sampling
MLESAC
MSAC
Loss Function
MAPSAC
QDEGSAC
Model Selection