# Lab 5: Hands-on with PhysiCell (Day 2)

Get lectures and materials here!

Paul Macklin, Ph.D.

Intelligent Systems Engineering
Indiana University

January 28, 2020

## Part 1: Python data analysis

- Populate heterogeneity model
- Choose parameters, run and generate data
- Load single time point in Python
  - Make plots
  - Extract cell count information
- Make a script to analyze time series

## Let's start with the heterogeneity model

- Enter the root directory

- Reset to a clean slate
  - `make data-cleanup`
  - `make reset`

- Populate and build the heterogeneity model
  - `make heterogeneity-sample`
  - `make`

# About the heterogeneity model

- **Diffusing substrates:**
  - Oxygen
    - Diffusing from a Dirichlet boundary condition
- **Cell types:**
  - Cancer cells:
    - Each has an *oncoprotein* p
    - Cell cycling scales with oxygen availability
    - Cells with more *p* respond more to the oxygen and cycle faster
- **Initial condition:**
  - An initial circle of tumor cells
  - Each tumor cell has a random amount of *p*, with normal distribution

# Let's set options and run (1)

- Open `./config/PhysiCell-settings.xml`
- Let's set the domain size in the **domain** block
  - Switch to [-500,500] x [-500,500] x [-10,10] to speed it up

```
<PhysiCell_settings version="devel-version">
        <domain>
                <x_min>-500</x_min>
                <x_max>500</x_max>
                <y_min>-500</y_min>
                <y_max>500</y_max>
                <z_min>-10</z_min>
                <z_max>10</z_max>
                <dx>20</dx>
                <dy>20</dy>
                <dz>20</dz>
                <use_2D>true</use_2D>
        </domain>
```

## Let's set options and run (2)

- Let's also look at the **user_parameters** block
  - Let's change the oncoprotein standard deviation (`oncoprotein_sd`) to 0.5 (more variation)
  - Let's change the max oncoprotein (`oncoprotein_max`) to mean + 3 sds = 1 + 1.5 = 2.5

```
<user_parameters>
       <tumor_radius type="double" units="micron">250.0</tumor_radius>
       <oncoprotein_mean type="double" units="dimensionless">
              1.0</oncoprotein_mean>
       <oncoprotein_sd type="double" units="dimensionless">
              0.5</oncoprotein_sd>
       <oncoprotein_min type="double" units="dimensionless">0.0</oncoprotein_min>
       <oncoprotein_max type="double" units="dimensionless">2.5</oncoprotein_max>
       <random_seed type="int" units="dimensionless">0</random_seed>
</user_parameters>
```

# Let's set options and run (3)

- Let's look at the **overall** block
  - Set max time to 5 days = 5 x 24 x 60 = 7200 minutes

```
<overall>
        <max_time units="min">7200</max_time> <!-- 5 days * 24 h * 60 min -->
        <time_units>min</time_units>
        <space_units>micron</space_units>
```

- Let's look at the **save** block
  - Set the full save interval to 6 hours = 360 minutes

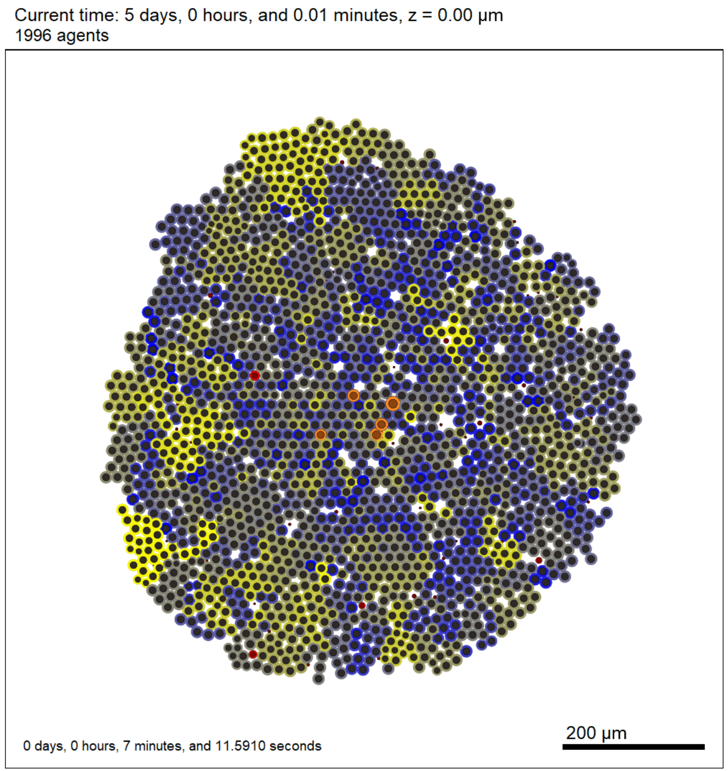
```
<save>
        <folder>output</folder> <!-- use . for root -->
        <full_data>
                <interval units="min">360</interval>
                <enable>true</enable>
        </full_data>
```

