Experimental design for predictive models in microbiology depending on environmental variables

Polina Gaindrik, Jonas Pleyer, Daniel Heger and Christian Fleck

Abstract The aim of predictive microbiology is the provision of tools and methods for predicting the growth, survival, and death of microorganisms in different food matrices under a range of environmental conditions. The parametrized mathematical models need to be calibrated using dedicated experimental data. In order to efficiently plan experiments, model-based experimental design is used. In this chapter, we explain model-based experimental design and provide step-by-step instructions for finding the optimal design using the well-known Baranyi-Roberts growth model as an example. We provide the Python software *eDPM* for Ordinary Differential Equation (ODE) -based models, such that the reader can apply model-based experimental design in his/her research context.

Key words: Optimal experimental design, Parameter estimation, Fisher information matrix (FIM), Identifiability, Uncertainty

Polina Gaindrik
University of Freiburg, Freiburg Center for Data Analysis and Modeling, Ernst-Zermelo-Str. 1, 79104 Freiburg, Germany,

Jonas Pleyer
University of Freiburg, Freiburg Center for Data Analysis and Modeling, Ernst-Zermelo-Str. 1, 79104 Freiburg, Germany,

Daniel Heger
tsenso GmbH, Johannesstr. 19, 70176 Stuttgart, Germany

Christian Fleck
University of Freiburg, Freiburg Center for Data Analysis and Modeling, Ernst-Zermelo-Str. 1, 79104 Freiburg, Germany, e-mail: `christian.fleck@fdm.uni-freiburg.de`

1 Introduction

Mathematical modeling is widely used to describe, understand and predict the behavior of living systems. In particular, in the field of predictive microbiology, one can find a large variety of works that dwell on building models of different levels of complexity controlled by model parameters, e.g., to describe bacterial growth [1]. Predictive microbiology is a subfield of food microbiology based on the premise that the behavior of microorganisms can be described by the use of mathematical models accounting the effects of various factors such as temperature, pH, water activity, and the presence of preservatives, on the growth and survival of microorganisms. These models can then be used to predict the behavior of microorganisms under different conditions and to evaluate the effectiveness of various control measures, such as refrigeration, heat treatment, or the addition of preservatives. Predictive microbiology has many applications in food safety, as it can be used to assess the risk of foodborne illness, to design safe food processing and storage practices, and to develop new food products with extended shelf-life. For successful predictions it is essential that the model parameters can be estimated from experimental data. However, due to measurement noise the estimates are accompanied by some uncertainties. That gives rise to a set of important questions: Can all parameters of the given model structure be estimated? What should be measured and when? What are the confidence intervals for the parameters? How can the experimental effort be minimized? Answering these questions equates to finding the optimal experimental design (OED). This results in defining optimized experimental conditions and/or measurement times which allows for a reduction of the experimental load without loss of information [2, 3]. In general, experimental design can be also used for model discrimination [4, 5]. However, in this chapter we focus on the design procedure developed to increase parameter estimation precision. Comprehensive reviews on can be found here: [6–9].

The whole parameter estimation process starts with choosing a model structure and defining observable quantities which the researcher is interested in (see Fig. 1). Once this is chosen, a first parameter set is selected based on literature review or prior data. In order to fine-tune the experimental design setup to the given use-case, we need to choose an objective function that will be optimized. Depending on the goal, a researcher faces an important choice; which of the several sometimes contradicting objectives should be chosen for a particular case? For example, is it more desirable to have precise knowledge in a chosen set of parameters and less information about the remaining ones? What is the best balance between experimental effort and precision? Using multi-objective approaches, some attempts were made to answer these questions and to improve the experimental design by combining several objectives [10, 11]. For models which depend on external cues like temperature, it is *a priori* not clear whether constant variable conditions are more efficient for the parameter estimation [12, 13]. Using the initial guess of parameters a first OED is determined taking into account specific constraints, such as maximum number of measurements, measurement times, etc. Next, the designed experiments need to be performed and this data results into a new estimation of the parameters. If the confidence intervals of the parameters are sufficiently small, the scheme ends, otherwise one uses the new estimates as the starting point for the next experimental design. The process can be repeated several times to increase the precision of the parameter estimates until the desired accuracy is achieved. To test the scheme one can also perform numerical experiments and obtain *in-silico* data.

