Computer Simulations in Physics 1st Project: Simulating the fruiting body formation of Dictyostelium slime mould and its fractal properties

Lénárd Lajos Szánthó

November 9, 2020

1 Motivation

Dictyostelium is a genus of slime moulds originally thought to be fungi, but in the 20th century phylogenetic analysis has shown [2] that they are a sister-clade to fungi and animals as depicted on Figure 1. This revelation places them even earlier on the dated phylogenetic tree of life, which means that – among many other species – they are around the transition from unicellularity to multicellularity. Multicellularity emerged multiple times independently along the tree of life, yet alone on the tree of fungi ([3]) so identifying the phylogenetic relationships among and biological functions of these "transition" species is one of the main goals of Biology today.

The article [1] – whose simulation I am reproducing here – catches one of the aspects of these questions: how do the unicellular Dictyostelium cells show multicellular pattern when their food source is depleted? Keep on reading to explore the remarkable process, how Dictyostelium cells work together and sacrifice themselves to build their fruiting body!

2 Introduction

Dictyostelium is an unicellular eukaryotic organism feeding on bacteria. Once the food source is depleted it releases a chemical signal, cyclic adenosine monophosphate (cAMP) which – once reached a threshold concentration – switches the cells into multicellular-like behaviour: they start to aggregate and then exit the 2D space and build a fruiting body which can be carried away by wind or animals to a new feeding ground where their offsprings can prosper again. Some Dyctiostelium individuals are sacrificed in the process which is yet another typical trait of multicellularity.

The aggregation pattern can be observed visually and MacKay [5] decided to write a simulation with the goal of quantifying the process. I base my simulation on another paper by Kessler and Levine [1] which introduces the problem in a more formal yet compact way.

They base their simulation on an extended diffusion equation:

$$\frac{\partial c}{\partial t} = a^2 \nabla^2 c - \Gamma c + \text{sources} \tag{1}$$

where a is the lattice size, Γ is the decay constant, and sources are continuously refreshed based on the position and state of the cells.

The cells themselves oblige to the following rules as described in [1]:

