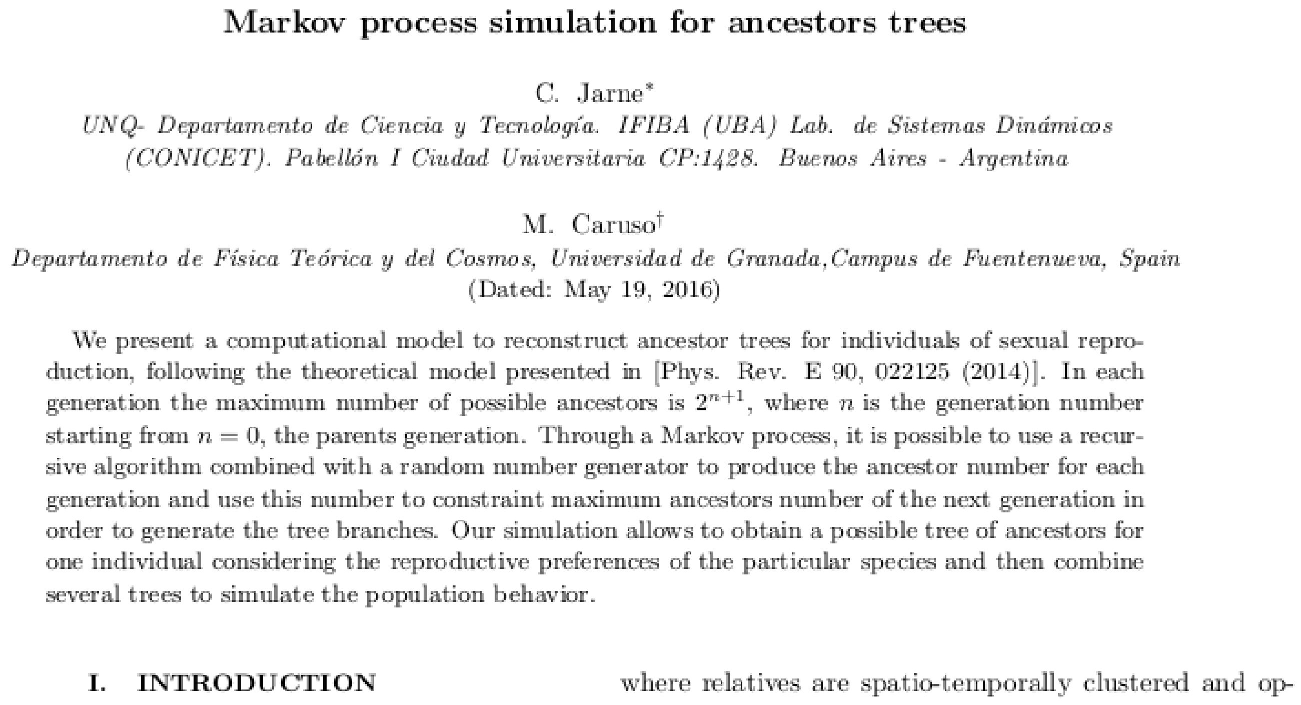
Markov process simulation for ancestors trees (the code parallelization)

By C. Jarne (IFIBA) and Universidad Nacional de Quilmes



Markov process simulation for ancestors trees

C. Jarne*
UNQ- Departamento de Ciencia y Tecnología. IFIBA (UBA) Lab. de Sistemas Dinámicos (CONICET). Pabellón I Ciudad Universitaria CP:1428. Buenos Aires - Argentina

M. Caruso†
Departamento de Física Teórica y del Cosmos, Universidad de Granada,Campus de Fuentenueva, Spain

(Dated: May 19, 2016)

We present a computational model to reconstruct ancestor trees for individuals of sexual reproduction, following the theoretical model presented in [Phys. Rev. E 90, 022125 (2014)]. In each generation the maximum number of possible ancestors is 2^{n+1}, where n is the generation number starting from $n = 0$, the parents generation. Through a Markov process, it is possible to use a recursive algorithm combined with a random number generator to produce the ancestor number for each generation and use this number to constraint maximum ancestors number of the next generation in order to generate the tree branches. Our simulation allows to obtain a possible tree of ancestors for one individual considering the reproductive preferences of the particular species and then combine several trees to simulate the population behavior.

