

Slides, videos, links and more:

<https://github.com/physicell-training/ws2021>

Session 2: **PhysiCell** First Dive

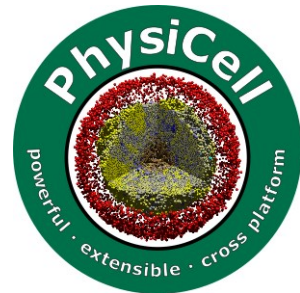


Paul Macklin, Ph.D.

 @MathCancer

PhysiCell Project

July 26, 2021



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 @PhysiCell

Goals

- Refresher: Sample and Template Projects
- Refresher: Project Structure
- Cells, Phenotype, and Cell Definitions
- Learn about general modeling workflow
 - **Basic** (Sessions 1, 2)
 - Intermediate (Session 5)
 - Full (Sessions 6-12)
- Populate, build, and run a basic model (Basic Workflow)
- Load and visualize data in Python

Key Background



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

Refresher: Sample and Template Projects

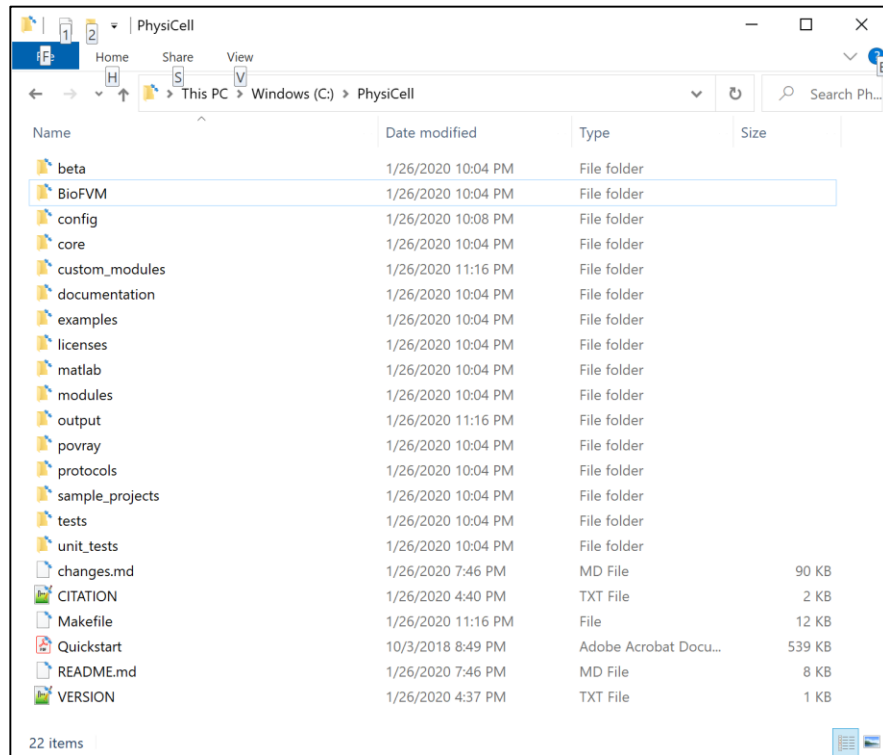
- Sample projects are pre-built projects that are bundled with PhysiCell
- **Key rules:**
 - make list-projects get a list of bundled projects
 - make [sample_name] populate a sample project
 - make compile the project
 - make data-cleanup clean up data for another run
 - make reset clear out the project to try another
- The **template** project is a good starting point for 2D and 3D projects.

Refresher: Project directory structure

- (key) directories:

- **./ (root):** main source, Makefile, and executable go here
- **./addons:** officially supported addons like PhysiBoSS and libRoadrunner
- **./beta:** for beta-testing (don't use)
- **./BioFVM:** diffusion solver
- **./config:** configuration files
- **./core:** PhysiCell core functions
- **./custom_modules:** put custom code for your project here.
- **./documentation:** user guide, etc.
- **./examples:** deprecated
- **./licenses:** yep
- **./matlab:** scripts and functions to load data in matlab
- **./modules:** standard add-ons for PhysiCell
- **./output:** where data are stored (by default, but can be changed)
- **./povray:** deprecated
- **./protocols:** instructions mostly for maintainers (e.g., release protocols)
- **./sample_projects:** where we add sample projects
- **./tests:** for automated testing (WIP)
- **./unit_tests:** for automated testing (WIP)