- Now, run! (`./heterogeneity`)

# Let's do a quick visualization

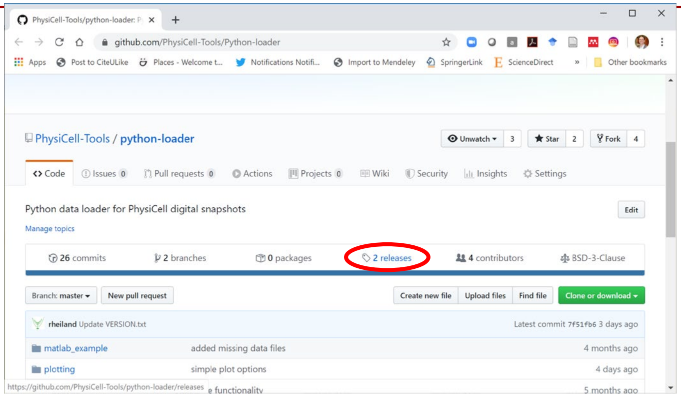
- `magick mogrify -format jpg *.svg`
- `magick convert *.svg out.gif`

- We can see that the yellow cells eventually "win": they grow faster and form microcolonies within the tumor
- The effect is greatest on the outside edge: They have access to more $O_2$ here



# Let's get ready to load in Python

- We'll go to Python-loader and get the source:
  - [https://github.com/PhysiCell-Tools/Python-loader](https://github.com/PhysiCell-Tools/Python-loader)
- Get the latest release:
  - Click on "releases" link
  - Click the green "clone or download" button
    - (For simplicity, I'm using "download ZIP" option)
- Copy all the Python files (end in .py) to the root of PhysiCell
  - `pyMCDS, pyMCDS_timeseries, read_MultiCellDS_xml,`
  - `test_single, test_timeseries`


PhysiCell-Tools/python-loader: P

github.com/PhysiCell-Tools/Python-loader

Apps | Post to CiteULike | Places - Welcome t... | Notifications Notifi... | Import to Mendeley | SpringerLink | ScienceDirect | » | Other bookmarks

PhysiCell-Tools / python-loader

Unwatch 3 | Star 2 | Fork 4

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

Python data loader for PhysiCell digital snapshots

Edit

Manage topics

26 commits | 2 branches | 0 packages | 2 releases | 4 contributors | BSD-3-Clause

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

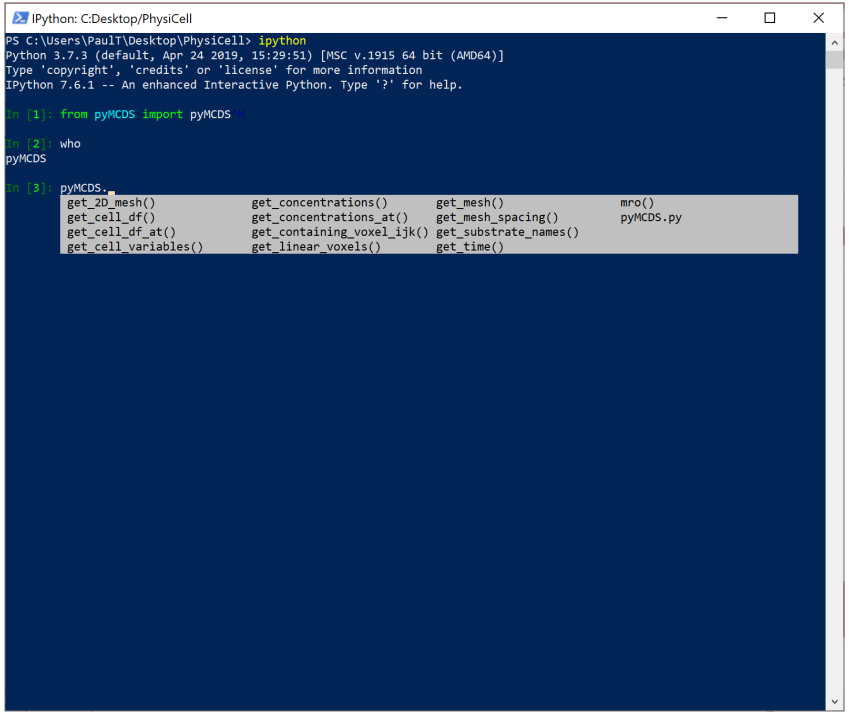
rheiland Update VERSION.txt — Latest commit 7f51fb6 3 days ago