I. INTRODUCTION

where relatives are spatio-temporally clustered and op-

Soon in PRE (we hope!)

Markov process simulation for ancestors trees (the code parallelization)

We need to create a set of trees of N generations to study properties!!

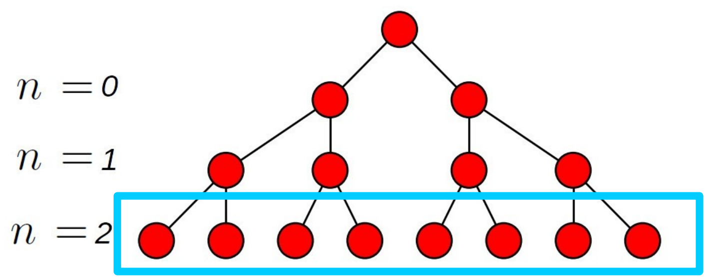
Markovian serial tree

Other other possible Markovian serial tree

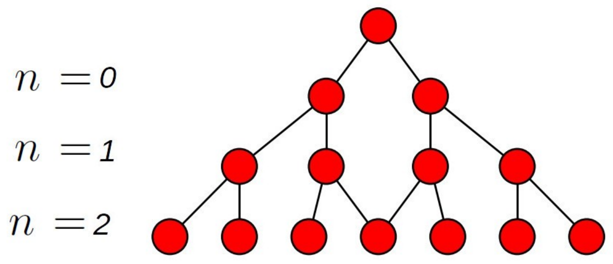
Other...even more endogamic Markovian serial tree

More

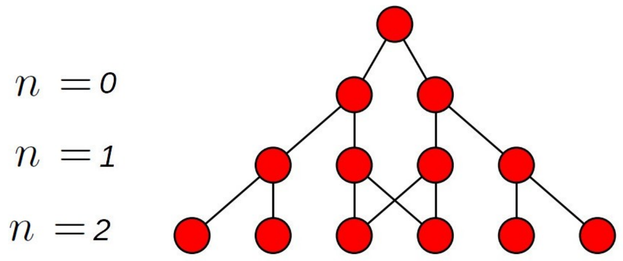
a)



b)



c)



Markov process simulation for ancestors trees (the code parallelization)