Most of your work will be in the **red** directories



Name	Date modified	Type	Size
beta	1/26/2020 10:04 PM	File folder	
BioFVM	1/26/2020 10:04 PM	File folder	
config	1/26/2020 10:08 PM	File folder	
core	1/26/2020 10:04 PM	File folder	
custom_modules	1/26/2020 11:16 PM	File folder	
documentation	1/26/2020 10:04 PM	File folder	
examples	1/26/2020 10:04 PM	File folder	
licenses	1/26/2020 10:04 PM	File folder	
matlab	1/26/2020 10:04 PM	File folder	
modules	1/26/2020 10:04 PM	File folder	
output	1/26/2020 11:16 PM	File folder	
povray	1/26/2020 10:04 PM	File folder	
protocols	1/26/2020 10:04 PM	File folder	
sample_projects	1/26/2020 10:04 PM	File folder	
tests	1/26/2020 10:04 PM	File folder	
unit_tests	1/26/2020 10:04 PM	File folder	
changes.md	1/26/2020 7:46 PM	MD File	90 KB
CITATION	1/26/2020 4:40 PM	TXT File	2 KB
Makefile	1/26/2020 11:16 PM	File	12 KB
Quickstart	10/3/2018 8:49 PM	Adobe Acrobat Docu...	539 KB
README.md	1/26/2020 7:46 PM	MD File	8 KB
VERSION	1/26/2020 4:37 PM	TXT File	1 KB

Cells (1)

- Cells are the key entity in PhysiCell.
- Each cell keeps track of:
 - Type
 - ID
 - Position and velocity
 - State
 - Phenotype
 - ♦ Intracellular model and data are included here.
 - Custom data

Cells (2)

- Cells have built-in techniques for:
 - Division
 - Death
 - Changing type
 - Accessing / sampling the microenvironment
 - Finding nearby cells
 - Ingesting cells
 - And more behaviors via phenotype (Sessions 3-4)

Key cell information

- Each cell agent is a member of the `Cell` class.
- Some key data:
 - `std::string type_name` // human-readable name of cell type
 - `int type` // machine-readable unique integer identifier for cell type
 - `int ID` // cell agent's unique integer identifier. (different for each cell)
 - `std::vector<double> position` // the cell's current position (**never write this!**)
 - `std::vector<double> velocity` // the cell's current velocity
 - `cell_state` // things like size, pressure, and cells in contact
 - `phenotype` // behavioral properties / state (Sessions 3-4)
 - `custom_data` // custom scalar and vector data (Session 6)
 - `functions` // list of key cell functions (Session 7)

Future refinement:

Each cell should have a pointer to its `Cell_Definition`

Cell state

- Each **Cell** has an instance of **Cell_State** called **state**:
 - `std::vector<Cell*> attached_cells:` (Sessions 9-10)
 - ♦ Use this for your own custom storage of interacting cells for contact-based interactions
 - `std::vector<Cell*> neighbors:` (Sessions 9-10)
 - ♦ a vector of pointers to all (mechanically interacting) neighbor cells.
 - ♦ Automatically updated to include all cells with non-zero mechanical interactions
 - `std::vector<double> orientation:`
 - ♦ Used during cell division: division plane passes through this vector.
 - ♦ A unit vector (length 1) directed from the cell's base to its apex
 - ♦ Cell division places daughter cells (randomly) perpendicular to this vector
 - ♦ In 2D, orientation = [0,0,1] so that daughter cells stay in plane
 - `double simple_pressure:` (Sessions 9-10)
 - ♦ a (normalized) measure of forces exerted by nearby adhered cells
 - ♦ in a 3-D, fully confluent (packed) tissue, 12 neighbors, and `simple_pressure` = 1
 - ♦ in a 2-D, fully confluent (packed) tissue, 6 neighbors, and `simple_pressure` = 0.5

Cell phenotype

- One of the most critical data elements in a PhysiCell Cell is ***phenotype***
- Hierarchically organize key behavioral elements:
 - Phenotype (Sessions 3-4)
 - ♦ **cycle**: advancement through a cell cycle model
 - ♦ **death**: one or more types of cell death
 - ♦ **volume**: cell's volume regulation
 - ♦ **geometry**: cell's radius and surface area
 - ♦ **mechanics**: adhesion and resistance to deformation ("repulsion")
 - ♦ **motility**: active motion (other than "passive" mechanics)
 - ♦ **secretion**: both release and uptake of chemical substrates. Interfaces with BioFVM
 - ♦ **molecular**: a place to store internalized substrates (Sessions 11-12)
 - ♦ **intracellular**: a place for intracellular models (Sessions 11-12)

Phenotype-centric programming

- The core cell behaviors are implemented:
 - Cell cycling (with user-selectable models)
 - Cell death
 - Cell adhesion / repulsion
 - Cell motility
 - Cell secretion / uptake
- Modelers can focus on writing functions that control these behaviors.
- This is **phenotype-centric programming**.

Cell Definitions

- A **Cell Definition** is a convenient way to set the parameters and functions for a whole class of cells
 - Users can instantiate cells of a specific type using `create_cell(A_cell_defn)`
 - With no argument, new cells use the `cell_defaults` definition
- Best practice: set up the `cell_defaults` definition first. Copy this to create other cell types
- Tip: Refer back to the phenotype in your agent's cell definition as a reference parameter set (i.e., to get the initial parameter values)

More on this in Session 5.