- a cell has three states: dormant, excited, recovery

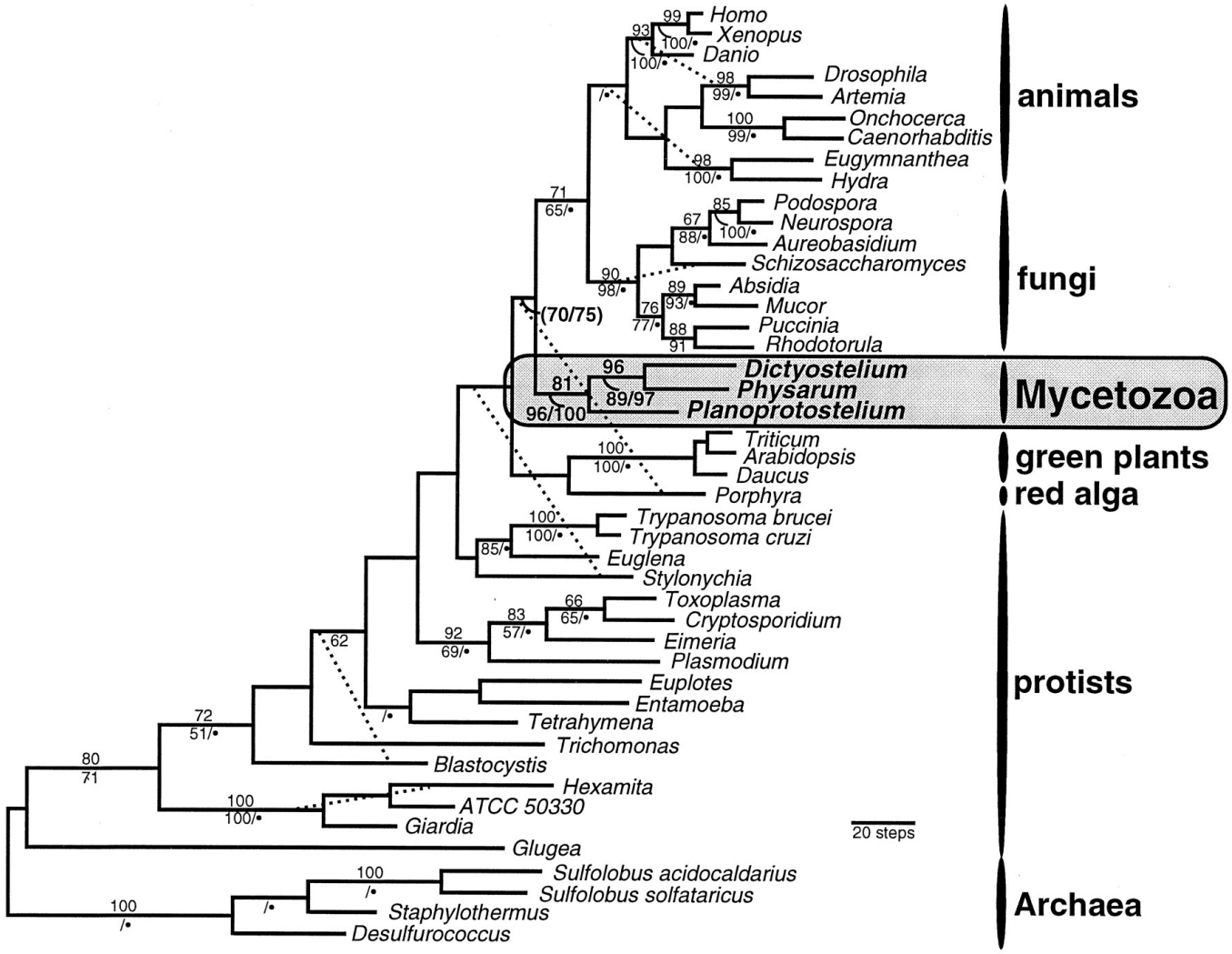

Figure 1: Phylogenetic reconstruction of [2] showing that EF-1α amino acid sequences support a monophyletic, late-branching Mycetozoa

- a dormant cell becomes excited if the cAMP concentration (c) is above threshold ($c_{\text{threshold}}$)
- an excited cell remains excited for τ time and releases Δc cAMP in total during this period
- after τ long excitation the cell enters a t_{R} time long recovery phase, during which it cannot be excited again
- when in excited state, the cell tries to move to a higher concentration neighbouring coordinate, if not already occupied
- only one successful move is allowed per excited state

3 Writing the simulation

The code for the simulation is published on GitHub `https://github.com/drenal/dictyostelium-camp`, please read the commits, *README.md* files and the code itself to have a full picture of the simulation and the obstacles met along the way. Below I just mention some of these in a dense format.

The simulation can be started with different parameter settings which correspond to the model explained in the previous section:

- -a LATTICE, –lattice LATTICE Lattice size (a)
- -g GAMMA, –gamma GAMMA decay constant (Γ)
- -r RHO, –rho RHO density of cells (ρ)
- -c THRESHOLD, –threshold THRESHOLD Threshold concentration ($c_{\text{threshold}}$)
- -d CAMP, –camp CAMP Amount of cAMP released by excited cell over period of tau (Δc)
- -t TAU, –tau TAU Timespan of a cell being in excited state (τ)
- -R RECOVERY, –recovery RECOVERY Timespan of a cell being resistant to excitation (t_{R})
- -m MESH, –mesh MESH 1D size of the 2D mesh to simulate on (it's always a square mesh)
- -s STEPS, –steps STEPS Steps to run PDE for
- -i IMPORTSTATE, –import IMPORTSTATE Import state from base path's files
- -o OUTPUT, –output OUTPUT Output state files base name
- -S SAMPLING, –sampling SAMPLING Number of steps to sample after

I have decided to use Object Oriented Programming paradigm to write the simulation mixed with the efficiency of vectorized procedural methods. The former was used for the cells, the later for the PDE solver (which was wrapped in a class, but is not following any real OOP-pattern). Some time was spent deciding on the best way to incorporate global settings (like size of mesh, cellular properties, etc.) into the simulation, at first I preferred to have a singleton class to hold all these properties which would then shared among all the class instances of the simulation, but in the end I have decided to use Python's class properties feature, which was the simplest to implement, yet achieving the goal of not having too much redundant data stored in the memory and possibly leading to inconsistent settings during the simulation.

