BIOENGINEERING

UNIVERSITY *of* WASHINGTON

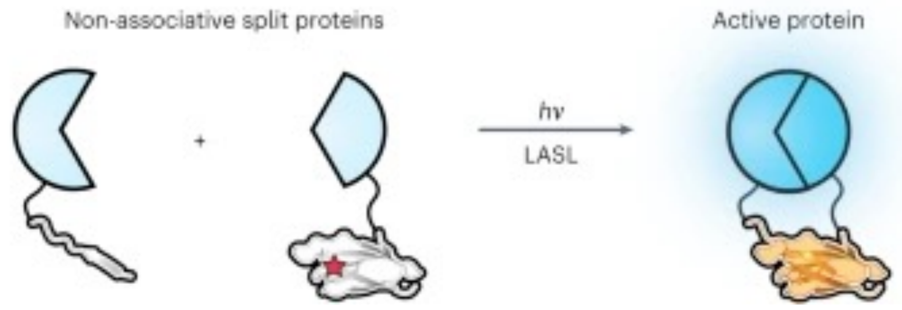
Photosimulator

Brizzia G. Munoz Robles

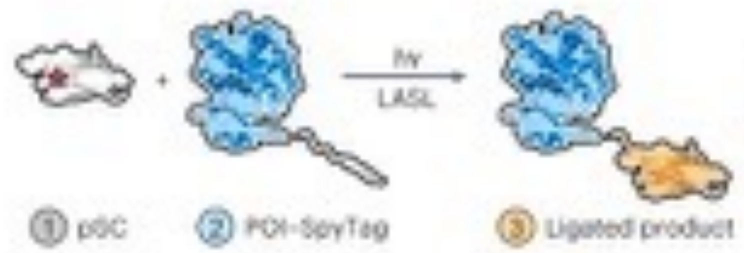
Graduate Student Software Project

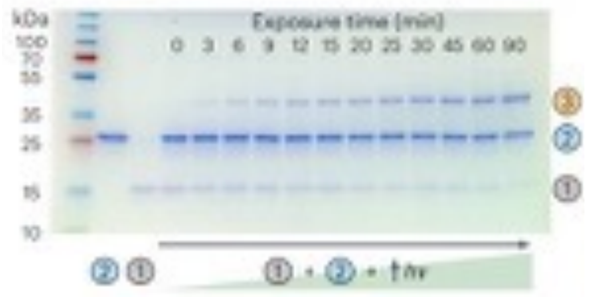
BIOEN 537

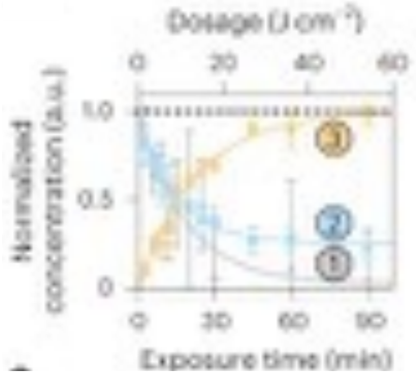
Background: Modeling Photokinetics



- ✓ Genetically encoded
- ✓ Functional protein activation in living cells
- ✓ Micrometre-scale patterning resolution in four dimensions
- ✓ Rapid and irreversible activation
- ✓ Covalent and stable product
- ✓ Versatile plug-and-play approach