Modeling Workflows



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

PhysiCell Modeling Workflows

- There are three typical modeling workflows in PhysiCell
 - **Basic** (Introduced in Session 1 pre-workshop and 2 today)
 - ♦ Build existing projects, change parameter values, and run
 - **Intermediate** (Introduced in Session 5 today)
 - ♦ Build your own models based on the template project
 - ♦ All model setup in a GUI (no modification of C++)
 - **Full** (Introduced in Session 7 tomorrow)
 - ♦ Enhance an intermediate model with custom C++ to implement cell hypotheses / rules

Basic modeling workflow



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

Basic modeling workflow

Suitable for running a built-in project with minor changes to parameters.

- Populate and build a project
- Edit settings
- Run
- View results

Choose, populate, and build a project

- Get list of sample projects:
 - `make list-projects`
- Populate the heterogeneity sample:
 - `make heterogeneity-sample`
- Compile the project
 - `make`

Edit settings

- Open the settings file:
 - **`./config/PhysiCell_settings.xml`**
- Let's change:
 - Change domain to $[-400,400] \times [-400,400]$
 - Reduce max simulation time to 2160 minutes
 - Save full data ever 360 minutes
 - Set oncoprotein standard deviation to 3 (increase heterogeneity)
 - Set the max oncoprotein value to 10 (mean + 3 standard deviations)

Edit settings: XML

- Open `./config/PhysiCell_settings.xml`
- Major sections:
 - **domain** -- how big of a region to simulate
 - **overall** -- how long to simulate, time step sizes
 - **parallel** -- OpenMP settings
 - **save** -- how often to save SVG images and full data
 - **microenvironment** -- settings on diffusing substrates
 - **cell_definitions** -- set baseline cell properties
 - **initial_conditions** -- options to set initial cell positions via CSV
 - **user_parameters** -- model-specific settings

Edit settings: Domain size

- 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>-400</x_min>
    <x_max>400</x_max>
    <y_min>-400</y_min>
    <y_max>400</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>
```

Edit settings: User parameters

- Let's also look at the **user_parameters** block
 - Let's change the oncoprotein standard deviation (**oncoprotein_sd**) to 3 (more variation)
 - Let's change the max oncoprotein (**oncoprotein_max**) to mean + 3 sds = $1 + 9 = 10$

```
<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">3.0</oncoprotein_sd>
  <oncoprotein_min type="double" units="dimensionless">0.0</oncoprotein_min>
  <oncoprotein_max type="double" units="dimensionless">10</oncoprotein_max>
  <random_seed type="int" units="dimensionless">0</random_seed>
</user_parameters>
```

Edit settings: Save settings

- Let's look at the **overall** block
 - Set max time to 1.5 days = $1.5 \times 24 \times 60 = 2160$ minutes

```
<overall>
  <max_time units="min">2160</max_time> <!-- 36 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

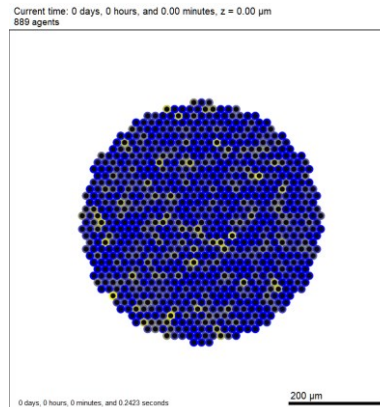
```
<save>
  <folder>output</folder> <!-- use . for root -->
  <full_data>
    <interval units="min">360</interval>
    <enable>true</enable>
  </full_data>
```

Run and View Results: Snapshots

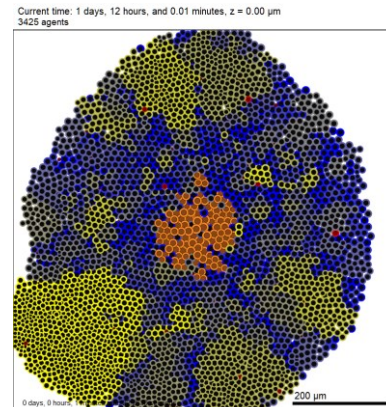
- run:
 - **./heterogeneity** (MacOS or Linux)
 - **heterogeneity.exe** (Windows)

- Look in output:
 - Look at snapshot SVG files
 - Look at **legend.svg**
 - ♦ (Not much to see on this example)


- Convert snapshots to JPEG:
 - **make jpeg** (results: output/snapshot00000000.jpg ...)