| | | |
|---|---|---|
| matlab_example | added missing data files | 4 months ago |
| plotting | simple plot options | 4 days ago |
| | e functionality | 5 months ago |

https://github.com/PhysiCell-Tools/python-loader/releases

# Let's get started in ipython

- Start ipython (interactive python)
  - `ipython`
- Import the python loader:
  - `from pyMCDS import pyMCDS`
- Import other useful things
  - `import numpy as np`
  - `import matplotlib.pyplot as plt`
- Let's see what is available.
  - Type `pyMCDS.`
  - Hit "tab" to autocomplete
- Historical note:
  - MCDS = MultiCellDS, our multicellular data standard


IPython: C:Desktop/PhysiCell

```
PS C:\Users\PaulT\Desktop\PhysiCell> ipython
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.6.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from pyMCDS import pyMCDS^M

In [2]: who
pyMCDS

In [3]: pyMCDS.
         get_2D_mesh()            get_concentrations()       get_mesh()               mro()
         get_cell_df()            get_concentrations_at()    get_mesh_spacing()       pyMCDS.py
         get_cell_df_at()         get_containing_voxel_ijk() get_substrate_names()
         get_cell_variables()     get_linear_voxels()        get_time()
```

# Let's load a single saved time

- Syntax: result = pyMCDS( filename , directory ):

```
mcds = pyMCDS('output00000000.xml', 'output')
```

- Let's get some basic info on the snapshot:

```
print(mcds.get_time())   # what simulation time is saved here?
print(mcds.get_cell_variables()) # what data are saved in the cells?
print(mcds.get_substrate_names())  # what diffusing substrates?
```

- mcds.data is a dict. Let's see what's available:

```
mcds.data.keys()
Out[41]: dict_keys(['metadata', 'mesh', 'continuum_variables', 'discrete_cells'])
```

# Let's access some cell data

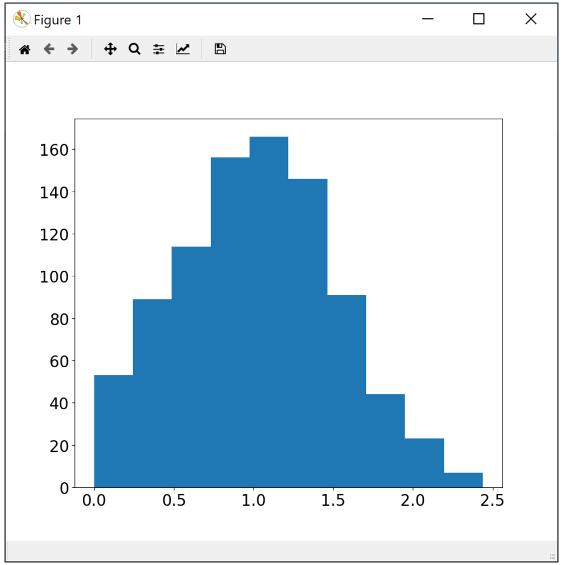
- First, let's find out the mean value of the oncoprotein
  - np.mean( mcds.data['discrete_cells']['oncoprotein'] )

```
Out[61]: 1.0177198768372775
```

- Let's make sure matplotlib doesn't use small fonts

```
import matplotlib
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
```

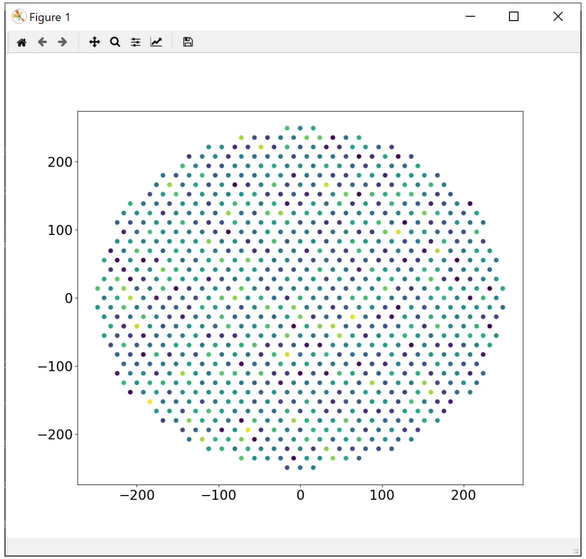
- Now, let's plot a histogram
  - `plt.hist( mcds.data['discrete_cells']['oncoprotein'] )`



# Let's plot the cells

- We'll do a scatter plot of the cells, and color by oncoprotein.
- First, let's grab the data to make our typing easier

```
cx = mcds.data['discrete_cells']['position_x']
cy = mcds.data['discrete_cells']['position_y']
op = mcds.data['discrete_cells']['oncoprotein']
```



- Now, a scatter plot.
  - Note: these are not plotting by the ***physical*** cell size

```
plt.scatter(cx,cy,c=op)
```

- This plot is pretty ugly. let's improve it.