The logic to decide where the cell moves can be understood in two ways in the paper [1]:

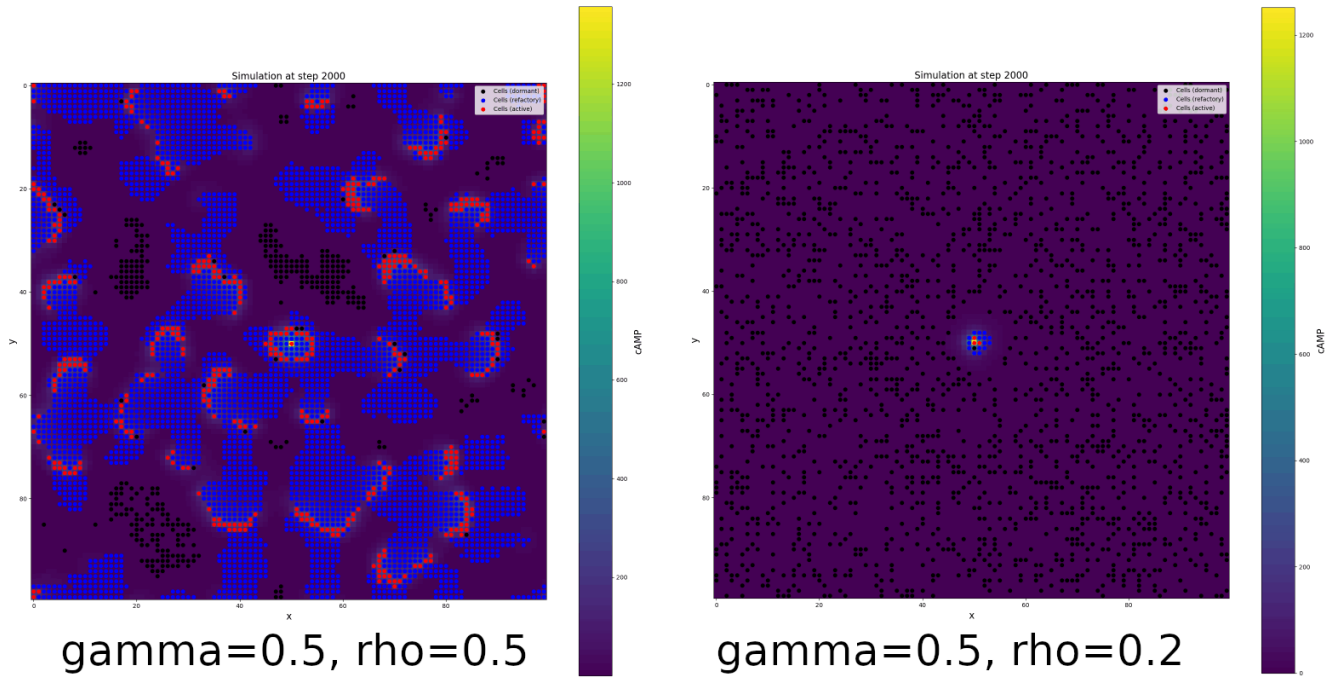

Figure 2: Final states of simulations started with the following parameters (on the left) $a = 1$, $\Gamma = 0.5$, $\rho = 0.5$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 20$; (on the right) $a = 1$, $\Gamma = 0.5$, $\rho = 0.2$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 20$. One can see that if the decay is too strong then a critical cell density (above 20%) is needed for successful signal propagation.

- the cell tries to move to the maximum concentration neighbour and fails if it's occupied
- the cell tries to move to the maximum concentration neighbour which is empty

I have implemented the later one, so a list is created of the neighbours with higher concentration than at the current position of the cell, but this creates thick-branched aggregation, so maybe the authors used the former rule.

The simulation first had PDE-solver and plotting together but this led to very slow runtimes (4 hours and above) and also to the inflexibility of changing the plots or reanalysing the in-between results.

I have killed two birds with one shot by orchestrating a state export mechanism which could be used both for creating the plots and for using them as inputs to continue the simulation from a given state. With this tweak the simulation time dropped from hours to minutes.

It was planned to also explore unit testing with Python in depth, but because of time constraints this was not done.

The Python module *cProfile* was used to benchmark the code and find bottlenecks. It has shed light on many bugs too.

I have tried to make the cell's update mechanism multithreaded, but creating the threads and copying the results back and forth in the memory made it not worth it. The PDE obviously couldn't be parallelised.

Once the simulation was efficient and free of programming and logical errors (at least hopefully), the next task at hand was to find the right parameters for the diffusion equation coupled with a cell machine. For this purpose bash scripts were utilized with the GNU-tool *parallel* to test many setups in a reasonable amount of time.

It turned out that the decay constant proposed by the [1] paper ($\Gamma = 0.5$) was too strong, as shown on Figure 2, one has to increase the cell density to 50% so that signal transduction is not broken.