initial.svg



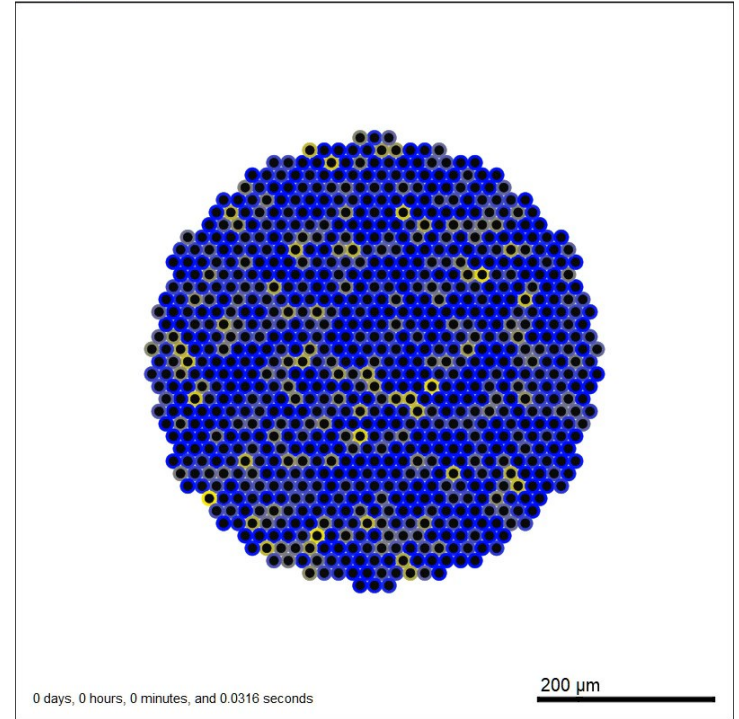
final.svg

 cancer cell
legend.svg

View results: GIF and movie

- Make an animated GIF:
 - **make gif** (result: output/out.gif)
- Make an mp4 movie
 - **make movie** (result: output/out.mp4)

Current time: 0 days, 0 hours, and 0.00 minutes, z = 0.00 μm
890 agents



Loading data in Python



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

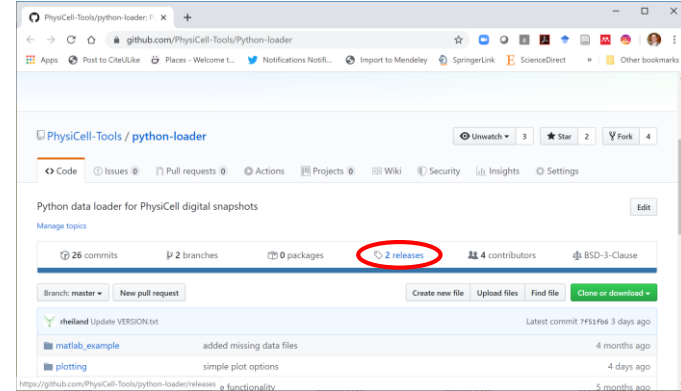
PhysiCell Project

PhysiCell.org

 **@PhysiCell**

Let's get ready to load in Python

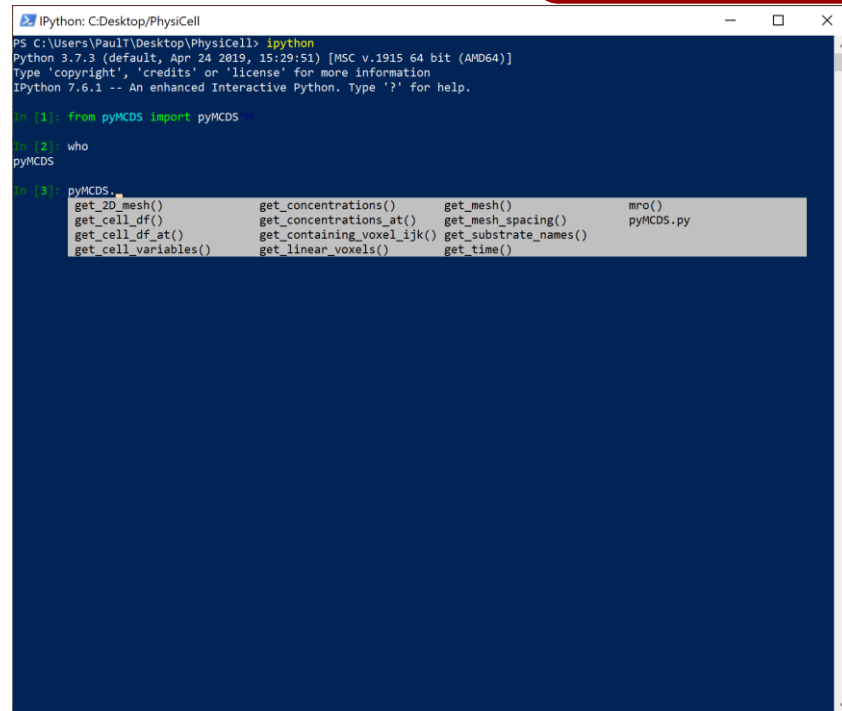
- We'll go to Python-loader and get the source:
 - <https://github.com/PhysiCell-Tools/Python-loader>
- Get the latest release:
 - Click on "releases" link
 - Click the green "clone or download" button
 - ♦ (For simplicity, I'm using "download ZIP" option)
- Copy the following Python file (end in .py) to the root of PhysiCell
 - pyMCDS