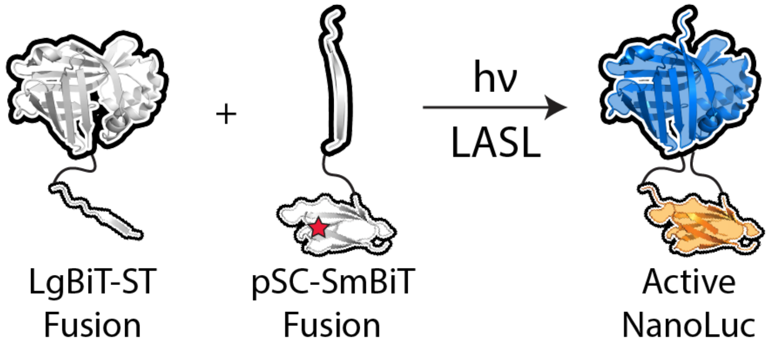


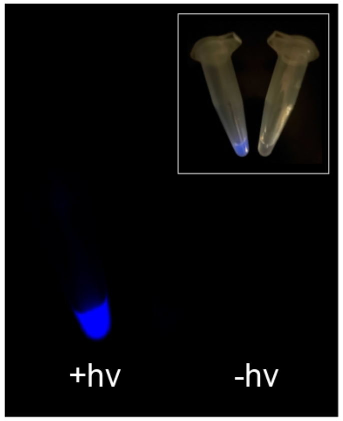


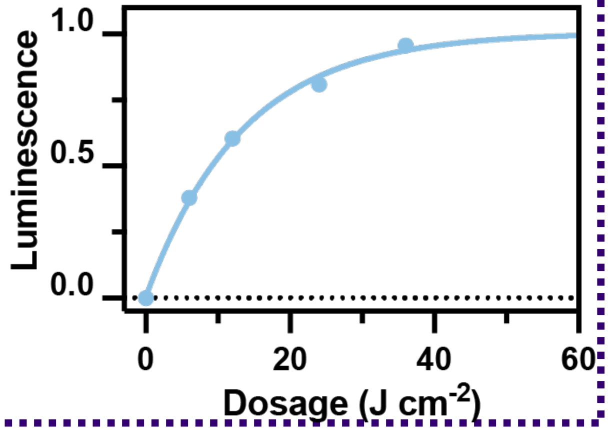


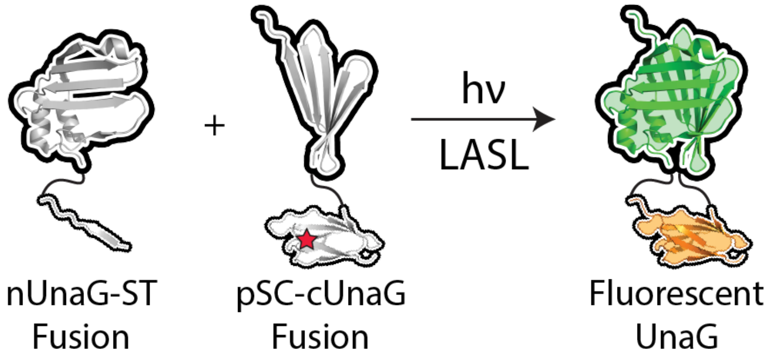
Munoz-Robles, B.G.*, **Ruskowitz, E.R**.*, Strange, A.C., Butcher, C.H., Kurniawan, S., Filteau, J.R. & DeForest, C.A**.** Spatiotemporal Functional Assembly of Split Protein Pairs through a Light-Activated SpyLigation. *Nature Chemistry* (2023) ***Equal contribution**

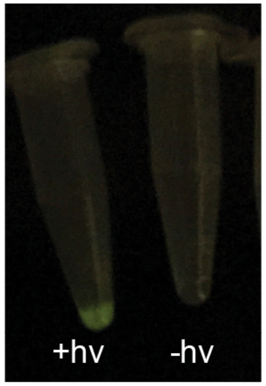
Background: Modeling Photokinetics

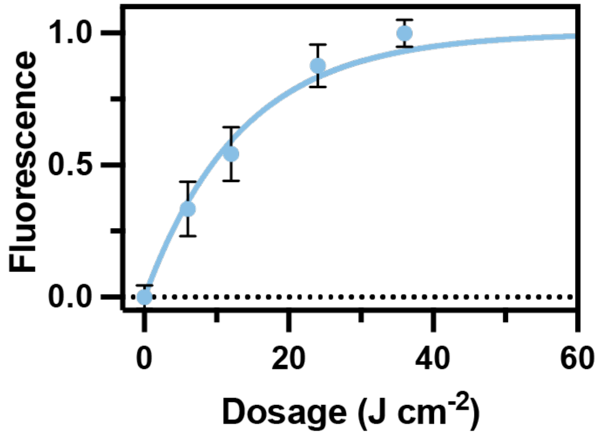












Munoz-Robles, B.G.*, **Ruskowitz, E.R.***, Strange, A.C., Butcher, C.H., Kurniawan, S., Filteau, J.R. & DeForest, C.A. Spatiotemporal Functional Assembly of Split Protein Pairs through a Light-Activated SpyLigation. *Nature Chemistry* (2023) ***Equal contribution**

Use Case I: PhotokineticAnalysis()

Use Case I: Theoretical photokinetic data:

- The user can input theoretical x values (in any units) and kinetic rate constant (unit must match x value unit) to obtain expected y values when input into the equation:

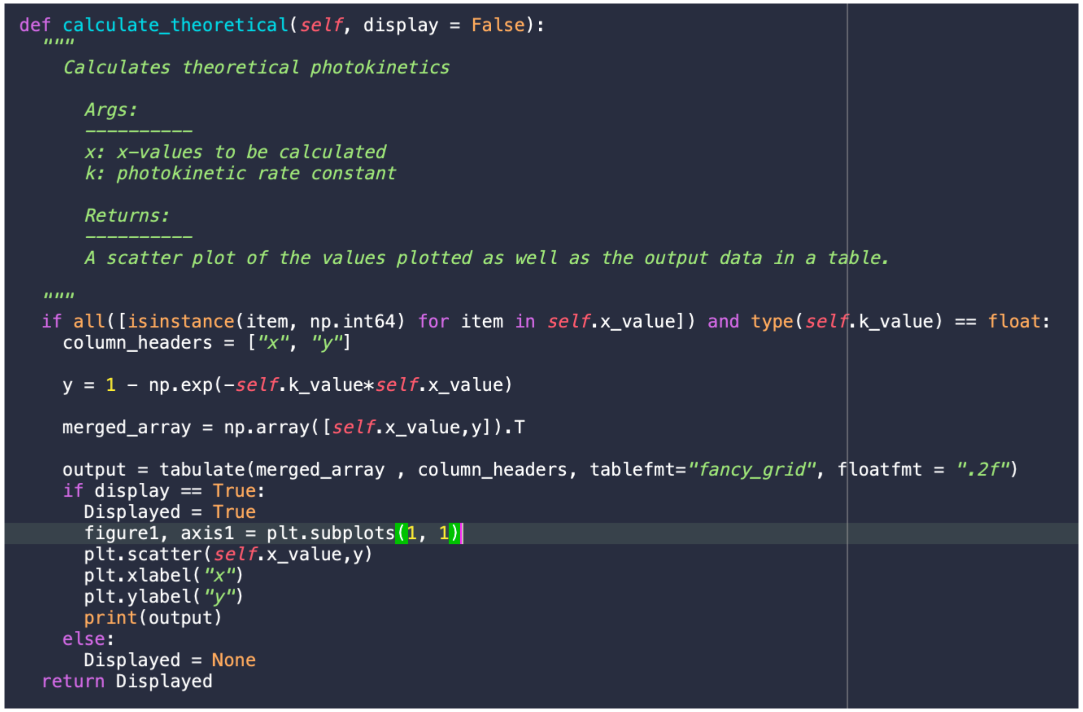
Design:

- **PhotokineticAnalysis.check()**
  a. **Purpose:** checks that the inputs for the calculate_theoretical function are correct
- **PhotokineticAnalysis.calculate_theoretical()**
  a. **Purpose:** Calculates theoretical photouncaging values using the following first order kinetic equation for photouncaging:

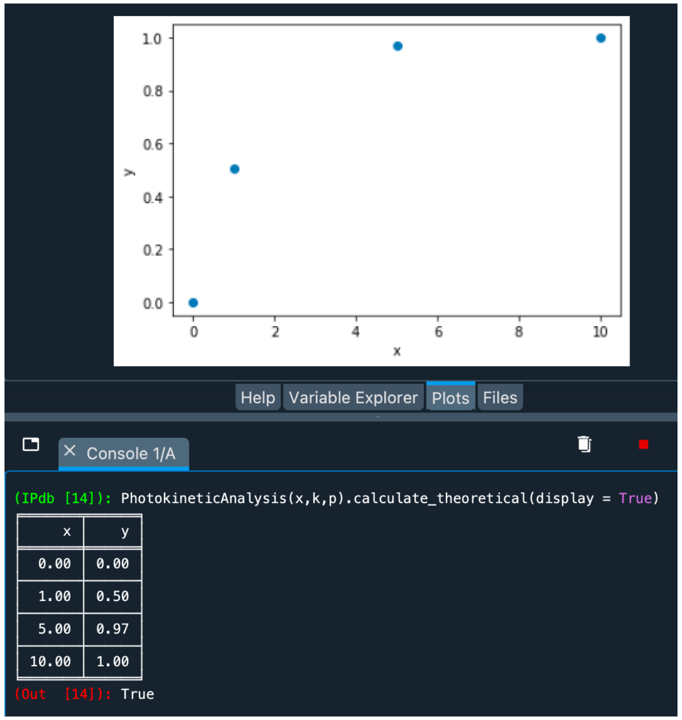
$$\boldsymbol{Y = Y_o + (plateau - Y_o)(1 - \exp(-k * x))}$$

  a. **Input:**
     i. x values
     ii. Kinetic constant
  b. **Output:**
     i. Kinetic rate constant with standard deviation
     ii. Table of x and corresponding y values (normalized to 1)
     iii. Graph of theoretical data

PhotokineticAnalysis.calculating_theoretical()



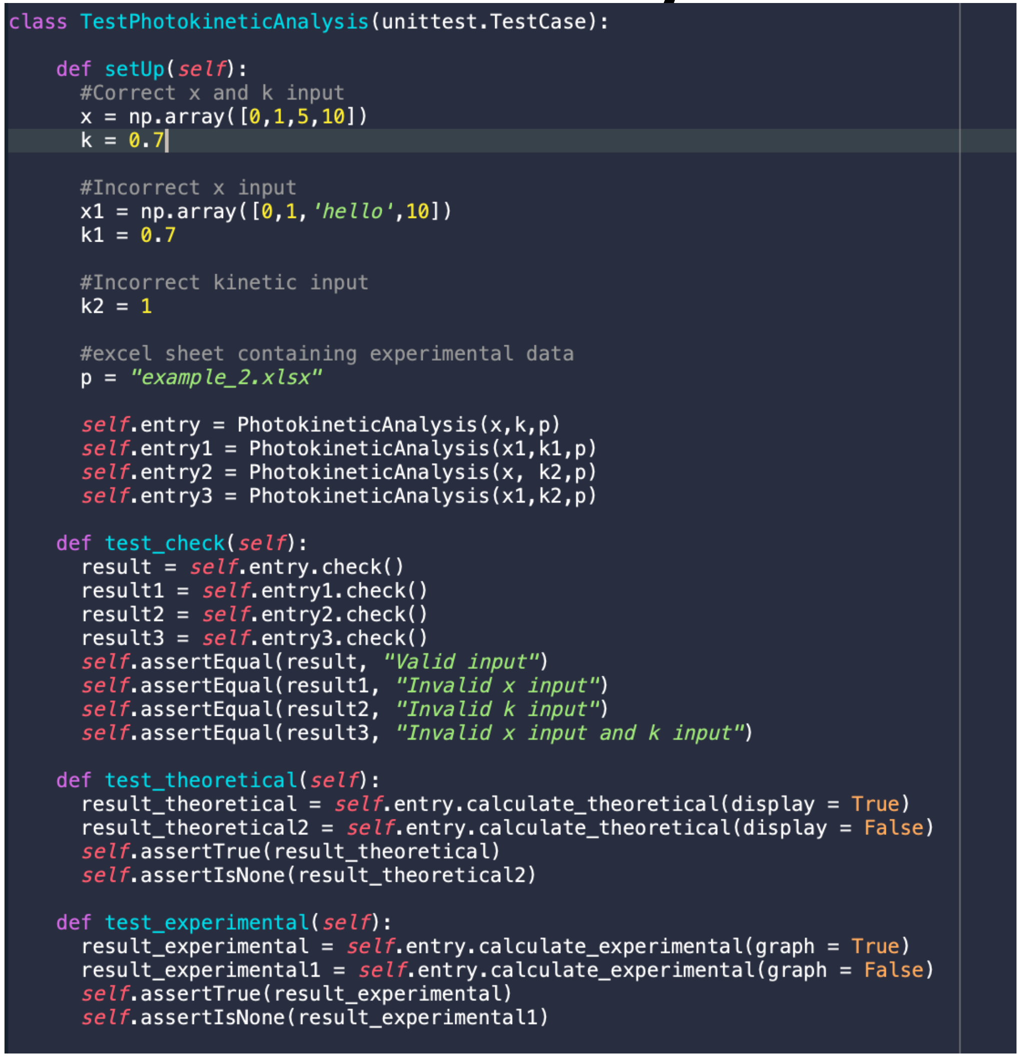
```
def calculate_theoretical(self, display = False):
  """
    Calculates theoretical photokinetics

      Args:
      ----------
      x: x-values to be calculated
      k: photokinetic rate constant

      Returns:
      ----------
      A scatter plot of the values plotted as well as the output data in a table.

  """
  if all([isinstance(item, np.int64) for item in self.x_value]) and type(self.k_value) == float:
    column_headers = ["x", "y"]

    y = 1 - np.exp(-self.k_value*self.x_value)

    merged_array = np.array([self.x_value,y]).T

    output = tabulate(merged_array , column_headers, tablefmt="fancy_grid", floatfmt = ".2f")
    if display == True:
      Displayed = True
      figure1, axis1 = plt.subplots(1, 1)
      plt.scatter(self.x_value,y)
      plt.xlabel("x")
      plt.ylabel("y")
      print(output)
    else:
      Displayed = None
  return Displayed
```



```
(IPdb [14]): PhotokineticAnalysis(x,k,p).calculate_theoretical(display = True)
╒═══════╤══════╕
│     x │    y │
╞═══════╪══════╡
│  0.00 │ 0.00 │
├───────┼──────┤
│  1.00 │ 0.50 │
├───────┼──────┤
│  5.00 │ 0.97 │
├───────┼──────┤
│ 10.00 │ 1.00 │
╘═══════╧══════╛
(Out  [14]): True
```