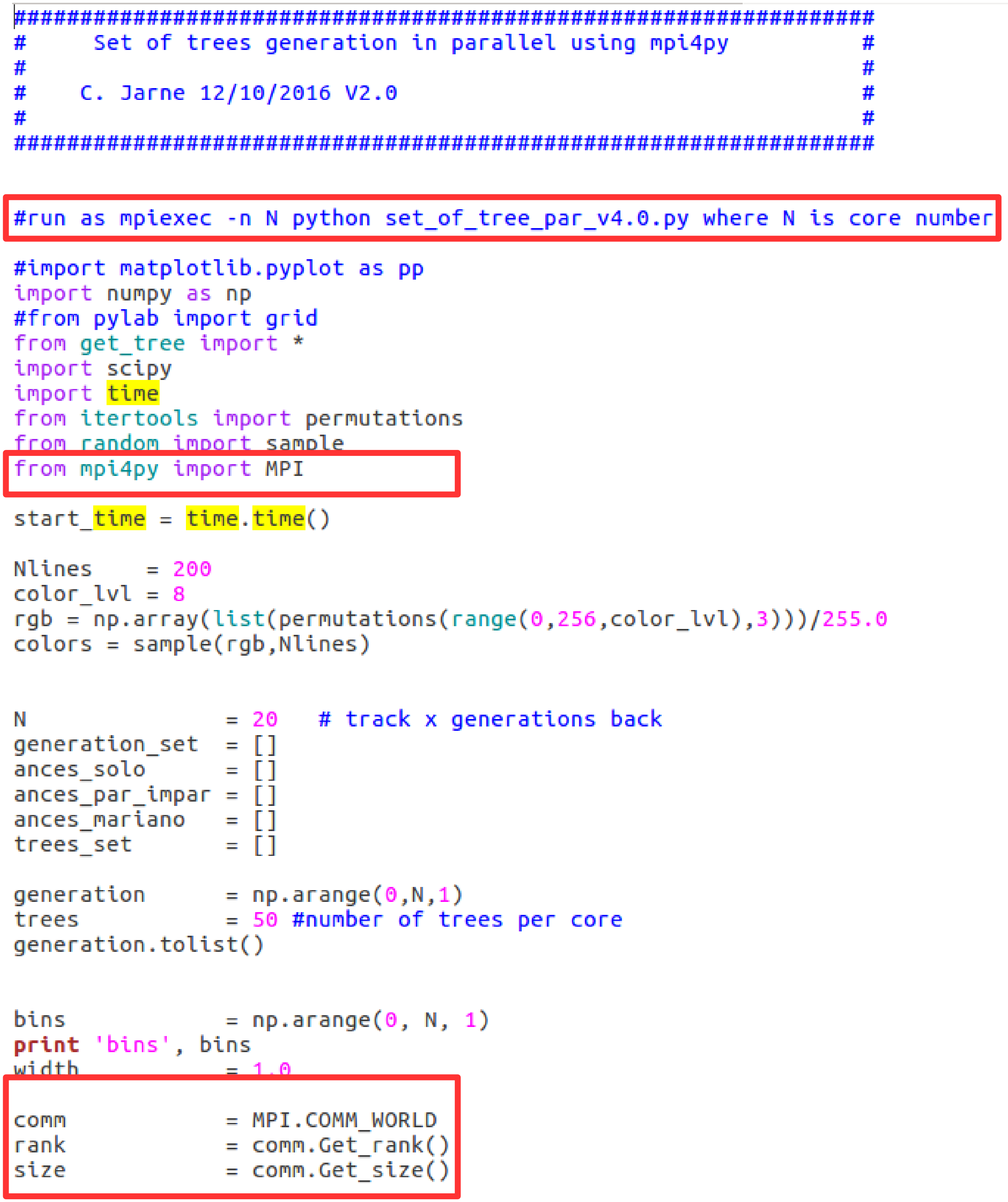
I try with little steps:

1) Since my code is in python I installed OpenBLAS library and compile numpy with OpenBLAS integration (not as easy as I expected).