Let's get started in ipython

Jupyter Notebook Code Section 1

- Start ipython (interactive python)
 - `ipython3 --pylab`
- OR: download & start the Jupyter notebook
 - `jupyter notebook Session2_heterogeneity.ipynb`
- 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 = MultiCellIDS, our multicellular data standard



```
IPython: C:\Desktop\PhysiCell
PS C:\Users\Paul\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

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_idx()  get_substrate_names()
get_cell_variables()  get_linear_voxels()      get_time()
```

Session 2 Jupyter notebook: (save to root of PhysiCell directory)

https://raw.githubusercontent.com/physicell-training/ws2021/main/code/Session_2/Session2_heterogeneity.ipynb

Let's load a single time

Jupyter Notebook
Code Section 2

- 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 cell data

Jupyter Notebook
Code Section 3

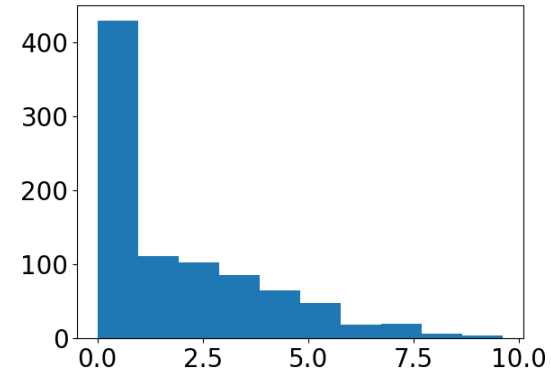
- First, let's find out the mean value of the oncoprotein

- `np.mean(mcds.data['discrete_cells']['oncoprotein'])`

`Out[61]: 1.8305931655741`

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

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



- Now, let's plot a histogram

- `plt.hist(mcds.data['discrete_cells']['oncoprotein'])`

Let's plot the cells

Jupyter Notebook
Code Section 4

- 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.

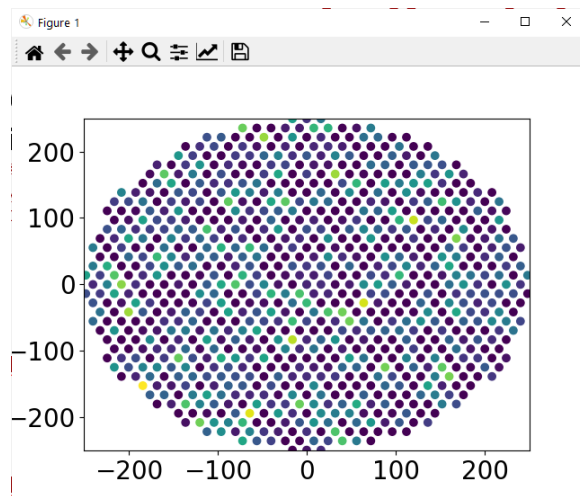
- Note: these are not plotting by the **physical** cell size

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

- If there are some cells out of range, fix the axes:

```
plt.axis([-250,250,-250,250])
```

- 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=30 )
```

- Make sure aspect ratio is right:

```
plt.axis( 'image' )  
plt.axis( [-250,250,-250,250] )
```

- Now, let's add a colorbar

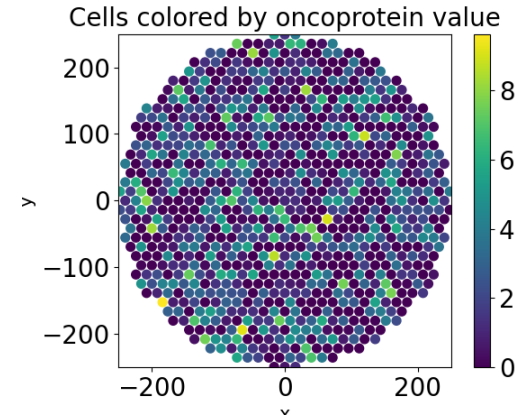
```
plt.colorbar()
```

- Now, let's add labels

```
plt.title( 'Cells colored by oncoprotein value' , size=20)  
plt.xlabel( 'x' , size=15 )  
plt.ylabel( 'y', size=15 )
```

The right value will vary based on your screen resolution, zoom, and window size.

This will take some experimentation!

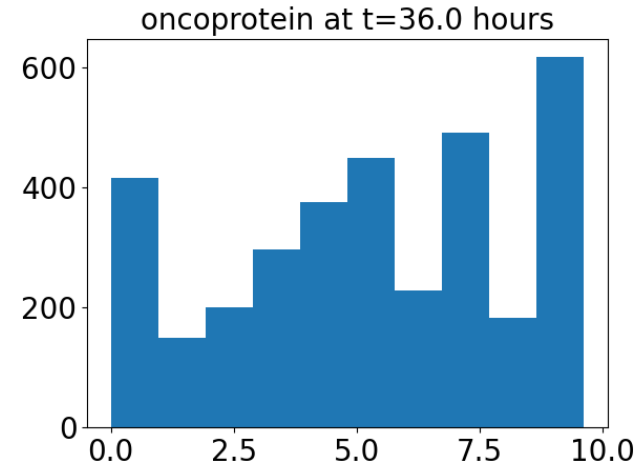


Jupyter Notebook
Code Section 5

Let's load another time

Jupyter Notebook
Code Section 6