# Improving the plot scatter plot

- Let's replot with bigger dots

```
plt.clf()
plt.scatter( cx , cy, c=op, s=200 )
```

- Make sure aspect ratio is right:

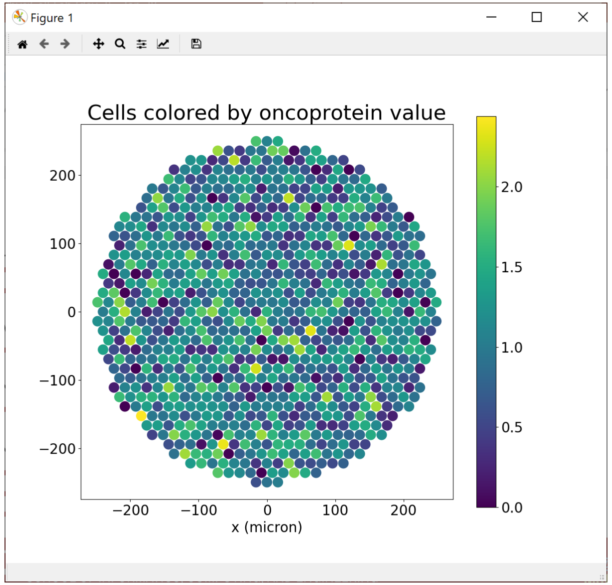
```
plt.axis( 'image' )
```

- Now, let's add a colorbar

```
plt.colorbar()
```

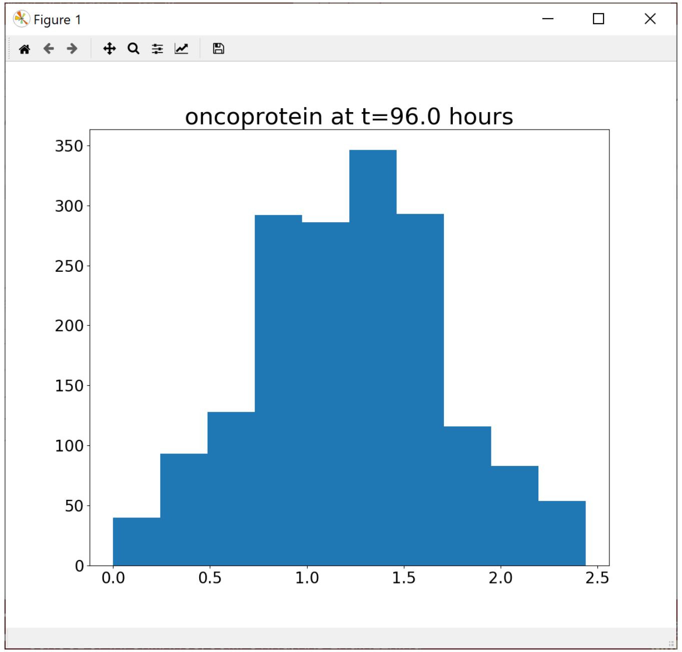
- Now, let's add labels

```
plt.title( 'Cells colored by oncoprotein value' , size=30)
plt.xlabel( 'x' , size=20 )
plt.ylabel( 'y', size=20 )
```



# Let's load another time

```
mcds = pyMCDS('output00000016.xml', 'output')
t=mcds.get_time()
cx = mcds.data['discrete_cells']['position_x']
cy = mcds.data['discrete_cells']['position_y']
op = mcds.data['discrete_cells']['oncoprotein']
plt.clf()
plt.hist( op )
plt.title( 'oncoprotein at t=' + \
str(t/60) + ' hours' , size=30)
```



# Let's find live and dead cells

- Each cycle model has a unique code
  - Codes ≥ 100 denote death cycles

- Let's get the cycle code of each cell, and convert to integers

```
cycle = mcds.data['discrete_cells']['cycle_model']
cycle = cycle.astype( int )
```

- Let's find the live cells

```
live = np.argwhere( cycle < 100 ).flatten()
dead = np.argwhere( cycle >= 100 ).flatten()
```