This is very useful to see what the expected curve/data should look like prior to running an experiment. Sometimes in the DeForest lab we run two to three time points instead of a larger range and it is good to anticipate what the next value should be, and if the experimental data does not match, typically this indicates there is a problem with the experimental set up.

TestPhotokineticAnalysis



```python
class TestPhotokineticAnalysis(unittest.TestCase):

    def setUp(self):
      #Correct x and k input
      x = np.array([0,1,5,10])
      k = 0.7

      #Incorrect x input
      x1 = np.array([0,1,'hello',10])
      k1 = 0.7

      #Incorrect kinetic input
      k2 = 1

      #excel sheet containing experimental data
      p = "example_2.xlsx"

      self.entry = PhotokineticAnalysis(x,k,p)
      self.entry1 = PhotokineticAnalysis(x1,k1,p)
      self.entry2 = PhotokineticAnalysis(x, k2,p)
      self.entry3 = PhotokineticAnalysis(x1,k2,p)

    def test_check(self):
      result = self.entry.check()
      result1 = self.entry1.check()
      result2 = self.entry2.check()
      result3 = self.entry3.check()
      self.assertEqual(result, "Valid input")
      self.assertEqual(result1, "Invalid x input")
      self.assertEqual(result2, "Invalid k input")
      self.assertEqual(result3, "Invalid x input and k input")

    def test_theoretical(self):
      result_theoretical = self.entry.calculate_theoretical(display = True)
      result_theoretical2 = self.entry.calculate_theoretical(display = False)
      self.assertTrue(result_theoretical)
      self.assertIsNone(result_theoretical2)

    def test_experimental(self):
      result_experimental = self.entry.calculate_experimental(graph = True)
      result_experimental1 = self.entry.calculate_experimental(graph = False)
      self.assertTrue(result_experimental)
      self.assertIsNone(result_experimental1)
```

Use Case II: PhotokineticAnalysis()

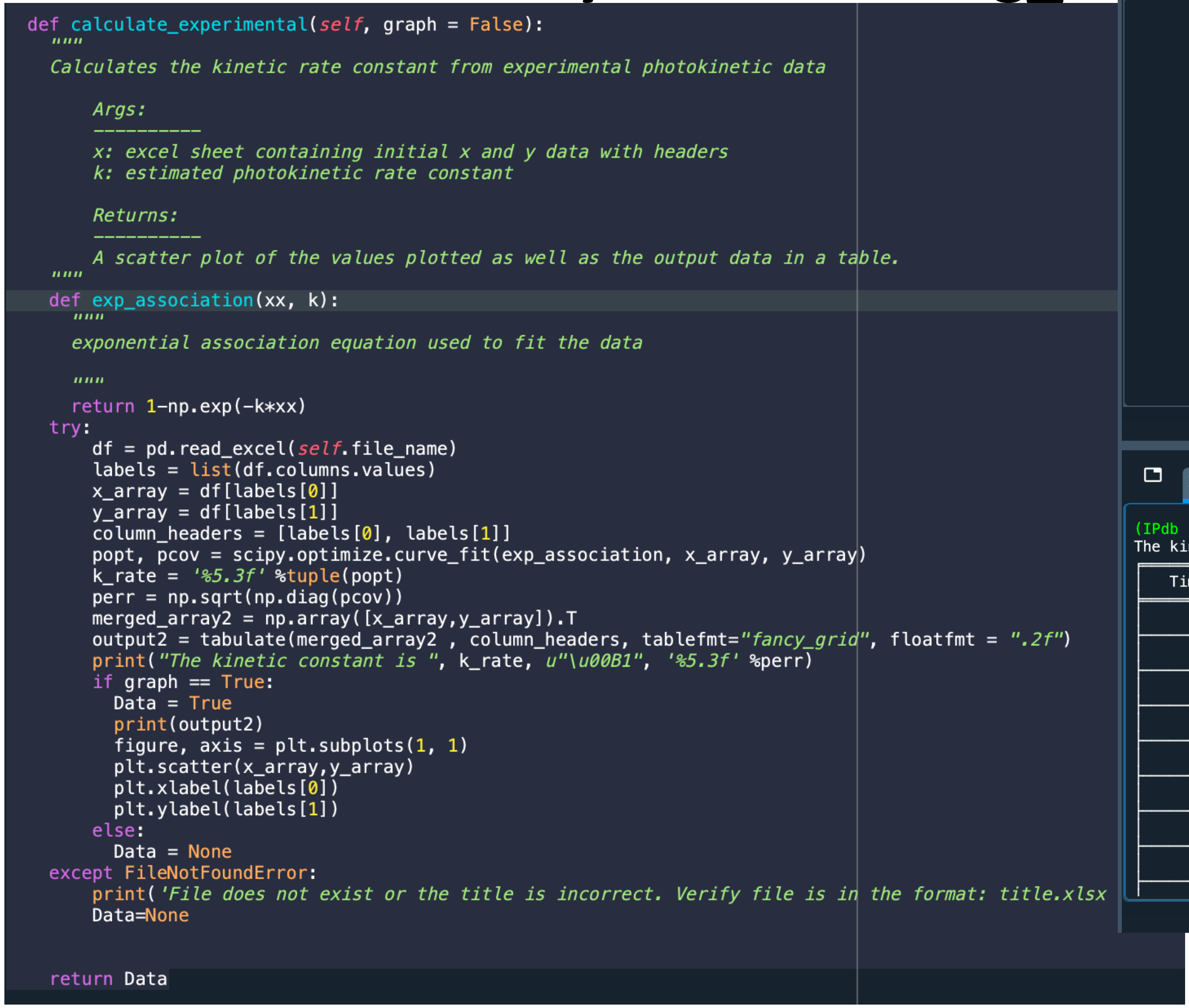
Use Case II: Experimental photokinetic data:

- If the user wants to obtain a graph and kinetic rate constant for their experimental data, the user can input their experimental time or light dosage conditions and corresponding y-values. This is useful as the calculation can be done very quickly and does not require the user to know how to interface with GraphPad, which can be tricky, or if they do not have access to the computer with GraphPad this is an alternative.

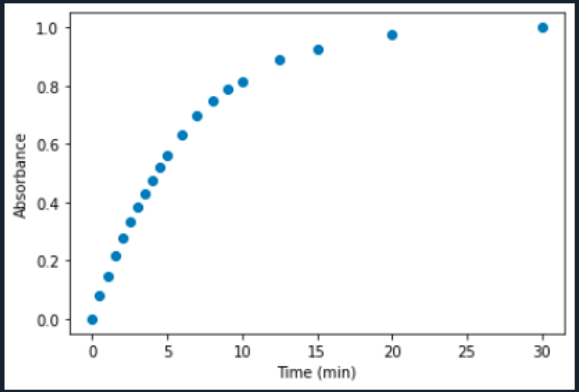
Design:

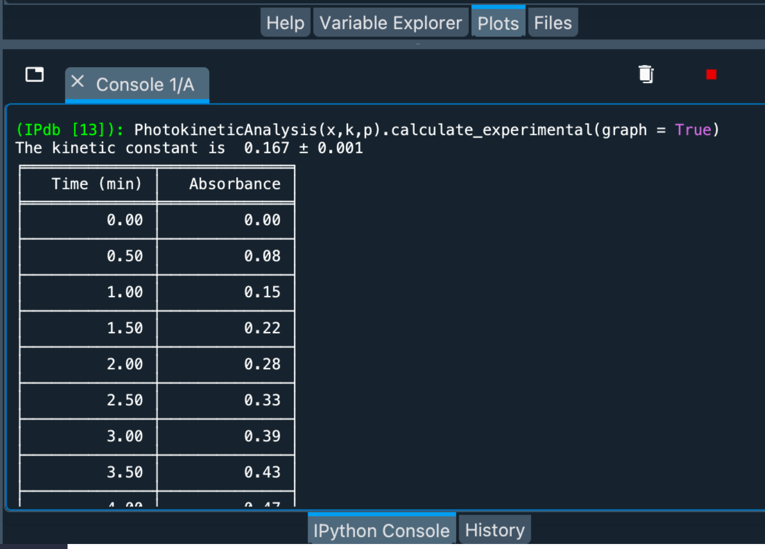
- **PhotokineticAnalysis.calculate_experimental()**
  - a. **Purpose:** Calculates the photouncaging constant k using the first order equation from above using raw data inputted from user as an excel file. It uses scipy.optimize.curve_fit to calculate the k value.
  - b. **Input:**
    - i. Excel file, formatted such that the first column is labelled with the x value and the corresponding y values titles labelled value
  - c. **Output:**
    - i. Table of x and y values
    - ii. Kinetic constant
    - iii. Data normalized and plotted in a scatter plot