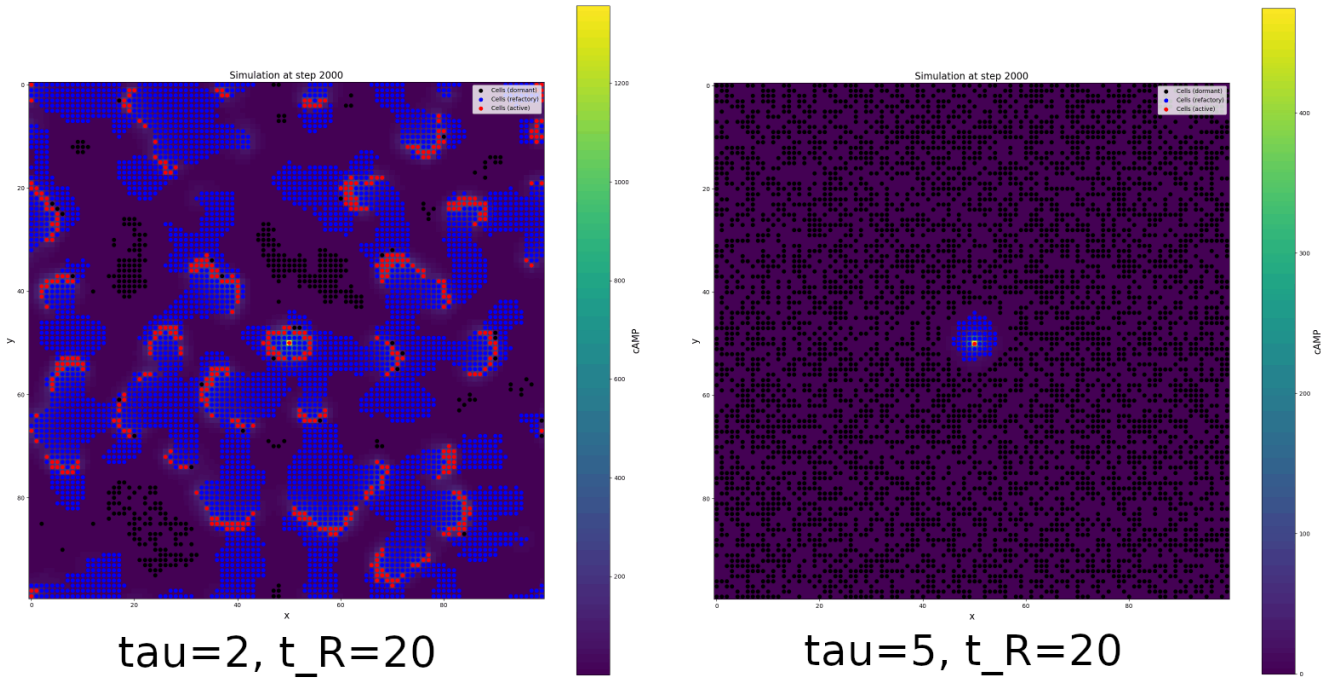

Figure 3: Final states of simulations started with the following parameters (on the left) $a = 1$, $\Gamma = 0.5$, $\rho = 0.5$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 20$; (on the right) $a = 1$, $\Gamma = 0.5$, $\rho = 0.5$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 5$, $t_{\text{R}} = 20$. One can see that if the cells spend more time active (which means more time to move), then the radius of signal transduction is very low (right picture).

I wanted to keep the 20% cell density proposed by the paper, so I have explored the parameter space for Γ and found that it should be lower than 0.13 to get patterns similar to the observed aggregation of Dictyostelium. The other parameters (once Γ and ρ are fixed) have a stable range, then if changed the signal transduction gets broken fast or silent islands emerge, see e.g. Figure 3 and Figure 4 where either the excitation time or the recovery time was changed from the ideal (2 and 20) values.

After the parameter search was done and the best parameters are found the last step is to quantify the resulting patterns (see Figure 5). For this I have chosen the fractal dimension which was determined with the box counting method. My code is heavily based on Dr Francesco's [6] blog entry.

The end results are the plots of the fitted lines on the box size versus number of needed boxes as shown on Figure 6.