```
mcds = pyMCDS('output00000006.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=20)
```



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

@PhysiCell

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()
```

Jupyter Notebook
Code Section 7

Let's work with these

Jupyter Notebook
Code Section 8

- Live and dead cell counts

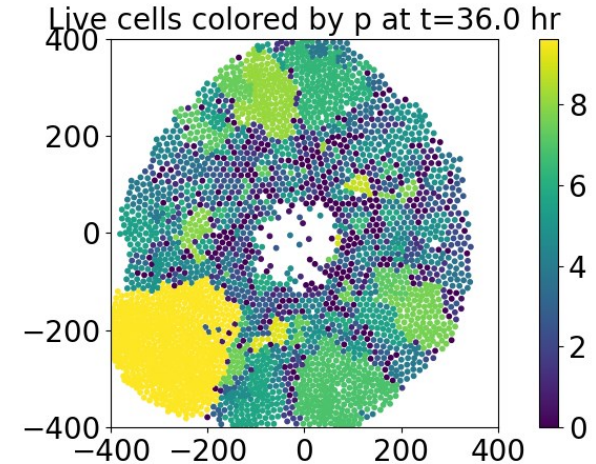
```
n_live = len( live ); print( n_live )  
n_dead = len( dead ); print( n_dead );
```

- 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=10);  
plt.colorbar()  
plt.axis('image')  
plt.axis([-400,400,-400,400])  
plt.title( 'Live cells colored by p at t=' +str(t/60) + ' hr',size=20)
```



More data loading

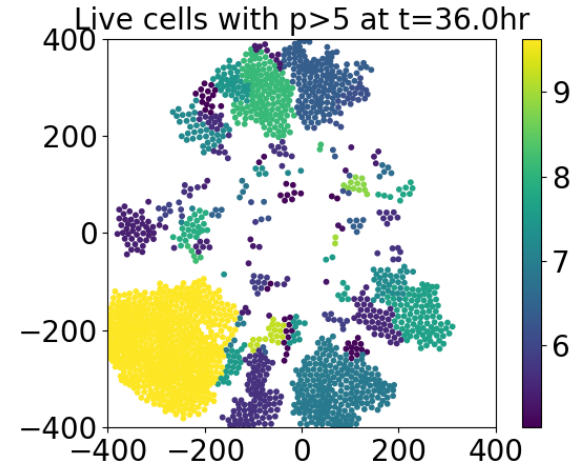
Let's do a fancier search

Jupyter Notebook
Code Section 8a

- Only plot live cells with $p > 5$:

```
ind = np.argwhere( (cycle<100) & (op>5) ) .flatten()
plt.clf()
plt.scatter( cx[ind], cy[ind], c=op[ind], s=10 )
plt.title( 'Live cells with p>5 at t='\
+str(t/60) + 'hr', size=20)
plt.axis('image')
plt.axis([-400,400,-400,400])
plt.colorbar()
```

- **Note:** The best circle size ($s=10$) will vary based on your desktop resolution, zoom and window size. You will need to experiment.



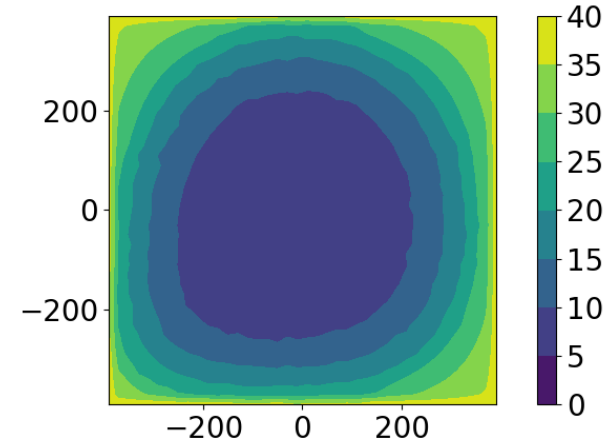
Now let's plot the oxygen

Jupyter Notebook
Code Section 9

```
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]);
plt.colorbar()
plt.axis('image')
```



Now let's plot the oxygen with cells

```
circle_size = 10
```

```
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.colorbar()
```

```
plt.scatter( cx[live],cy[live],c=op[live],s=circle_size);
```

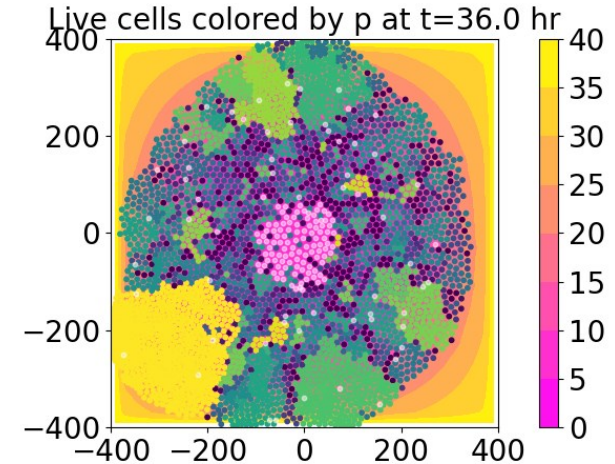
```
plt.axis('image')
```