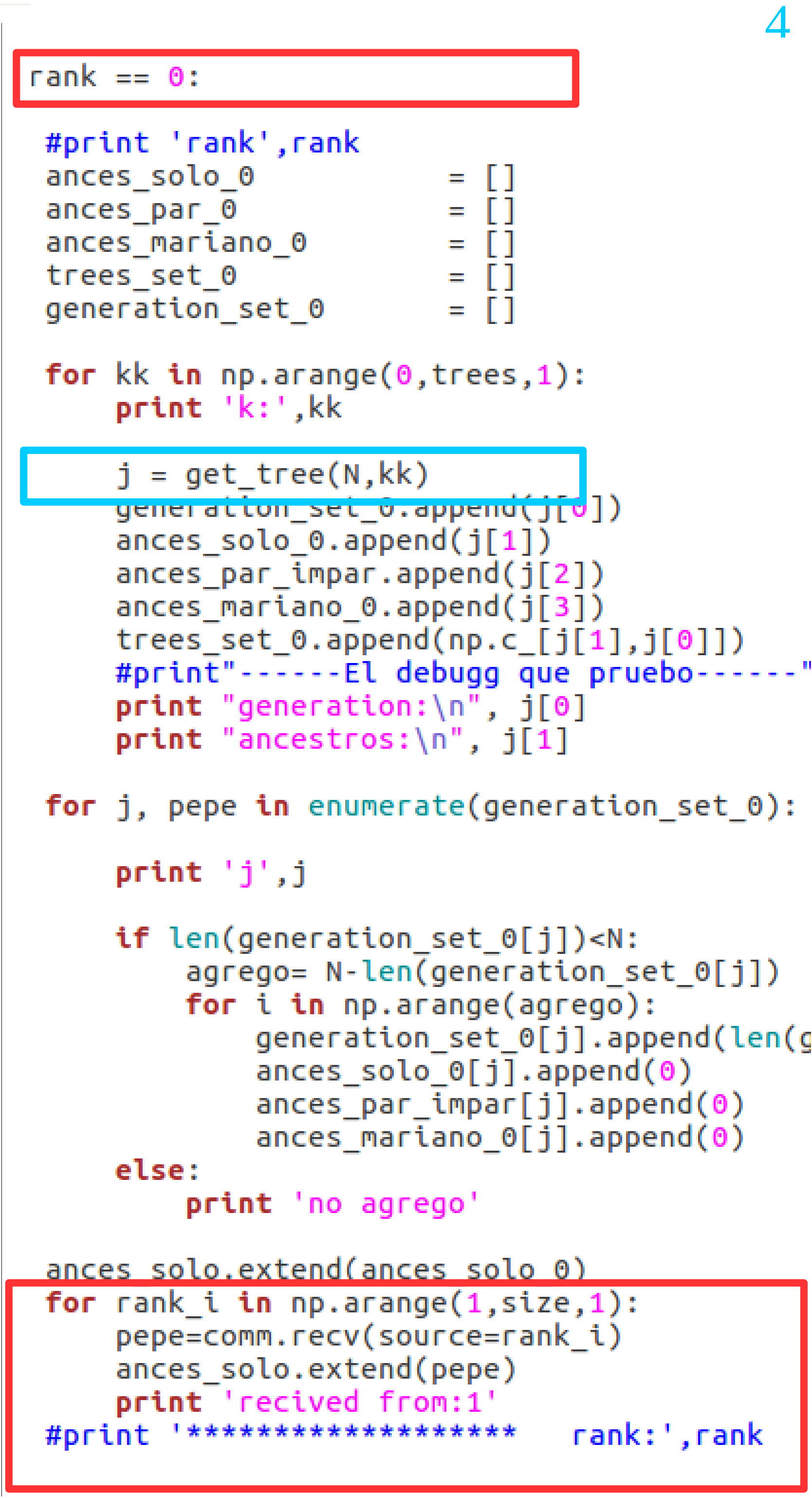
2) Then I used MPI for python (I installed the package MPI4py by Eric)

3) Now the hardest part! I adapted my serial code to used MPI4py.

