```
#!/usr/bin/env python
import cytosim
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

# First frame to be analyzed : we need to leave time for initiation
__init_frame__ = 0
# Minimum length of MT to be considered
__threshold_length__ = 0.75
# Minimum radius to count the +end as "on the membrane"
__threshold_rad__ = 3.75

def get_r_z(fiber):
    """ A function to return radial and z coordinate of MT + end """
    pt = fiber.posEndP()
    return [np.sqrt(np.sum(np.square(pt[0:2]))),pt[2]]

if __name__ == "__main__":
    """ An analysis of MT + ends positions """
    ## Initiation
    # Opening a simulation that has already run (reads objects.cmo and
properties.cmo in current folder)
    sim = cytosim.open()
    # Initial frame
    frame = sim.load(__init_frame__)
    # counting frame number
    nb_frames = 0
    while frame:
        nb_frames +=1
        frame = sim.next()
    # preparing variables
    nb_MT = len(sim.fibers)
    n_max = nb_MT * nb_frames # maximum expected number of MT
    pos_end = np.zeros((n_max, 2)) # array containing all MT position
through history

    # Getting the parameters and results in a dictionary
    res_and_pars = {}
    frame = sim.frame()

    #  MT property :
    mt_prop = frame["microtubule"].prop
    res_and_pars["growing_speed"] = mt_prop.growing_speed()[0]
    res_and_pars["growing_off_speed"] = mt_prop.growing_off_speed()[0]
    res_and_pars["catastrophe_rate"] = mt_prop.catastrophe_rate()[0]
    res_and_pars["catastrophe_rate_stalled"] =
mt_prop.catastrophe_rate_stalled()[0]
    res_and_pars["shrinking_speed"] = mt_prop.shrinking_speed()[0]

    ## Analysis
    # Gathering end positions
    count = 0
    sim.load(__init_frame__)
```

```
    while sim.next():
        for fib in sim.fibers:
            if fib.length() > __threshold_length__:
                # One more filament that is longer than the threshold
                pos_end[count, :] = get_r_z(fib)
                count += 1
    print(count)
    # If we have any filament, we make an histogram of end positions
    if count:
        # We keep only the + end positions that were measured
        pos_end = pos_end[0:count, :]
        print(pos_end)
        # we also identify the + ends at the membrane
        pos_end_mb = pos_end[pos_end[:, 0] >= __threshold_rad__, 1]

        # Now we plot
        #counts, bins = np.histogram(pos_end[:, 1])
        #plt.stairs(counts, bins)
        counts, bins = np.histogram(pos_end_mb[:])
        plt.stairs(counts, bins)
        plt.xlim([-20,20])
        plt.savefig("histo.png")

        # putting in the result dictionnary
        mean_pos = None
        mean_pos_mb = None
        if len(pos_end):
            # we make sure pos_end is not empty
            mean_pos = np.mean(pos_end[:, 1])
        if len(pos_end_mb):
            # we make sure pos_end_mb is not empty
            mean_pos_mb = np.mean(pos_end_mb[:])

        res_and_pars["mean_end_pos"] = mean_pos
        res_and_pars["mean_end_pos_mb"] = mean_pos_mb
        list= pos_end_mb[:]
        dict_plus_ends = {'Z_pos_of_plus_ends': list}
        df= pd.DataFrame(dict_plus_ends)
        df.to_csv('EB1_dataset.csv')

        # saving it to csv
        # we write the folder name as the row index to keep track
        # parameters and results are columns
        # this allows for quick concatenation of csv files
        folder_name = os.path.basename(os.getcwd())
        res_pars_dataframe = pd.DataFrame.from_dict({folder_name :
res_and_pars}, orient="index")
        res_pars_dataframe.to_csv(r'res_and_pars.csv',  header=True)
        EB1_dataframe = pd.DataFrame.from_dict({folder_name : EB1_dataset},
orient="index")
        EB1_dataframe.to_csv(r'EB1_dataset.csv',  header=True)
```

TEMPLATE

```
% Two asters and a nucleus

% The simulation configuration
set simul system
{
    time_step = 0.06
    viscosity = 1
    verbose = 0
    steric = 1, 100
    kT = 0.0042
    steric_max_range = 0.1
    display = (color = light_gray;)
}

% The cell ; now a cylinder along the vertical (Z) axis
set space cell
{
    shape = cylinderZ
    display = ( color=0x44444444, dark_gray; visible=3 )
}

new cell
{
    radius = 4
    bottom = -20
    top = 20
}

% Definiting microtubule proprteies
set fiber microtubule
{
    % They are stiff
    rigidity = 30
    confine = inside, 200
    segmentation = 1
    steric = 1, 0.05  % enable, radius
    % They have a dynamical instability
    activity = classic
    growing_speed     = 0.5,  0
    shrinking_speed   = -0.5316194421233533, 0
    catastrophe_rate  = 0.00014890275303706125, 0.0
    % When stalled, they undergo more catastrophe
    catastrophe_rate_stalled = 4.0, 0
    rescue_rate       = 0.34179675465975734,    0
    growing_force     = 0.4283064525442795,  3.5
    % They do not fully disappear if depolymrized
    persistent     = 1
    min_length     = 0.6
    display = ( line_width=3; color=green; )
    total_polymer = 500
}

% The centrosomes
set solid core
{
    % core is the solid at the center of centrosomes
```

```
    display = ( style=3; color=red; )
    viscosity = 10000000
}

% A centrosome is a solid holding MT with a given stiffness
set aster centrosome
{
    stiffness = 1000, 500
}

new centrosome
{
    solid = core
    position = 0 -1.25 0
    radius = 0.5
    point1 = center, 0.5
    fibers = 200, microtubule, ( length = 0.6; end_state = 1,4 )
}

new centrosome
{
    solid = core
    position = 0 1.25 0
    radius = 0.5
    point1 = center, 0.5
    fibers = 200, microtubule, ( length = 0.6; end_state=1,4 )
}

% The nucleus
set bead kern
{
    viscosity  = 1000000000
    confine = inside, 1000
    steric = 1
    display = ( style=7; color=blue; )
}

new 1 kern
{
    radius = 3
    position = 0 0 3.5
}

% Running the simulation
run system
{
    % Number of time steps (duration : nb_steps * time_step )
    nb_steps = 10000
    % Number of saved frames
    nb_frames = 200
}
```