```
plt.axis([-400,400,-400,400])
```

```
plt.title( 'Live cells colored by p at t=' +str(t/60) + ' hr', size=20)
```

```
# let's plot dead cells as white and transparent
```

```
plt.scatter( cx[dead],cy[dead],c='w',s=circle_size, alpha=0.5 );
```



**Jupyter Notebook
Code Section 10**

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 = 6;
live_count = np.zeros( last_index+1 );
dead_count = np.zeros( last_index+1 );
mean_p = np.zeros( last_index+1 );
std_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] )
    std_p[n] = np.std( p[live] )
```

**Jupyter Notebook
Code Section 11**

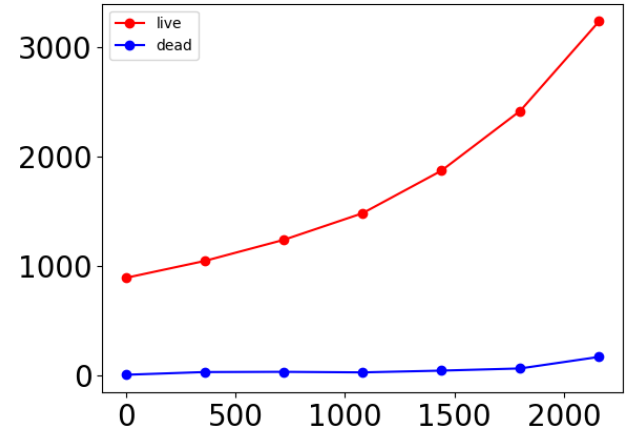
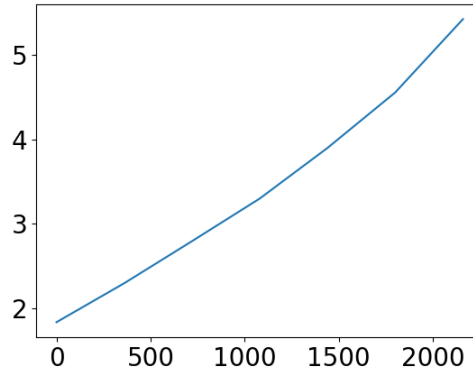
Let's plot and get growth rates

Jupyter Notebook
Code Section 12

```
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)  
print( poly[0] )  
# growth rate is 0th element  
# in units of 1/min  
# 0.0005928392815655603
```

```
plt.clf()  
plt.plot(times,mean_p);  
# mean increases rapidly  
# due to selection processes
```



Cleanup

- Clear out data (to prepare for another run)
 - **make data-cleanup** (clears all out of /output)
- Reset to a clean slate (e.g., to start another project)
 - **make reset** (depopulates custom files, restores Makefile)



LUDDY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 **@PhysiCell**

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:

- run to 720 min
 - save full data ever 240 min
 - save SVGs every 30 min
- ```
./biorobots
```

```
<save>
 <folder>output</folder> <!-- use . for root -->

 <full_data>
 <interval units="min">240</interval>
 <enable>true</enable>
 </full_data>

 <SVG>
 <interval units="min">45</interval>
 <enable>true</enable>
 </SVG>

 <legacy_data>
 <enable>false</enable>
 </legacy_data>
</save>
```



**LUDDY**

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

PhysiCell Project

PhysiCell.org

 @PhysiCell

# Let's load the last time

**Jupyter Notebook  
Code Section 13**

```
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)

ind1 = np.argwhere(cell_type==1).flatten(); # director
ind2 = np.argwhere(cell_type==2).flatten(); # cargo
ind3 = np.argwhere(cell_type==3).flatten(); # worker

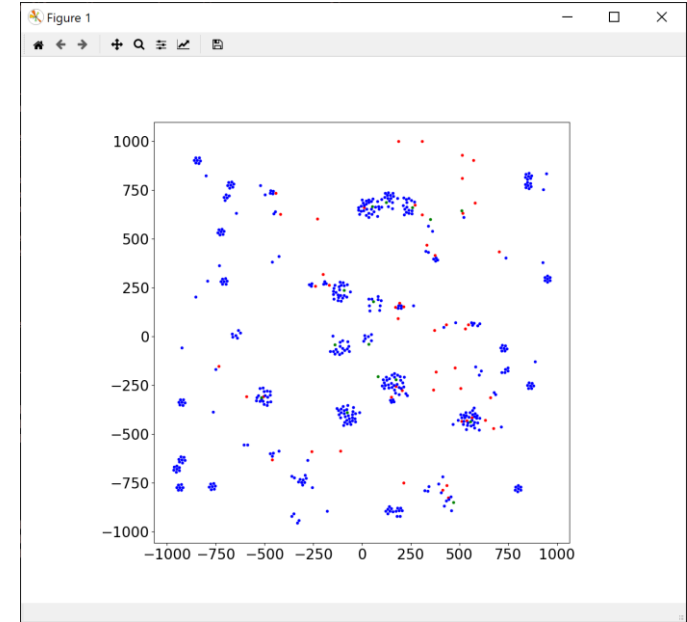
cx = mcds.data['discrete_cells']['position_x']
cy = mcds.data['discrete_cells']['position_y']
```

