# Project: Bacterial growth

## Part 1: Demonstration in MATLAB

Many dynamical systems are self-limiting because interactions within the system create negative feedback. Logistic growth captures this behavior by tying the growth rate to both the current size and the remaining capacity:

$$\frac{dx}{dt} = \lambda\, x(t)(1 - (x(t)/\theta)^{\alpha}), \quad x(0) = p_0 \tag{1}$$

where

- $x(t)$ is the population at time $t$,
- $\lambda$ is the intrinsic *reproduction rate*,
- $\theta$ is the *carrying capacity*, and
- $\alpha$ is an *impedance* (shape) parameter that controls how quickly the ceiling is "felt" as $x(t)$ grows.

When $\alpha = 1$ the model reduces to the classical logistic equation; other values reshape the trajectory's curvature. Because resources or crowding often curb growth, this structure applies broadly—from bacterial colonies to tumors.

Given $(\lambda, \theta, \alpha, p_0)$, standard ODE solvers yield the trajectory $x(t)$. Given data $\{(t_i, x_i)\}$, we can estimate the parameters that best fit the observations, and then use the fitted model to predict future behavior.

The standard way of doing relies on classical optimization theory. Namely, write the solution of Eq. (1) as $x(t; \lambda, \theta, \alpha, p_0)$. Then, the error between the model and data can be measured using the sum of squared errors:

$$SSE(\lambda, \theta, \alpha, p_0) = \sum_i \left(x(t_i; \lambda, \theta, \alpha, p_0) - x_i(t_i)\right)^2. \tag{2}$$

The optimization problem is to find the combination $(\lambda, \theta, \alpha, p_0)$ that minimizes the SSE. Mathematically, this can be written as:

$$(\lambda^*, \theta^*, \alpha^*, p_0^*) = \arg\min_{(\lambda,\theta,\alpha,\mathrm{p_0})} \mathrm{SSE}(\lambda, \theta, \alpha, \mathrm{p_0}). \tag{3}$$

This is known as an inverse problem– from the data, infer the parameters of the model that yield the best fit between the model and data.

In classical optimization theory, this is done by generating a sequence of approximations $(\lambda_n, \theta_n, \alpha_n, p_{0,n})$ that ideally converge to $(\lambda^*, \theta^*, \alpha^*, p_0^*)$ as $n \to \infty$. The approximations are generated by trying to enforce that the error $SSE(\lambda_n, \theta_n, \alpha_n, p_{0,n})$ decreases as $n$ increases. Note that each $n$, we need to calculate $x(t_i; \lambda_n, \theta_n, \alpha_n, p_{0,n})$, which requires Eq. (1) to be solved (forward problem).

There have been many methods developed to determine the minimizers of such nonlinear functionals. In mathematics, one of the standard ways is to use gradient descent. This can be described as follows. Let $\boldsymbol{\rho} = (\lambda, \theta, \alpha, p_0)$ denote the vector of parameters. Then, the gradient descent iteration is

$$\boldsymbol{\rho}_{n+1} = \boldsymbol{\rho}_n - \delta \boldsymbol{\nabla}_{\boldsymbol{\rho}} SSE(\rho_n), \tag{4}$$

where $\delta$ denotes a rate of descent. Here, $\boldsymbol{\nabla}_{\boldsymbol{\rho}}$ denotes the gradient with respect to the parameters. The challenge is that to calculate $\boldsymbol{\nabla}_{\boldsymbol{\rho}} SSE(\boldsymbol{\rho})$, one needs to compute $\boldsymbol{\nabla}_{\boldsymbol{\rho}} x(t; \boldsymbol{\rho})$, that is, the derivative of the solution to Eq. (1) with respect to the parameters $\boldsymbol{\rho}$. This can be done by introducing by differentiating Eq. (1) with respect to the parameters and solving auxilliary equations, known as the adjoint equations.

An alternative, favored by computer scientists, does not require derivatives and instead uses the simplex method (Nelder-Mead). This approach relies on a geometric approach using a sequence of generalized triangles (simplicies) that trap the minimum values within their ever decreasing interiors. This is implemented in Matlab through the *fminsearch* function.