2 Materials and Methods

There are several software packages available for model-based experimental design [14–16]. These packages comprise numerous tools and consequently require some time to understand how to use them. We developed *eDPM* (Experimental Design for Predictive Microbiology), which focuses on ease of use. It is a set of tools to calculate the Sensitivity and Fischer Information Matrix and do parameter space exploration in order to find optimal results. We require a working installation of the popular scripting language Python $\geq$ 3.7 [17]. For installation instructions, please refer to the website python.org. In addition, we expect users to be able to write, execute and display output of scripts. This tutorial can also be followed using Jupyter Notebook [18]. Users can obtain it by installing *eDPM* from pypi.org. The python website has guides for installing packages packaging.python.org/. Most Unix-Based systems (e.g., GNU/Linux) and Mac-OS can use `pip`

```
pip install eDPM
```

or `conda` to install the desired package.

```
conda install eDPM
```

The documentation of the package is continuously updated. After the installation of the package is complete, we open a file with a text editor of choice and simply start writing code. We begin by writing a so-called she-bang which is responsible to signalize that this file is to be executed with python. Afterwards, we import the needed packages. This will serve as the starting point for our script.

```python
#!/usr/bin/env python3

import numpy as np
from eDPM import *
```

Code Sample 1. Import statements to use *eDPM*

In the following, we will append more code to it and utilize the methods developed in *eDPM* [19]. The Code Samples containing line numbers correspond to one script that defines the system subjected to optimization. The line numbers indicate the order of the code. If line numbers are missing, these should be filled by blank lines for better readability.

2.1 Model Formulation

2.1.1 Theory

As a starting point it is necessary to define the mathematical model. We restrict our discussion to biological system which can be described by a system of Ordinary Differential Equations:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}, \mathbf{u}, \mathbf{p}) \\ \mathbf{x}(t_0) = \mathbf{x}_0. \end{cases} \tag{1}$$

Here $\mathbf{x} = (x_1, x_2, ..., x_n)$ is a vector of state variables of the system with initial condition $\mathbf{x}_0$, t is a time, $\mathbf{u}$ is a vector of an externally controlled inputs into the system (e.g., supply of glucose or the external temperature) and $\mathbf{p}$ are the parameters of the system. We assume that a subset of the parameters $\mathbf{p}$ is unknown and should be estimated from data. Predominantly, the state variables cannot be directly observed, but rather the observables are functions of the state variables:

$$\mathbf{y}(t) = \mathbf{g}(t, \mathbf{x}(t), \mathbf{u}, \mathbf{p}) + \epsilon(t), \tag{2}$$

where the function $\mathbf{g}$ is an observation or output function, and ϵ is a measurement noise. The observational noise is often assumed to be an independently distributed Gaussian noise with zero mean and variance σ^2: $\epsilon(t) \sim N(0, \sigma^2)$, $\forall t$.

In this tutorial, we demonstrate the usage of experimental design on the widely employed mathematical model developed by Baranyi and Roberts [20], which was devised to describe bacterial growth. The model introduces two state variables $\mathbf{x} = (x_1, x_2)$, where $x_1(t)$ denotes the cell concentration of a bacterial population at the time t and $x_2(t)$ is the quantity defining the proportion of the growth rate specified by the environment, e.g., a limiting nutrient critical for bacterial growth.

$$\begin{cases} \dot{x}_1(t) = f_1(x_1, x_2) = \frac{x_2(t)}{x_2(t)+1}\mu^{\max}\left(1 - \frac{x_1(t)}{x_1^{\max}}\right)x_1(t) \\ \dot{x}_2(t) = f_2(x_1, x_2) = \mu^{\max}x_2(t) \end{cases} . \tag{3}$$

Here $\mu^{\max}$ determines the maximum growth rate, and $x_1^{\max}$ is the maximal bacteria concentration due to environmental constraints. The condition $x_2(0)$ allows to quantify the initial physiological state of the cells and, hence, the process of adjustment (lag-phase) [21]. To account for the influence of the temperature on the activity of the model, we will use the 'square root' or Ratkowsky-type model for the maximum growth rate [22]

$$\sqrt{\mu^{\max}} = b(T - T_{\min}), \tag{4}$$

where b is the regression coefficient, and $T_{\min}$ is the minimum temperature at which the growth can occur. As the observable we choose the bacteria count, i.e.:

$$y(t) = x_1(t) + \epsilon(t), \tag{5}$$

a common choice in predictive microbiology.
The equations (3 - 5) fully define the system. Here $x_1^{\max}, b, T_{\min}$ are the parameters that we estimate using observational data y at measurement times t_i, and temperature T is an input of the system. Based on this model, we would like to optimize the choice of measurement times as well as temperatures (inputs) of the system to find the optimal experimental design.

2.1.2 Code

In order to be able to solve the equations numerically, we first need to define the ODE system described by Eq. (1) in python. Next, we determine all numerical values present in the system. We distinguish between time points t_i at which the result of the ODE is evaluated, inputs $\mathbf{u}$, which alter the behavior of our system (for example temperature, humidity, etc.), parameters $\mathbf{p}$, which are the quantities that we want to estimate and other arguments which might be needed to solve the ODE. Table 1 provides an overview of these quantities and gives corresponding names in the code. In the following, we will step-by-step explain how to specify all variables needed by using the example of the Baranyi-Roberts model.
First we define the right-hand side of the ODEs, i.e., we need to implement the Baranyi-Roberts model (3 - 4) into a function as can be seen in Code Sample 2.

```python
# The function name can be chosen freely, but the order
# of required function arguments is fixed by the definition.
def baranyi_roberts_ode(t, x, u, p, ode_args):
    # Unpack the input vectors x,u,p for easy access
    # of their components
    (x1, x2) = x
    (Temp, ) = u
    (x_max, b, Temp_min) = p

    # Calculate the maximum growth rate
    mu_max = b**2 * (Temp - Temp_min)**2

    # Calculate the right hand side of the \ac{ode}, store
    # it in a list [ ... ] and return it.
    return [
        mu_max * (x2/(x2 + 1)) * (1 - x1/x_max) * x1,
        mu_max * x2
    ]
```

Code Sample 2. Definition of the Baranyi-Roberts ODE model.