PhotokineticAnalysis.calculating_experimental



```
def calculate_experimental(self, graph = False):
  """
  Calculates the kinetic rate constant from experimental photokinetic data

      Args:
      ----------
      x: excel sheet containing initial x and y data with headers
      k: estimated photokinetic rate constant

      Returns:
      ----------
      A scatter plot of the values plotted as well as the output data in a table.
  """
  def exp_association(xx, k):
    """
    exponential association equation used to fit the data

    """
    return 1-np.exp(-k*xx)
  try:
      df = pd.read_excel(self.file_name)
      labels = list(df.columns.values)
      x_array = df[labels[0]]
      y_array = df[labels[1]]
      column_headers = [labels[0], labels[1]]
      popt, pcov = scipy.optimize.curve_fit(exp_association, x_array, y_array)
      k_rate = '%5.3f' %tuple(popt)
      perr = np.sqrt(np.diag(pcov))
      merged_array2 = np.array([x_array,y_array]).T
      output2 = tabulate(merged_array2 , column_headers, tablefmt="fancy_grid", floatfmt = ".2f")
      print("The kinetic constant is ", k_rate, u"\u00B1", '%5.3f' %perr)
      if graph == True:
        Data = True
        print(output2)
        figure, axis = plt.subplots(1, 1)
        plt.scatter(x_array,y_array)
        plt.xlabel(labels[0])
        plt.ylabel(labels[1])
      else:
        Data = None
  except FileNotFoundError:
      print('File does not exist or the title is incorrect. Verify file is in the format: title.xlsx
      Data=None


  return Data
```





```
(IPdb [13]): PhotokineticAnalysis(x,k,p).calculate_experimental(graph = True)
The kinetic constant is  0.167 ± 0.001
┌────────────┬──────────────┐
│ Time (min) │   Absorbance │
├────────────┼──────────────┤
│       0.00 │         0.00 │
├────────────┼──────────────┤
│       0.50 │         0.08 │
├────────────┼──────────────┤
│       1.00 │         0.15 │
├────────────┼──────────────┤
│       1.50 │         0.22 │
├────────────┼──────────────┤
│       2.00 │         0.28 │
├────────────┼──────────────┤
│       2.50 │         0.33 │
├────────────┼──────────────┤
│       3.00 │         0.39 │
├────────────┼──────────────┤
│       3.50 │         0.43 │
├────────────┼──────────────┤
│       4.00 │         0.47 │
```

Use Case III: ChemotaxisSimulation()

Use Case III: Modeling 2D cell migration in response to a stimulus:

- If the user wants to visualize 2D cell migration in response to different conditions such as cell type (cell types have varying characteristics, which can be modeled by changing the cell proliferation, death, movement rate inputs), size of the experiment, number of cells they are starting with and their orientation (e.g. for an experimental scratch assay usually cells are confluent on two sides of 2D area with no cells at the center, and their migration into the center is tracked. Another example are transwell migration assays.

Design:

- **ChemotaxisSimulation.cell_movement()**
  - a. **Purpose:** Simulates cell migration using a stochastic agent based model designed by Fadai et al. 2019.
    - i. **Input: n,m,CC,V,rm,rp,rd,tf,C_start,C_end,rhox**
    - ii. **Output:** Matrix consisting of cell location using a heat map, where yellow equates to occupancy by a cell and purple equates to an empty compartment
- **ChemtaxisSimulation.()**
  - a. **Purpose:** Calculates the photouncaging constant k using the first order equation from above using raw data inputted from user as an excel file. It uses scipy.optimize.curve_fit to calculate the k value.
  - b. **Input:**
    - i. Title of the video generated
  - c. **Output:**
    - i. Movie of cell migration over time

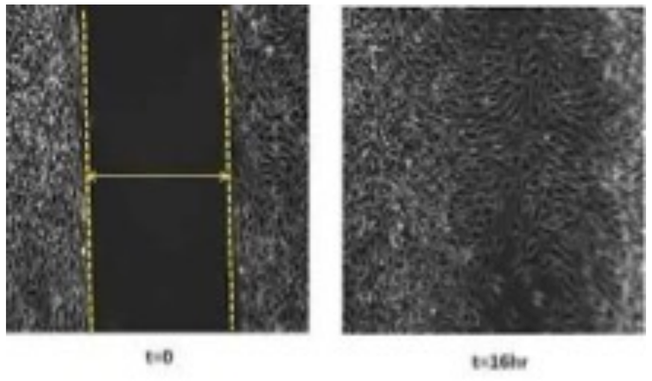
ChemotaxisSimulation.simulate()

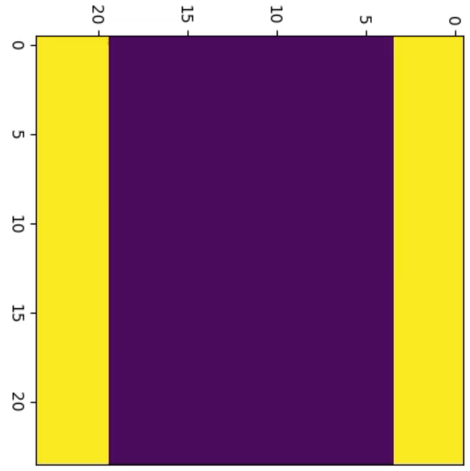
Input:

- •V:compartment size
- •rm: motility rate between 0-1, where 1 equates to faster cells
- •rp: proliferation rate between 0-1, where 1 equates to max proliferation
- •rd: death rate between 0-1, where 1 equates to max death rate
- •tf: final dimensionless time
- •C_start: if there is a chemotactic variable applied, when is it applied.
- •C_end: if there is a chemotactic variable applied, when does it end.
- •rhox: how the chemotactic variable is applied over space (i.e gradient,etc)