# Let's work with these

- Live and dead cell counts

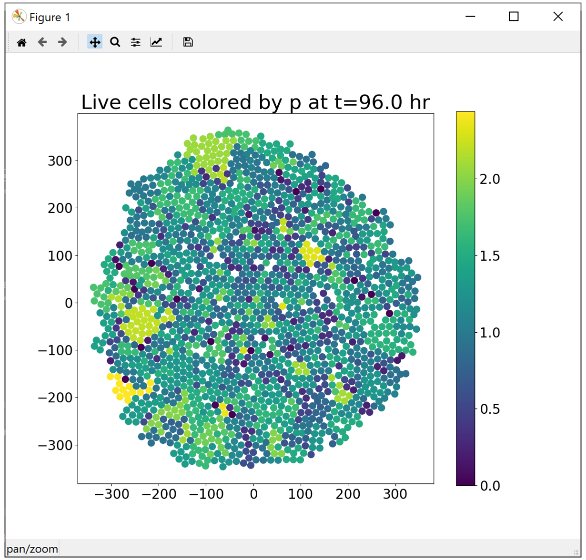
```
n_live = len( live )
n_dead = len( dead )
```

- Mean oncoprotein in live cells only

```
np.mean( op[live] )
```

- Let's scatter plot of only live cells

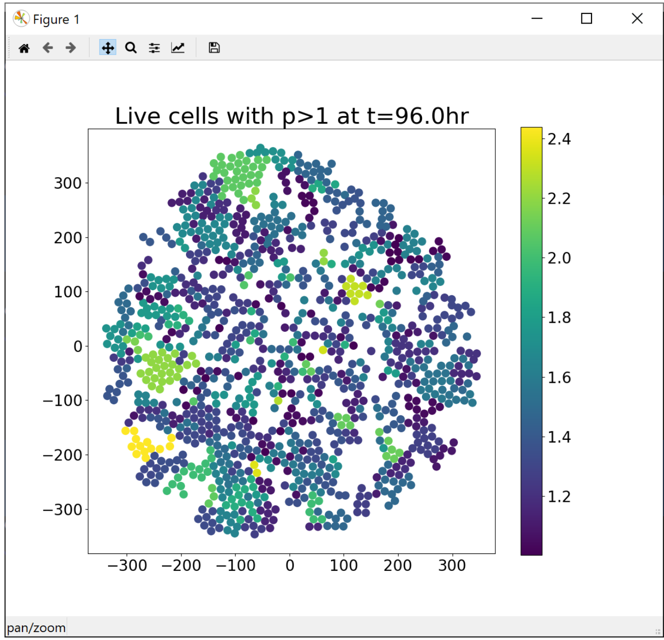
```
plt.clf()
plt.scatter( cx[live],cy[live],c=op[live],s=100);
plt.colorbar()
plt.axis('image')
plt.title( 'Live cells colored by p at t=' +str(t/60) + ' hr', size=30)
```



# Let's do a fancier search

- Only plot live cells with *p* > 1:

```
ind = np.argwhere( (cycle<100) & (op>1) ) .flatten()
plt.clf()
plt.scatter( cx[ind], cy[ind], c=op[ind], s=100 )
plt.title( 'Live cells with p>1 at t='\
+str(t/60) + 'hr', size=30)
plt.axis('image')
plt.colorbar()
```



# Now let's plot the oxygen

```
plt.clf()
mcds.get_substrate_names();

o2 = mcds.get_concentrations( 'oxygen' );
X,Y = mcds.get_2D_mesh();

plt.clf()
plt.contourf(X,Y,o2[:,:,0]);
```

## Now let's plot the oxygen with cells

```
plt.clf()
mcds.get_substrate_names();

o2 = mcds.get_concentrations( 'oxygen' );
X,Y = mcds.get_2D_mesh();
plt.contourf(X,Y,o2[:,:,0],cmap='spring');

plt.scatter( cx[live],cy[live],c=op[live],s=100);
plt.colorbar()
plt.axis('image')
plt.title( 'Live cells colored by p at t=' +str(t/60) + ' hr', size=30)

# let's plot dead cells as black
plt.scatter( cx[dead],dy[dead],c='k',s=100 );
```

# Now, let's do some time series analysis

- Let's get live and dead cell counts, mean *p* (in live cells). We need to loop overall simulation times

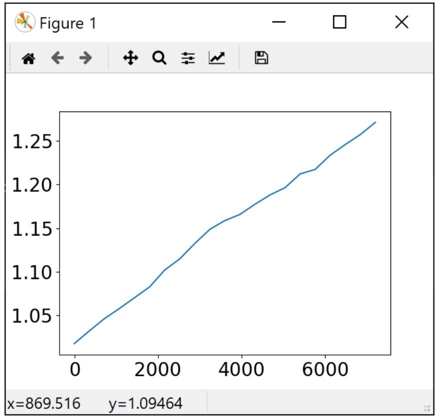
```
last_index = 20;
live_count = np.zeros( last_index+1 );
dead_count = np.zeros( last_index+1 );
mean_p = np.zeros( last_index+1 );
times = np.zeros( last_index+1 );
for n in range( 0,last_index+1 ):
     filename='output'+"%08i"%n+'.xml'
     mcds=pyMCDS(filename,'output')
     times[n]= mcds.get_time()
     cycle=mcds.data['discrete_cells']['cycle_model']
     p = mcds.data['discrete_cells']['oncoprotein']
     live = np.argwhere(cycle<100).flatten()
     dead = np.argwhere(cycle>=100).flatten()
     live_count[n] = len(live)
     dead_count[n] = len(dead)
     mean_p[n] = np.mean( p[live] )
```