We will demonstrate how to fit the logistic growth model to synthetic noisy data. The codes can be found in `https://github.com/CalebHend/MCSB-Bootcamp-Data-Fitting/tree/main/fminsearch-MATLAB-Demo`. Note that this approach still requires the solution of Eq. (1) to be computed at each stage of the iteration.

## Part 2: Physics-Informed Neural Networks (PINNs)

In this section you will use Physics-Informed Neural Networks (PINNs) to estimate parameters in a biological growth model. PINNs combine data fitting with the governing differential equation by penalizing both the data misfit and the equation residual. As we will demonstrate, this approach has the advantage that the forward and inverse problem can be solved in a single step. Namely, unlike the conventional approach, one does not have to solve the forward problem multiple times to solve the inverse problem.

The necessary files can be found at:

`https://drive.google.com/drive/folders/10YSAciWFgAvxpZKgGNzzt9tpcDoNeF4W?usp=sharing`

Have one member of your group **make a copy** of the folder in their Drive and **share and edit the copy** with the group. Work only in the copied folder.

**Problems:**

(1) Click the the hyperlink to bring you to the drive where you will see a few Google Colab notebooks. In this drive open *Simple NN* where our first exploration will begin. Follow the instructions provided in the notebook, and read it over carefully. Answer the following questions when you get familiar with this code:

   (a) How well does this NN perform? What do you notice about the plots? Namely, how does the behavior inside the fitting region compare with extrapolating beyond the fitting region?

   (b) Can we improve its fitting capabilities by adjusting network depth and width?

(2) For this next part you will need to consider *Simple NN regularization* and *linear estimate parameters* files. In this approach, we will add an additional constraint– namely that the 2nd derivative should be small using the PINN approach. We will start with the simple NN regularization and will follow this up by considering the linear estimate parameters notebook. Again follow the instructions in the notebook and play around with the code.

   Also answer the following questions when you get familiar with this code:

   (a) How well does this PINN perform? What do you notice about the plots in contrast to (1)?

   (b) Can we improve its fitting capabilities by adjusting network depth and width?

   (c) What does $\eta_{\text{phys}}$ do to the fit when increased and decreased?

   (d) Do the results depend on the size of the noise? (e.g., vary the noise parameter).

   (e) Do the results depend on the slope and intercept of the ground truth solution?

   (f) Repeat these questions for the *linear estimate parameters* note book where you will quantify the accuracy of the parameters.

(3) Now you should be familiar enough with the structure of a PINN, such that you can now explore the *simple NN regularization Logistic Growth.* Where now we will pretend we know the parameters.

   (a) How well does this PINN perform? What do you notice about the plots?

   (b) Can we improve its fitting capabilities by adjusting network depth and width?

   (c) What does $\eta_{\text{phys}}$ do to the fit when increased and decreased?

   (d) Do the results depend on the size of the noise?

(e) Do the results depend on the choices of parameters for the ground truth solution?

(4) Now consider that we only guessed that the model we are using is the logistic growth but we have no idea what the parameters are. We now will use the PINN to extract the biologically relevant parameters. You now must open the *inverse logistic growth pinn*, where you will now answer the following questions after reading over the code:

(a) How well does this PINN perform? What do you notice about the plots in contrast to (3)?

(b) Can we improve its fitting capabilities by adjusting network depth and width?

(c) Try fixing and unfixing parameters, what happens? What can you do to justify fixing a parameter?

(d) What does $\eta_{\text{phys}}$ do to the fit when increased and decreased?

(e) Do the results depend on the level of the noise or the ground truth parameters?

(5) Now we shall use your data gathered in the lab portion to extract the parameters for your experiments using *inverse logistic growth pinn*. Modify the code to do so and answer the following questions:

(a) How well does the PINN fit your data? What observations can you make?

(b) How does tuning the PINN, affect the performance?

(c) How could you improve this code and or method to get better results.

## Bonus: Thinking beyond outside the $> \epsilon$ ...

Can anyone tell me what models would work better for your experimental data? And how would you validate this claim?