## Experiments

We used batch monoculture [experiments](https://github.com/danielriosgarza/hungerGamesModel/wiki/Experiments) to estimate the [parameters](https://github.com/danielriosgarza/hungerGamesModel/wiki/State-equations#parameters) for the state equations in our model. The codes that execute the complete procedure described below for each of the strains are available [here](https://github.com/danielriosgarza/hungerGamesModel/tree/main/scripts/parameterFIt).

### Moving average

We carried out three sets of monoculture measurements for cell counts, metabolite concentrations, and pH, with each set including at least two replicates. These experiments, conducted under nearly identical conditions, showed consistent kinetics across different days, despite some variability (e.g. [lactate](https://github.com/danielriosgarza/hungerGamesModel/blob/main/files/Figures/riExperiments/lactate.png)). We averaged measurements within regular intervals to establish a consensus kinetic curve for each state, which was used to estimate the model's kinetic parameters. Below, we provide a code example of how we derived the moving average from three independent experiments, where we measured the concentration of live *Roseburia intestinalis* cells:

import os  
import sys  
from pathlib import Path  
  
import numpy as np  
import matplotlib.pyplot as plt  
  
sys.path.append(os.path.join(Path(os.getcwd()).parents[0], 'compare2experiments'))  
from general import \*  
from parseTable import \*  
  
#chose a species  
species = 'ri'  
  
#chose replicates to summarize  
experiments = ['bhri', 'btri', 'bhbtri']  
  
  
#location of the experiment data folder  
strainSummaryFolder = os.path.join(Path(os.getcwd()).parents[1], 'files', 'strainSummaries', species)  
  
  
#chose the state ('live', 'dead', 'glucose', 'acetate', 'pH', etc.)  
measuredState = 'live'  
  
#chose the state type for the plot  
stateType = 'cells'  
  
#chose the regular interval  
intervals = 4  
  
#get the data  
stFile = parseTable(os.path.join(strainSummaryFolder, measuredState + '.tsv'))  
df\_state = getDFdict(stFile, measuredState, False)  
  
labels = ['ri1', 'ri2', 'ri3']  
colors = ['#00ff26', '#003eff', '#ff0000']  
  
s = summarizeExperiments(df\_state, 'ri\_live', experiments, interval = intervals)  
  
makeExperimentPlot(species, measuredState, stateType, stFile, labels, colors)  
plt.plot(s, lw = 5, ls = '--', color = 'k')  
plt.savefig(os.path.join(Path(os.getcwd()).parents[1], 'files', 'Figures', 'movingAverage.png'), dpi = 100)

The black traced line shows the moving average:

![fig:](data:text/html; charset=utf-8;base64,)

### Cubic splines

Moving averages were interpolated by cubic-splines, a method for smoothing data. Cubic spline interpolation is a mathematical technique often employed in data analysis to create a smooth curve that intersects a given set of points (in this case, the moving averages). This technique involves constructing a series of piecewise continuous polynomials of degree three that fit the data.

Cubic splines provide the advantage of calculating state values at any given point in time, not just at the sampled intervals. This means we can evaluate the state at any arbitrary point within the range of the measured time-points, not merely at specified time intervals. Using cubic splines, we can directly compare the smoothed values from the data (the spline's function values) with the solutions of our model's differential equations. This comparison is accomplished by using the simulation times from the model as inputs for the cubic spline functions.

Below, we offer a code example that demonstrates the computation of the cubic spline from the moving average illustrated above:

#get spline  
sp = get\_spline('live', strainSummaryFolder, labels, df\_state = s)  
  
#define an arbitrary time interval  
t = np.linspace(0, 120, 10000)  
  
#get spline points  
live = sp(t)  
  
#add to plot  
plt.plot(t, live, lw = 8, color='#FF6700', alpha = 0.75)  
  
plt.savefig(os.path.join(Path(os.getcwd()).parents[1], 'files', 'Figures', 'spline.png'), dpi = 100)

The orange line is the spline:

![fig:](data:text/html; charset=utf-8;base64,)

### Optimization

Finally, we searched for the parameter set that minimizes the difference between the [states](https://github.com/danielriosgarza/hungerGamesModel/wiki/State-equations#states) in our model simulations and their corresponding smoothed moving averages. To compute the discrepancy between the model simulations and the experimental data, we used the Hubber loss function, which is a function known to be robust against eventual outliers. We then minimized this loss function using Powell's method, an approach that approximates the function using a quadratic model. The method sequentially minimizes this approximation along one direction at a time, generating a set of mutually conjugate search directions to navigate towards the minimum. The parameters that minimize this loss were subsequently used in all downstream simulations of our model.

In summary, the process of fitting our model to sets of experimental data involved the following steps:

1. Acquiring the moving average from experimental data that represent our model states
2. Fitting cubic splines to these averages to enable a smooth estimation of their values within their measured time frames;
3. Adjusting model parameters by minimizing the Huber loss function between model simulations and the smoothed moving averages (simulated by the cubic splines), using Powell's optimization method.