4 Discussion

The goal of this project was to reproduce [1] paper's results and quantify them with the help of fractal dimension. All the necessary tools were developed and tested in order to do so, yet the results are not as similar to the paper's as anticipated. Three obvious reasons can be accounted for this discrepancy:

- the aforementioned cell update logic which wasn't clear in the paper. Maybe they did not use the list of proposals to move to, which I decided to implement
- even with the extensive parameter space exploration maybe I did not find the best set of parameters to use

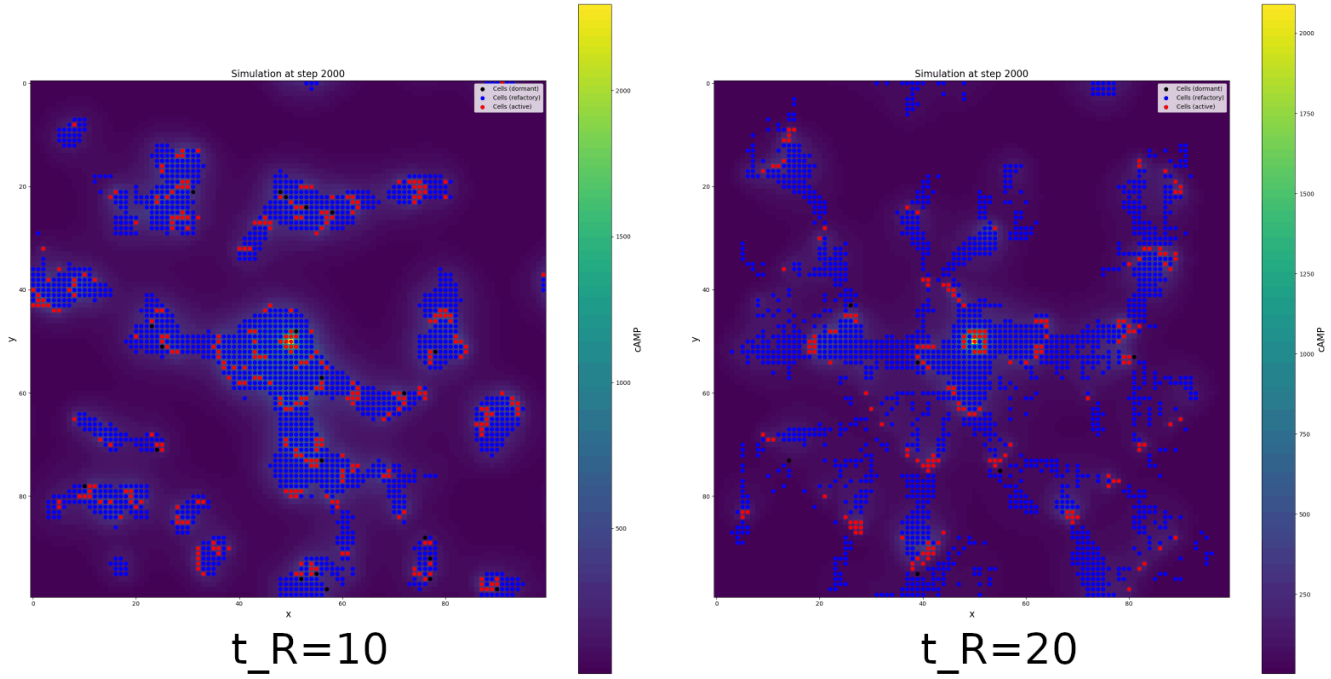

Figure 4: Final states of simulations started with the following parameters (on the left) $a = 1$, $\Gamma = 0.1$, $\rho = 0.2$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 10$; (on the right) $a = 1$, $\Gamma = 0.1$, $\rho = 0.2$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 20$. One can see that if the recovery time is shorter (cells get activated too soon compared to the concentration decay), we get thick branches.

- the central source is an always-active, but paralyzed (cancerous) cell in my simulation. The paper [1] mentions a periodic beacon or a spiral beacon.

Given enough time each of these possibilities could be explored and ruled out or proven.

The fractal dimension for 10 simulations with the best parameters $a = 1$, $\Gamma = 0.115$, $\rho = 0.2$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 20$ with mesh size 400×400 are:

$$d = 1.6649 \pm 0.0229 \qquad (\delta = 1.3\%) \tag{2}$$

This means that the simulations are fairly stable against input parameters, no wild results, which would emerge from e.g. bugs in the code.

5 Conclusion

This project was full of fun, the goal to reproduce the results of a 1993 paper with a simple diffusion equation looked rather easy, I have underestimated the task's nuances, like the dependence on parameters (although it's still a fairly stable and predictable PDE-system), and its dependence on the chosen cell behaviour strategy. I also didn't expect to learn, that *numpy*'s multiple dimensions array indexing is not as straightforward as one would think.