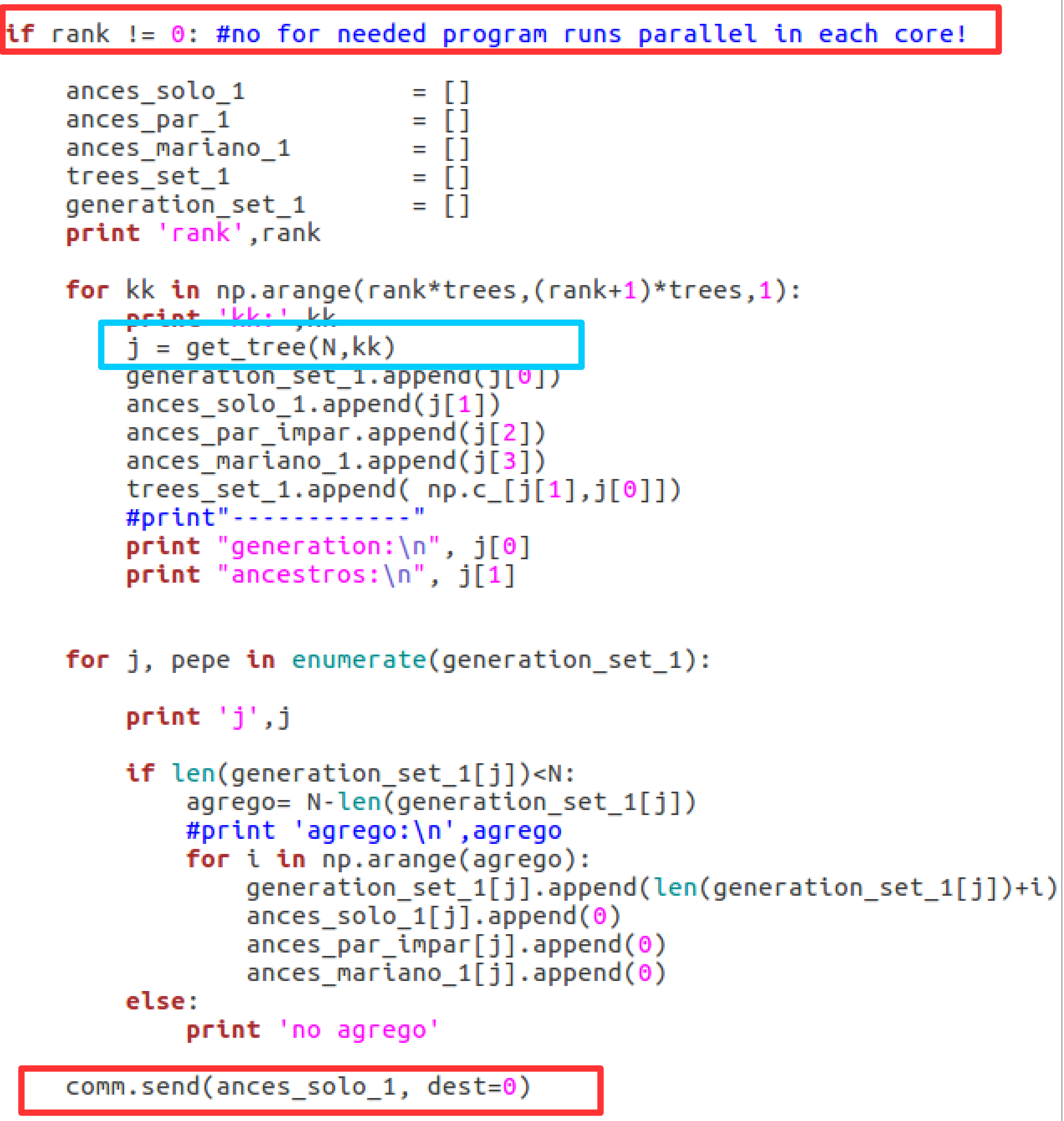
I did it in small steps. First I try to divide the work in to processes up to the final version 4.0