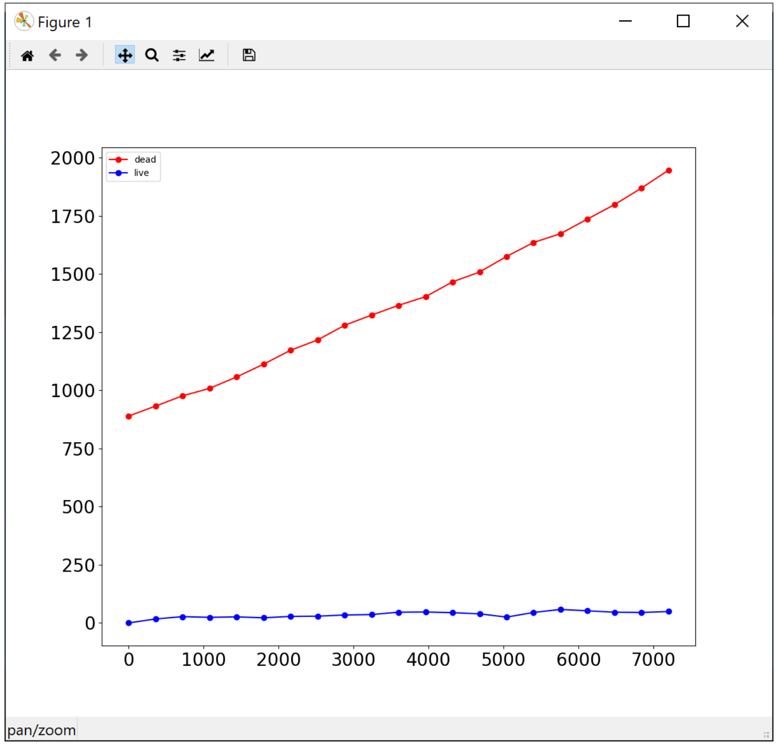
# Let's plot and get growth rates

```
plt.clf()
plt.plot( times, live_count , 'r-o' )
plt.plot( times, dead_count , 'b-o' );
plt.legend( {'live', 'dead' } )
```

```
poly=np.polyfit( times,np.log(live_count),1)
# growth rate is 0th element
# in units of 1/min
```

```
plt.clf()
plt.plot(times,mean_p);
```





# Let's work on data with multiple types

- Let's go and run the biorobots sample

```
make data-cleanup
make reset
make biorobots-sample
make
```

- Edit the config file to only run to 1440 min, and save every 240 min

```
./biorobots
```

## Let's load an intermediate time

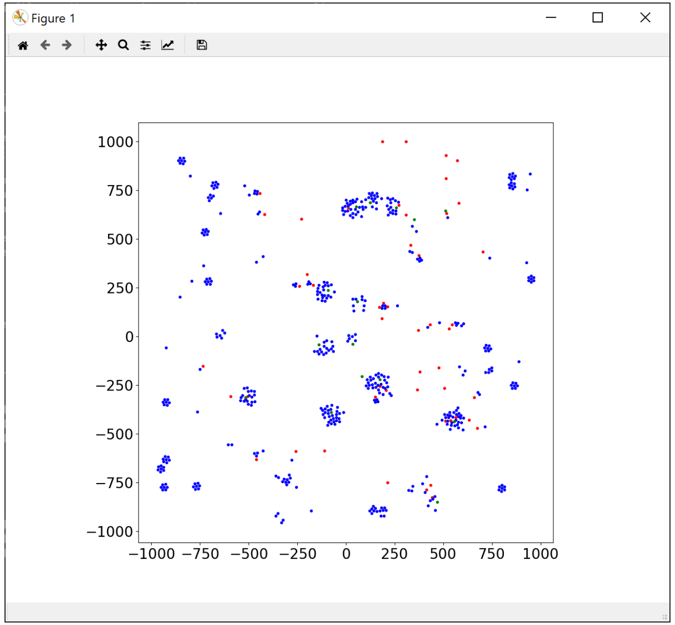
```
n = 3
filename='output'+"%08i"%n+'.xml'
mcds=pyMCDS(filename,'output')
t = mcds.get_time()
cell_type=mcds.data['discrete_cells']['cell_type']
cell_type=cell_type.astype(int)

ind0 = np.argwhere(cell_type==0).flatten();
ind1 = np.argwhere(cell_type==1).flatten();
ind3 = np.argwhere(cell_type==3).flatten();

cx = mcds.data['discrete_cells']['position_x']
cy = mcds.data['discrete_cells']['position_y']
```