I have made a simulation from scratch which implements a diffusion-equation solver with forward Euler method incorporating a layer of autonomous cells. The connection between the two systems is through the sources term in the diffusion equation. During the project I have explored many *Python*-specific best-practices and solutions (e.g. tricks to make a simulation faster), deepen my knowledge in numerical methods.

The results (e.g. Figure 7) although do not fully agree with the paper's [1] results, this is probably due to the choice of cell movement strategy which was not clearly documented in the paper. It is interesting how we shifted in the last 2 decades to make the publications fully reproducible in many fields of science.

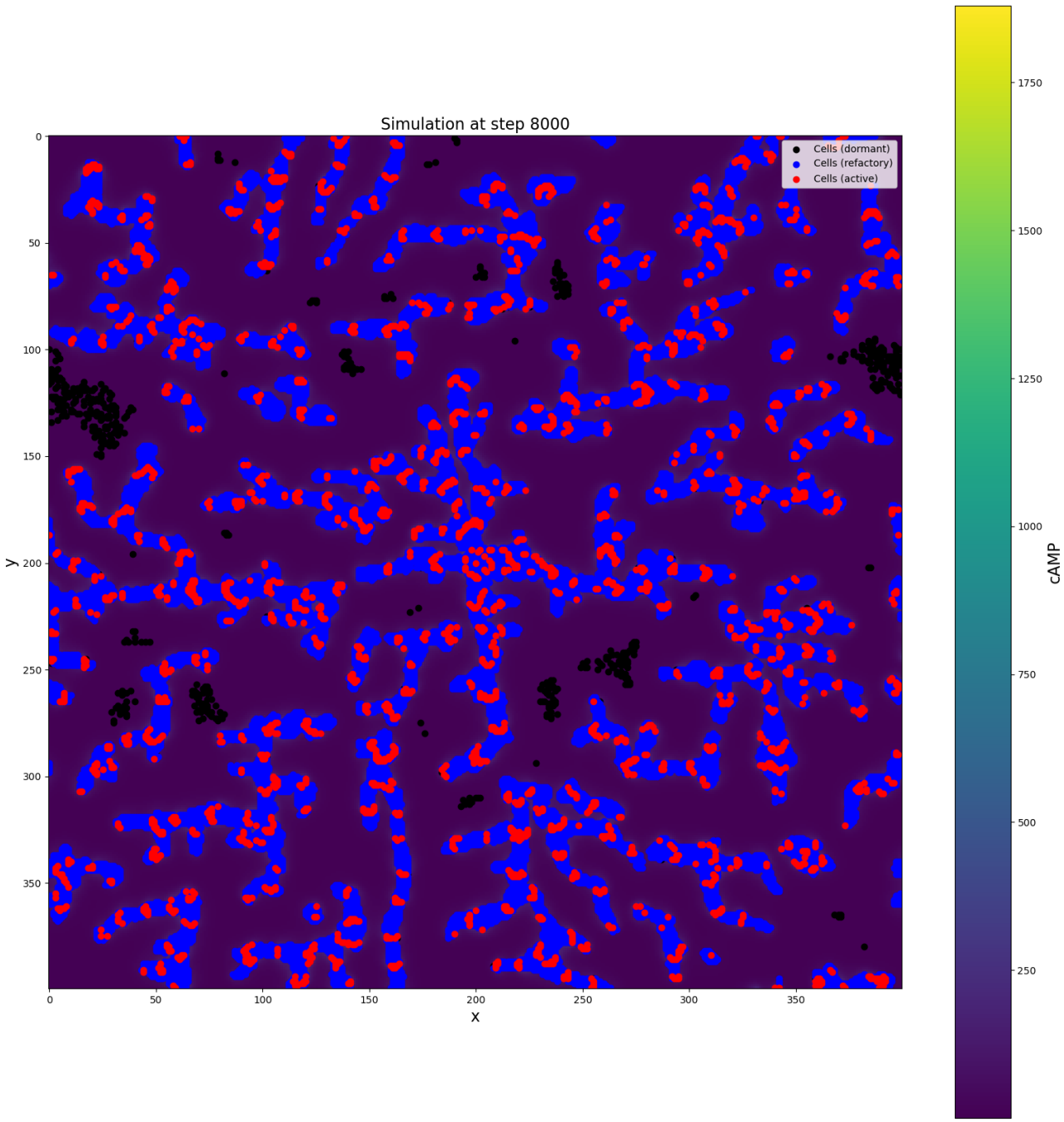

Figure 5: Final state of simulation started with the following parameters $a = 1$, $\Gamma = 0.115$, $\rho = 0.2$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 20$ with mesh size 400×400. Thin branches, periodic signal propagation through the whole population, yet not the anticipated fractal like structure.