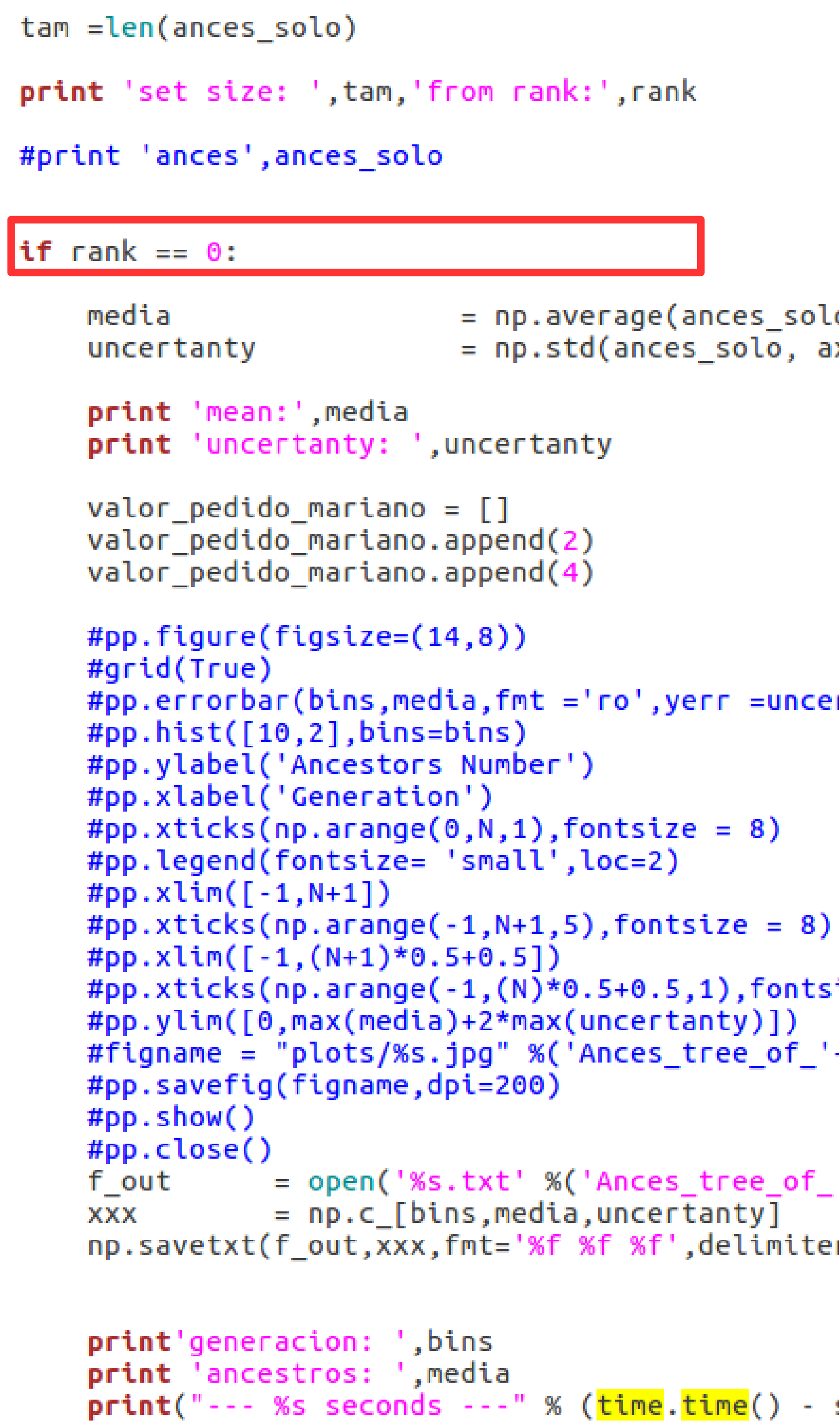
```
##################################################################
#     Set of trees generation in parallel using mpi4py          #
#                                                                #
#    C. Jarne 12/10/2016 V2.0                                    #
#                                                                #
##################################################################


#run as mpiexec -n N python set_of_tree_par_v4.0.py where N is core number

#import matplotlib.pyplot as pp
import numpy as np
#from pylab import grid
from get_tree import *
import scipy
import time
from itertools import permutations
from random import sample
from mpi4py import MPI

start_time = time.time()

Nlines    = 200
color_lvl = 8
rgb = np.array(list(permutations(range(0,256,color_lvl),3)))/255.0
colors = sample(rgb,Nlines)


N                = 20   # track x generations back
generation_set   = []
ances_solo       = []
ances_par_impar  = []
ances_mariano    = []
trees_set        = []

generation       = np.arange(0,N,1)
trees            = 50 #number of trees per core
generation.tolist()


bins             = np.arange(0, N, 1)
print 'bins', bins
width            = 1.0

comm             = MPI.COMM_WORLD
rank             = comm.Get_rank()
size             = comm.Get_size()
```



```
rank == 0:

  #print 'rank',rank
  ances_solo_0           = []
  ances_par_0            = []
  ances_mariano_0        = []
  trees_set_0            = []
  generation_set_0       = []

  for kk in np.arange(0,trees,1):
      print 'k:',kk

      j = get_tree(N,kk)
      generation_set_0.append(j[0])
      ances_solo_0.append(j[1])
      ances_par_impar.append(j[2])
      ances_mariano_0.append(j[3])
      trees_set_0.append(np.c_[j[1],j[0]])
      #print"------El debugg que pruebo------"
      print "generation:\n", j[0]
      print "ancestros:\n", j[1]

  for j, pepe in enumerate(generation_set_0):

      print 'j',j

      if len(generation_set_0[j])<N:
          agrego= N-len(generation_set_0[j])
          for i in np.arange(agrego):
              generation_set_0[j].append(len(g
              ances_solo_0[j].append(0)
              ances_par_impar[j].append(0)
              ances_mariano_0[j].append(0)
      else:
          print 'no agrego'

  ances_solo.extend(ances_solo_0)
  for rank_i in np.arange(1,size,1):
      pepe=comm.recv(source=rank_i)
      ances_solo.extend(pepe)
      print 'recived from:1'
  #print '******************    rank:',rank
```



```
if rank != 0: #no for needed program runs parallel in each core!

    ances_solo_1              = []
    ances_par_1               = []