# Let's plot each type a different color

```
plt.clf()
plt.scatter(cx[ind0],cy[ind0],c='r',s=10)
plt.scatter(cx[ind1],cy[ind1],c='b',s=10)
plt.scatter(cx[ind3],cy[ind3],c='g',s=10)
plt.axis('image');
```



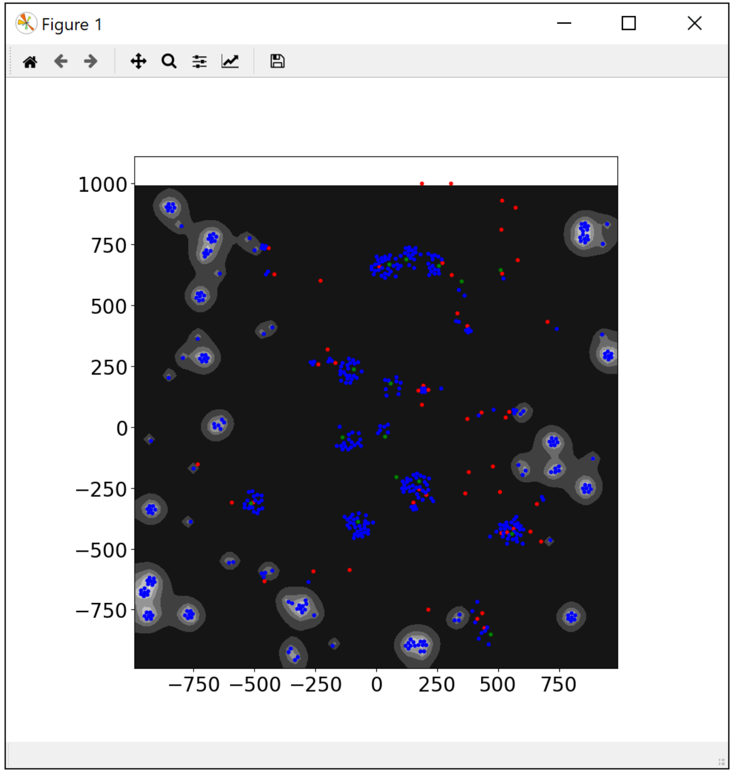
# Overlay on cargo signal

```
mcds.get_substrate_names();

cs = mcds.get_concentrations( 'cargo signal' );
X,Y = mcds.get_2D_mesh();

plt.clf()
plt.contourf(X,Y,cs[:,:,0],cmap='gray');


plt.scatter(cx[ind0],cy[ind0],c='r',s=10)
plt.scatter(cx[ind1],cy[ind1],c='b',s=10)
plt.scatter(cx[ind3],cy[ind3],c='g',s=10)
plt.axis('image');
```



## Part 2: Jupyter notebooks and nanoHUB

- Learn to generate a Jupyter notebook
- Run within the notebook, see what it does
- Learn process to deploy as nanoHUB app

# xml2jupyter

- As part of our NSF nanoBIO grant, we automated a process:
  - Create a C++ based model in PhysiCell
  - Make sure the XML config file is fully annotated with descriptions
  - run xml2jupyter to create a Jupyter notebook → adds GUI
  - create project on nanoHUB
  - submit github repository (with proper structure) to nanoHUB
  - voila! Shareable model!

**Reference:**

R. Heiland, D. Mishler, T. Zhang, E. Bower, and P. Macklin. xml2jupyter: Mapping parameters between XML and Jupyter widgets. *Journal of Open Source Software* 4(39):1408, 2019. DOI: 10.21105/joss.01408

## Let's get started!

- We'll practice on the biorobots sample (already built)

- First, let's get the code
  - https://github.com/PhysiCell-Tools/PhysiCell-Jupyter-GUI

- Then create a new empty repository for your soon-to-be-tool
  - I'll call mine pc4thanos (since I'll eventually make that model!)