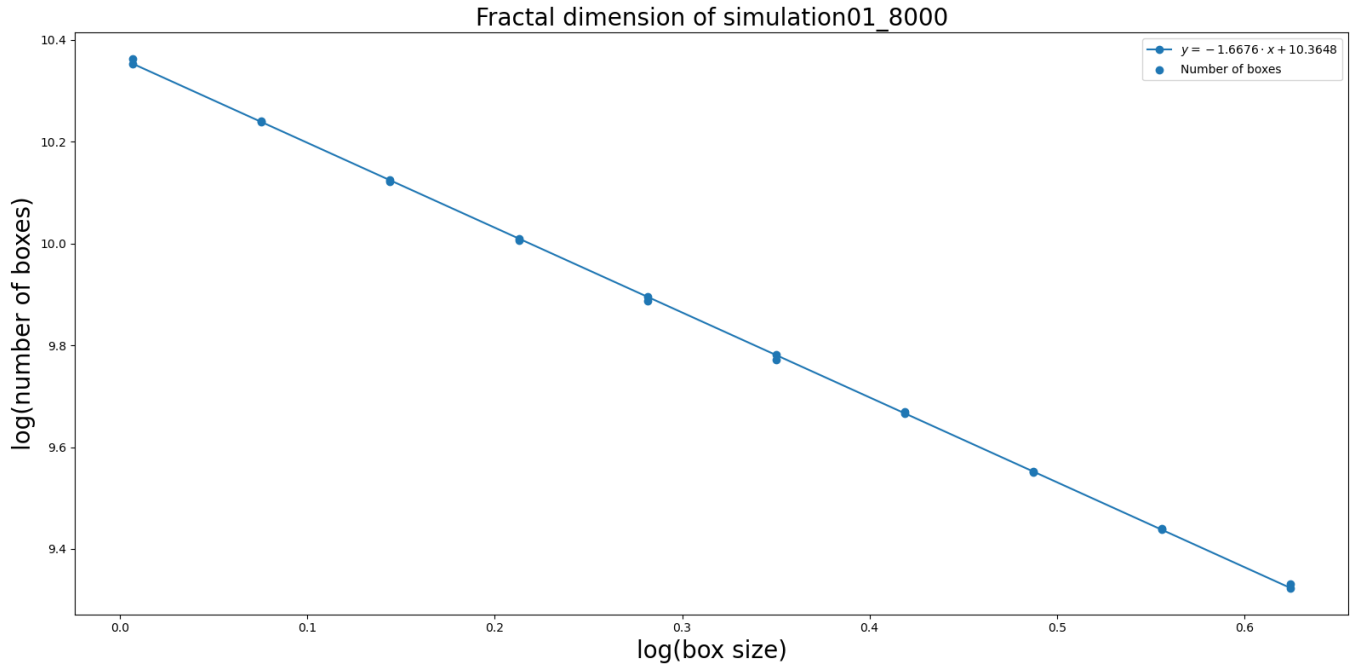

Figure 6: Fractal dimension fit result of the box counting method for the run presented on Figure 5

References

[1] David A. Kessler and Herbert Levine *Pattern formation in Dictyostelium via the dynamics of cooperative biological entities* Physical Review E 48(**6**), 1993

[2] Sandra L. Baldauf and W. Ford Doolittle *Origin and evolution of the slime molds(Mycetozoa)* PNAS 94 (**22**), 1997 `https://doi.org/10.1073/pnas.94.22.12007`

[3] László G. Nagy, Gábor M. Kovács and Krisztina Krizsán *Complex multicellularity in fungi: evolutionary convergence, single origin, or both?* Biol. Rev. pp. 000 – 000. 2018. `https://doi.org/10.1111/brv.12418`

[4] Rubin Landau, Manuel J. Paez and Cristian Bordeianu *A Survey of Computational Physics: Python Multimodal eBook*. Princeton University Press, 2011