    ances_mariano_1           = []
    trees_set_1               = []
    generation_set_1          = []
    print 'rank',rank

    for kk in np.arange(rank*trees,(rank+1)*trees,1):
        print 'kk:',kk
        j = get_tree(N,kk)
        generation_set_1.append(j[0])
        ances_solo_1.append(j[1])
        ances_par_impar.append(j[2])
        ances_mariano_1.append(j[3])
        trees_set_1.append( np.c_[j[1],j[0]])
        #print"------------"
        print "generation:\n", j[0]
        print "ancestros:\n", j[1]


    for j, pepe in enumerate(generation_set_1):

        print 'j',j

        if len(generation_set_1[j])<N:
            agrego= N-len(generation_set_1[j])
            #print 'agrego:\n',agrego
            for i in np.arange(agrego):
                generation_set_1[j].append(len(generation_set_1[j])+i)
                ances_solo_1[j].append(0)
                ances_par_impar[j].append(0)
                ances_mariano_1[j].append(0)
        else:
            print 'no agrego'

    comm.send(ances_solo_1, dest=0)
```



```
tam =len(ances_solo)

print 'set size: ',tam,'from rank:',rank

#print 'ances',ances_solo


if rank == 0:

    media                    = np.average(ances_solo
    uncertanty               = np.std(ances_solo, a

    print 'mean:',media
    print 'uncertanty: ',uncertanty

    valor_pedido_mariano = []
    valor_pedido_mariano.append(2)
    valor_pedido_mariano.append(4)

    #pp.figure(figsize=(14,8))
    #grid(True)
    #pp.errorbar(bins,media,fmt ='ro',yerr =unce
    #pp.hist([10,2],bins=bins)
    #pp.ylabel('Ancestors Number')
    #pp.xlabel('Generation')
    #pp.xticks(np.arange(0,N,1),fontsize = 8)
    #pp.legend(fontsize= 'small',loc=2)
    #pp.xlim([-1,N+1])
    #pp.xticks(np.arange(-1,N+1,5),fontsize = 8)
    #pp.xlim([-1,(N+1)*0.5+0.5])
    #pp.xticks(np.arange(-1,(N)*0.5+0.5,1),fonts
    #pp.ylim([0,max(media)+2*max(uncertanty)])
    #figname = "plots/%s.jpg" %('Ances_tree_of_'
    #pp.savefig(figname,dpi=200)
    #pp.show()
    #pp.close()
    f_out       = open('%s.txt' %('Ances_tree_of_
    xxx         = np.c_[bins,media,uncertanty]
    np.savetxt(f_out,xxx,fmt='%f %f %f',delimite


    print'generacion: ',bins
    print 'ancestros: ',media
    print("--- %s seconds ---" % (time.time() -
```

Parallel np=2

set size: 100

--- 13.1909019947 seconds ---

Serial

set size: 100

--- 20.6414310932 seconds ---

$\longrightarrow$ Me!!!!

Next!!!!!!

- Finish the second project I started.

  - parallelize a my code to analyze time series (bird songs) in chunks.

- A first step now I can run it in separated directories simultaneously to get the analysis plot.