- Clone the **PhysiCell-Tools/PhysiCell-Jupyter-GUI repo**

## Let's follow the steps. We need:

- The directory where your working project sits. For me: 
  **c:\temp\PhysiCell\**
- The directory where you intend to create your GUI. For me: 
  **C:\GitHub\pc4thanos\**
- Tool name. I'll call mine **pc4thanos**

## Now, let's run the script

- Go to the repo you cloned for the Jupyter GUI tool. For me:

```
cd C:\GitHub\PhysiCell-Tools\PhysiCell-Jupyter-GUI
```

- Run the script with those thing things we wrote down:

  python scriptname project_destination project_source new_tool_name

```
python setup_new_proj.py  C:\GitHub\pc4thanos\ c:\temp\PhysiCell\  pc4thanos
```

# Now we build

- Now, go into your new tool folder (for me c:\github\pc4thanos)

- Enter the **src** directory and **make**

```
cd c:\GitHub\pc4thanos
cd src
make
```

- Copy the new executable to the ../bin directory

```
cp myproj* ../bin
```

## Try your notebook!

- Change back up one level
- Run the notebook (it will match your tool name). For me:
  - cd ..
  - jupyter notebook pc4thanos.ipynb

- It should open up in a webbrowser and execute locally.
- Click "cell" and "run all".

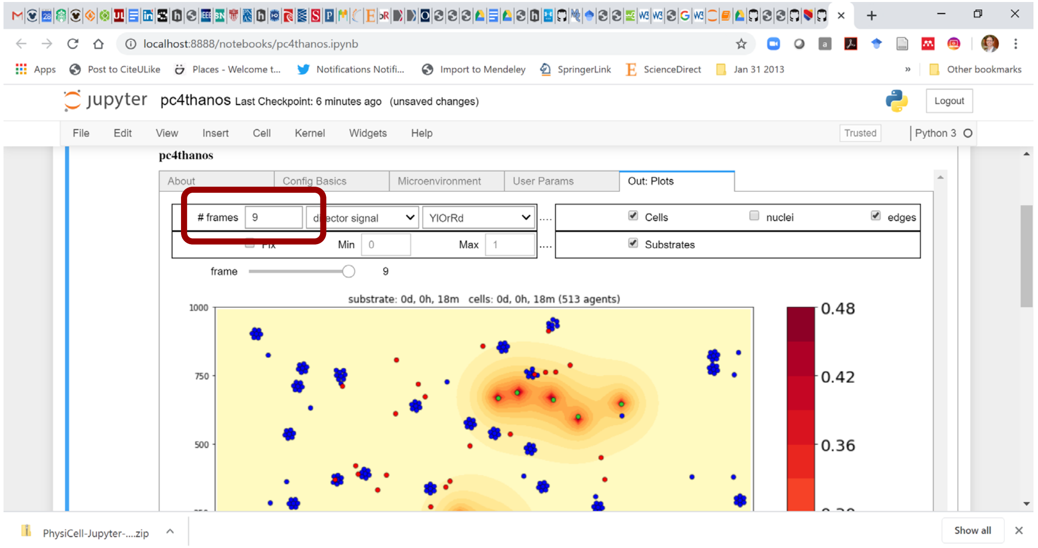
- Choose settings in the notebook and click the green "run" button

# Try your notebook! (2)

- The desktop version isn't quite as fancy as nanoHUB:
  - When you click "run", the GUI doesn't really show you it's working.
  - When you click the "plots" tab, you need to manually enter the number of frames ot scroll through the data

- But it's there!

- Now that works, commit and push your code.



pc4thanos Last Checkpoint: 6 minutes ago (unsaved changes)

pc4thanos

# frames 9

Min 0 Max 1

frame 9

substrate: 0d, 0h, 18m cells: 0d, 0h, 18m (513 agents)

# Get's get ready to deploy!

- On nanoHUB create a tool (use the same name you did earlier)
  - https://nanohub.org/tools/create
- Make sure to choose "Host GIT repository on GitHUB"
- Give the URL of your github repo (must be public). For me:
  - https://github.com/MathCancer/pc4thanos
- Click "register tool"

# Special for windows users

- Windows users need ot make sure the invoke script is executable. Go into the "middleware" directory of the tool repo
  - cd middleware
  - git update-index --chmod=+x invoke
  - git commit -m "Changing file permissions"
  - git push

## Tell nanoHUB you're ready!

- Once you're done, click the text that says:

  My code is committed, working, and ready to be installed

- The nanoHUB team (now in Sunny San Diego!) will manually compile and test install your tool. It's in their hands now!

- You will get email instructions on what to do next.

# Thanos vs Avengers

- **The story:**
  - **Civilian** cells migrate randomly, reproduce, and consume resources that regrow slowly
  - **Thanos** appears at a random time and place, and when unopposed, does "the snap": 50% of all non-Thanos cells immediately die.
    - If a few Avengers are nearby, Thanos stops attempt the snap and attacks the nearest Avenger.
    - If many Avengers are nearby, Thanos uses the space stone to teleport to a random location
  - **Avenger** cells randomly patrol. If they detect Thanos, they move towards him and attack.