# Let's plot each type a different color

```
circle_size=20
```

```
plt.clf()
plt.figure(figsize=(15,15))
plt.scatter(cx[ind1],cy[ind1],c='g',s=circle_size)
plt.scatter(cx[ind2],cy[ind2],c='b',s=circle_size)
plt.scatter(cx[ind3],cy[ind3],c='r',s=circle_size)
plt.axis('square');
```

Jupyter Notebook  
Code Section 14



# Overlay on top of the cargo signal

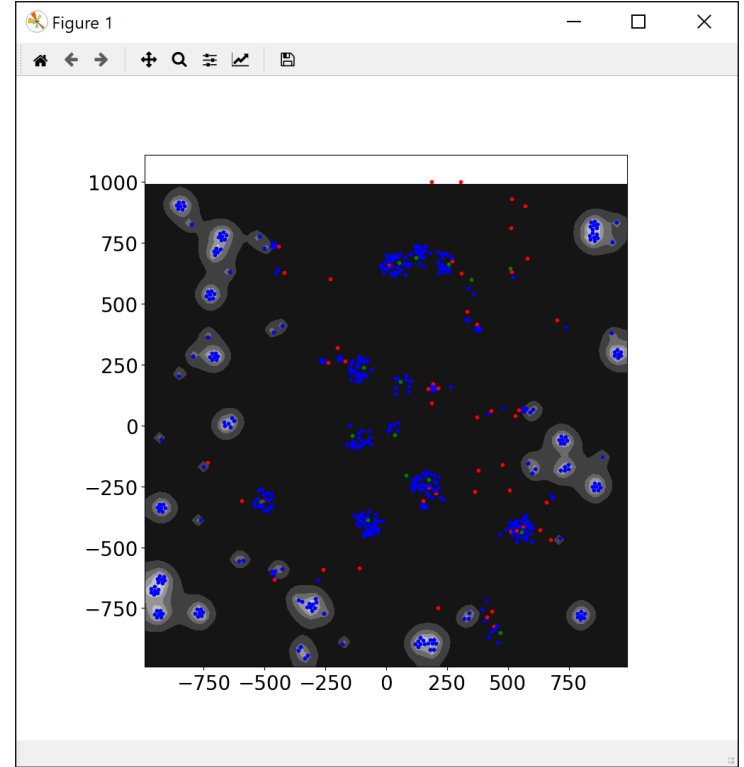
```
mcds.get_substrate_names();

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

plt.clf()
plt.figure(figsize=(15,15))
plt.contourf(X,Y,cs[:, :, 0], cmap='gray');

plt.scatter(cx[ind1],cy[ind1],c='g',s=circle_size)
plt.scatter(cx[ind2],cy[ind2],c='b',s=circle_size)
plt.scatter(cx[ind3],cy[ind3],c='r',s=circle_size)
plt.axis('image');
```

**Jupyter Notebook**  
**Code Section 15**



# Cleanup

- Clear out data (to prepare for another run)
  - **make data-cleanup** (clears all out of /output)
- Reset to a clean slate (e.g., to start another project)
  - **make reset** (depopulates custom files, restores Makefile)

# Intermediate modeling workflow

Suitable for creating a new PhysiCell model without writing custom C++ (no dynamical phenotype changes)

- **Plan the model**
- Populate and build the template project
- Edit configuration with Model Builder GUI
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
- Run
- View results

# Looking Forward: Full modeling workflow

Suitable for creating a new PhysiCell model with custom C++ to drive dynamical phenotype changes

- Plan the model
- Populate a project
- Edit configuration Model Builder GUI
  - Edit domain
  - Edit microenvironment
  - Edit cell definitions
  - **Add custom variables**
  - **Add custom parameters**
- **Edit custom modules:**
  - **Declare functions in custom.h**
  - **Implement functions in custom.cpp**
  - **Assign functions to cell definitions**
- **Edit initial cell placement**
- **Edit cell coloring function**
- Build
- Run
- View results



# Funding Acknowledgements



## PhysiCell Development:

- Breast Cancer Research Foundation
- Jayne Koskinas Ted Giovanis Foundation for Health and Policy
- National Cancer Institute (U01CA232137)
- National Science Foundation (1720625)

## Training Materials:

- Administrative supplement to NCI U01CA232137 (Year 2)