[5] S.A. MacKay *Computer simulation of aggregation in Dictyostelium discoideum* Journal of Cell Science 33: 1-16; 1978 `https://jcs.biologists.org/content/33/1/1`

[6] Francesco Turci: *Box Counting in Numpy*, 2016. Accessed: 2020-11-09 `https://francescoturci.net/2016/03/31/box-counting-in-numpy/`

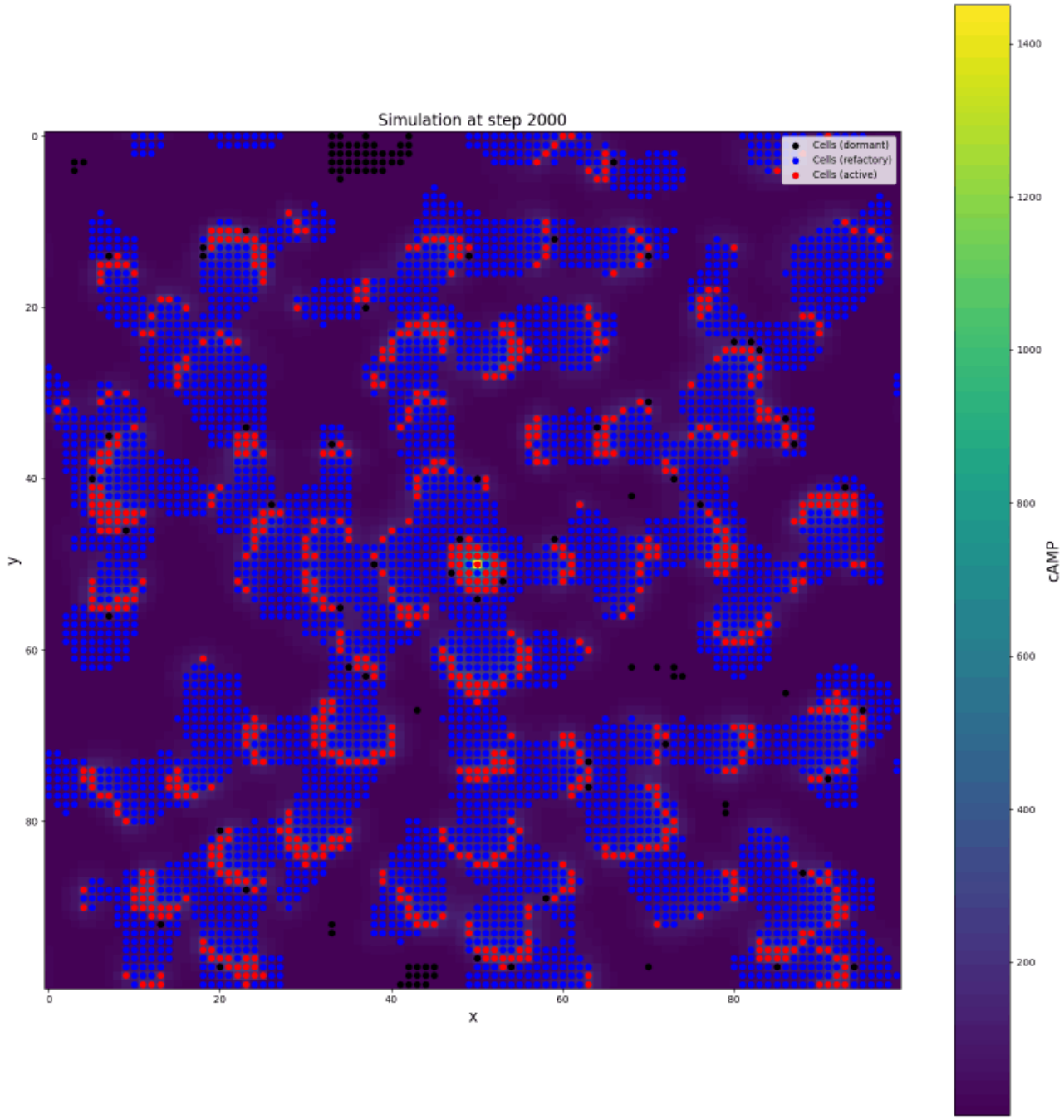

Figure 7: Final state of simulation started with the following parameters $a = 1$, $\Gamma = 0.5$, $\rho = 0.5$, $c_{\text{threshold}} = 20$, $\Delta c = 6000$, $\tau = 2$, $t_{\text{R}} = 10$ with mesh size 100×100