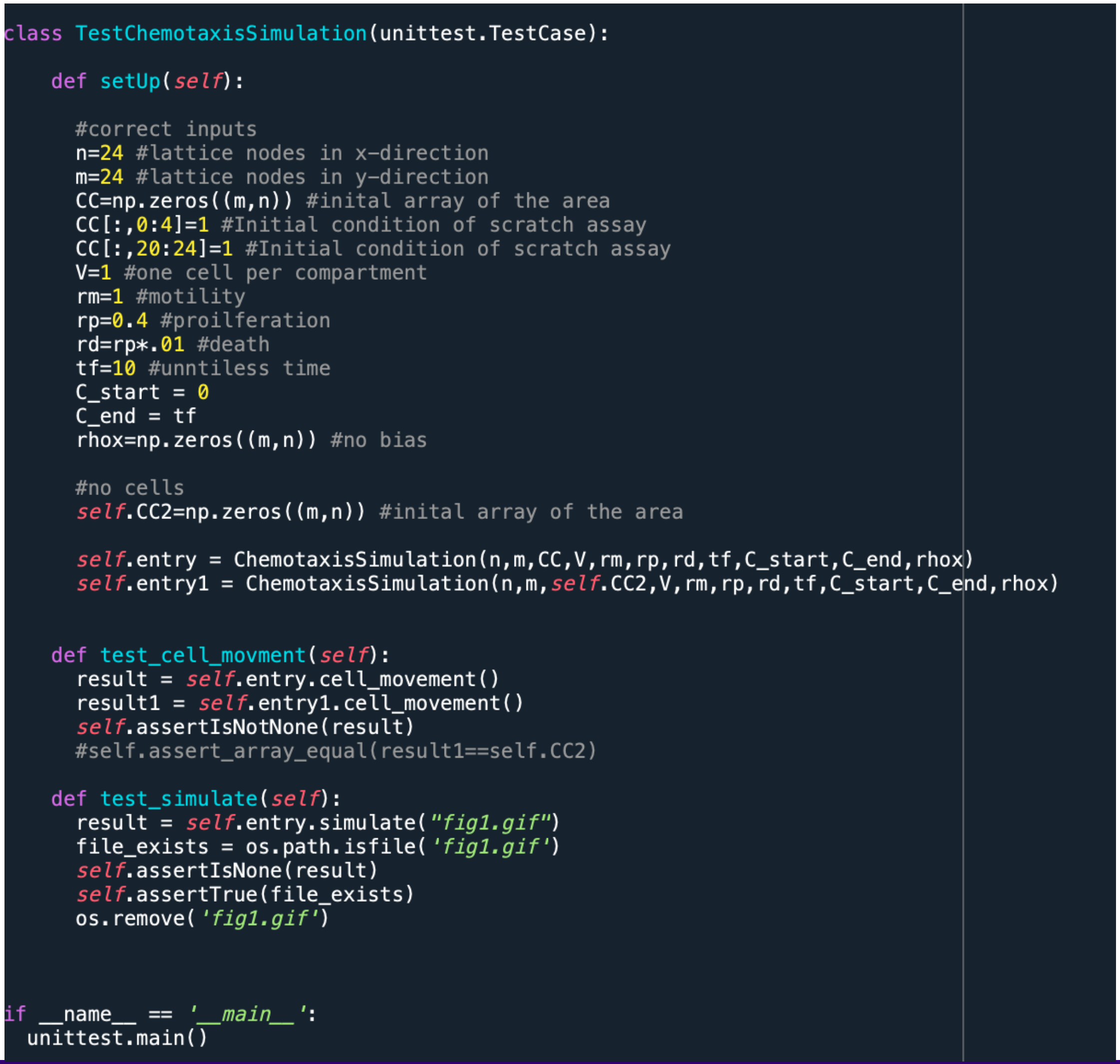
Output:

- •Graph of cell position



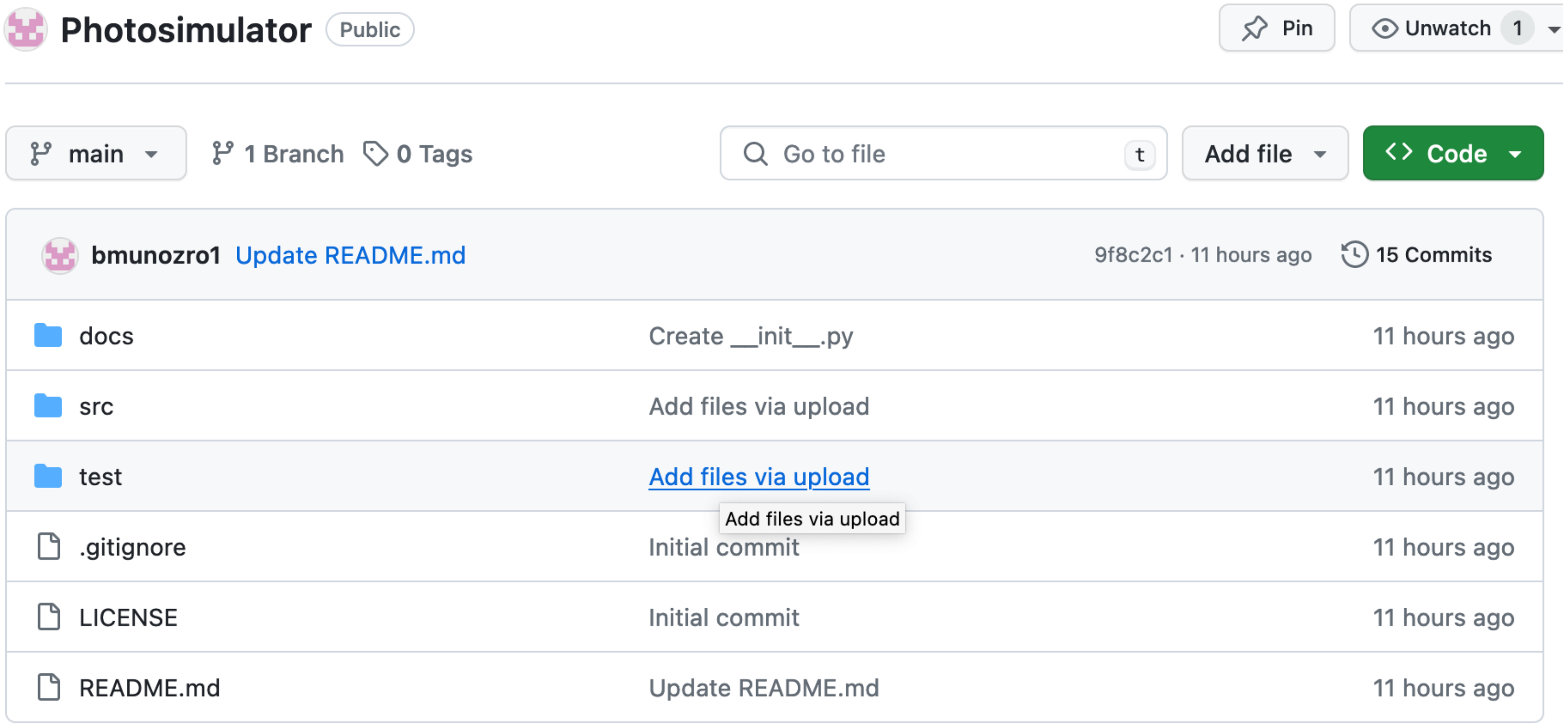


TestChemotaxisSimulation



```
class TestChemotaxisSimulation(unittest.TestCase):

    def setUp(self):

      #correct inputs
      n=24 #lattice nodes in x-direction
      m=24 #lattice nodes in y-direction
      CC=np.zeros((m,n)) #inital array of the area
      CC[:,0:4]=1 #Initial condition of scratch assay
      CC[:,20:24]=1 #Initial condition of scratch assay
      V=1 #one cell per compartment
      rm=1 #motility
      rp=0.4 #proilferation
      rd=rp*.01 #death
      tf=10 #unntiless time
      C_start = 0
      C_end = tf
      rhox=np.zeros((m,n)) #no bias

      #no cells
      self.CC2=np.zeros((m,n)) #inital array of the area

      self.entry = ChemotaxisSimulation(n,m,CC,V,rm,rp,rd,tf,C_start,C_end,rhox)
      self.entry1 = ChemotaxisSimulation(n,m,self.CC2,V,rm,rp,rd,tf,C_start,C_end,rhox)


    def test_cell_movment(self):
      result = self.entry.cell_movement()
      result1 = self.entry1.cell_movement()
      self.assertIsNotNone(result)
      #self.assert_array_equal(result1==self.CC2)

    def test_simulate(self):
      result = self.entry.simulate("fig1.gif")
      file_exists = os.path.isfile('fig1.gif')
      self.assertIsNone(result)
      self.assertTrue(file_exists)
      os.remove('fig1.gif')



if __name__ == '__main__':
  unittest.main()
```

Project Structure


Photosimulator Public

Pin · Unwatch 1

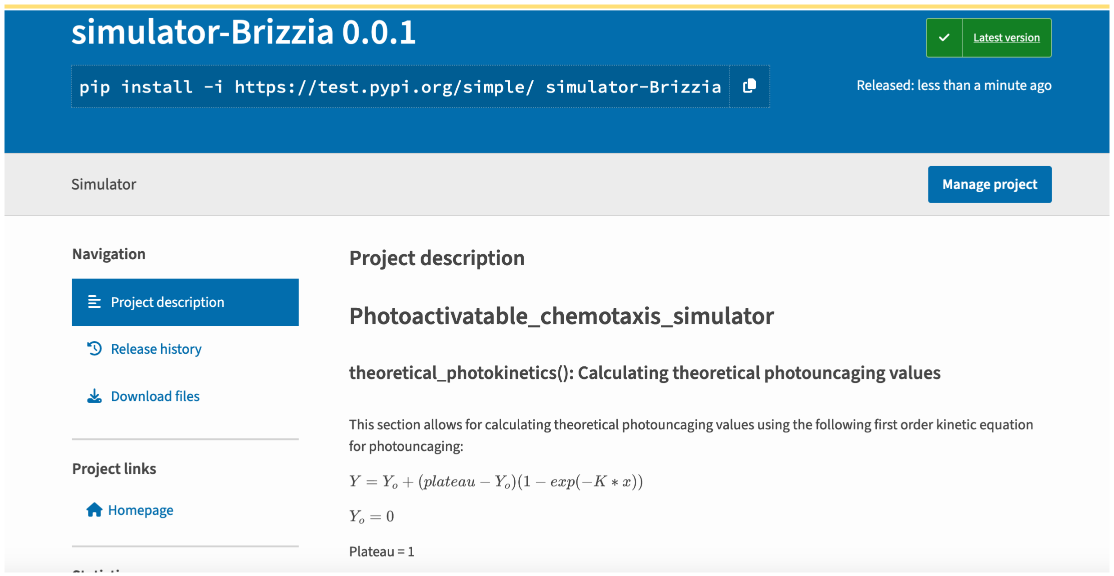
main · 1 Branch · 0 Tags · Go to file t · Add file · Code

bmunozro1 Update README.md · 9f8c2c1 · 11 hours ago · 15 Commits

docs	Create __init__.py	11 hours ago
src	Add files via upload	11 hours ago
test	Add files via upload	11 hours ago
.gitignore	Initial commit	11 hours ago
LICENSE	Initial commit	11 hours ago
README.md	Update README.md	11 hours ago

Lessons Learned/Future Outlook

- I can download github repository but not push changes,
- Make package of final products (used test.pypi to test one of the components and it did work. Unittest does run but need to verify that it runs properly when placed in a different folder)



```
pip install -i https://test.pypi.org/simple/ simulator-Brizzia
```

Simulator

Project description

Photoactivatable_chemotaxis_simulator

theoretical_photokinetics(): Calculating theoretical photouncaging values

This section allows for calculating theoretical photouncaging values using the following first order kinetic equation for photouncaging:

$Y = Y_o + (plateau - Y_o)(1 - exp(-K * x))$

$Y_o = 0$

Plateau = 1