2.2 Parameter Estimation

After defining the model, the user needs to provide an initial parameter set. This could be chosen from the literature or estimated from previously gathered experimental data. It is common to assume that the observational or measurement noise is Gaussian white noise (e.g., no temporal correlations) with zero mean and variance $\sigma^2(t)$: $\epsilon(t) \sim N(0, \sigma^2(t))$, $\forall t$ [23]. In this case the logarithm of the likelihood function (log-likelihood) is given by Eq. (6):

$$\ln L(\mathbf{p}) \propto -\sum_i \frac{\left(\mathbf{g}^i(\mathbf{p}) - \mathbf{d}^i\right)^2}{2\sigma_i^2}. \tag{6}$$

Here we introduced the following shorthand notations:

- $\mathbf{d}^i$: data measured at time $t = t_i$
- $\mathbf{g}^i(\mathbf{p})$: the observation function depending on the parameter vector $\mathbf{p}$ evaluated at time $t = t_i$
- σ_i^2: variance of the observational noise at time $t = t_i$.

The method of maximum likelihood consists in searching for the parameter vector $\mathbf{p}$ which maximizes the (log-)likelihood [24].

2.3 Experimental Design

Following our definition of the model and setting the initial parameter vector $\mathbf{p}_0$ we can proceed with experimental design. In essence, experimental design comprises maximizing an objective function depending on the model, the initial parameter vector $\mathbf{p}_0$, external input $\mathbf{u}$ into the system, and a set of discrete observation times t_i at which the potential measurements should be performed. The objective function needs to quantify the information the observation $\mathbf{y}$ has about the parameter vector $\mathbf{p}$. A common choice here is constructing objective functions based on the Fisher information matrix (FIM) [25]. According to the so-called 'Cramer-Rao inequality', the Fisher information is inversely proportional to the minimal squared estimation error [26]. This relation justifies using the FIM to improve our experimental design. In the following we explain one of the ways to calculate the FIM for an ODEs system (Eq. 1) and the observable function (Eq. 2) [25].

2.3.1 Sensitivity Calculation

An easy way to calculate the FIM uses local sensitivities of the observables y, that are in turn calculated from local sensitivities of the internal variables x [12, 27]. Assume that functions $\mathbf{f}$ and $\mathbf{g}$ are differentiable functions with respect to the state variables $\mathbf{x}$ and parameters $\mathbf{p}$. The local sensitivities of i-component of the vector $\mathbf{x}$ w.r.t. r

j-component of the parameter vector are defined by $s^x_{ij} = (\mathrm{d}x_i/\mathrm{d}p_j)$. These can be calculated using an enhanced system of the ODEs:

$$\begin{cases} \dot{x}_i(t) = f_i(t, \mathbf{x}, \mathbf{u}, \mathbf{p}) \\ \dot{s}^x_{ij} = \sum_k \frac{\partial f_i}{\partial x_k} s^x_{kj} + \frac{\partial f_i}{\partial p_j} \end{cases} . \tag{7}$$

For the FIM we need the sensitivities of observable functions $(\mathrm{d}y_i/\mathrm{d}p_j) = s_{ij}$. We determine these at the certain times t_m and input u_n using the solutions of s^x_{ij}:

$$s_{ij}(t_m, u_n) = \sum_k \left.\frac{\partial g_i}{\partial x_k}\right|_{t_m,u_n} s^x_{kj}(t_m, u_n) + \left.\frac{\partial g_i}{\partial p_j}\right|_{t_m,u_n} . \tag{8}$$

In the case the parameters are of very different scale (e.g., on the second and on the day scale), it may be advisable to use the relative or normalized sensitivities to improve not the absolute but the relative accuracy of the parameter estimates.:

$$\tilde{s}_{ij}(t_m, u_n) = \frac{\mathrm{d}\ln(y_i)}{\mathrm{d}\ln(p_j)} = \left.\frac{\mathrm{d}y_i}{\mathrm{d}p_j}\frac{p_j}{y_i}\right|_{t_m,u_n} . \tag{9}$$

These sensitivities are the elements of the sensitivity matrix [28]. For example, in case of two observables $\mathbf{y} = (y_1, y_2)$, two different inputs $\mathbf{u} = (u_1, u_2)$, N different measurement times and N_p parameters, sensitivity matrix reads:

$$S = \begin{pmatrix} s_{11}(t_1,u_1) & \ldots & s_{1N_p}(t_1,u_1) \\ \vdots & & \vdots \\ s_{11}(t_N,u_1) & \ldots & s_{1N_p}(t_N,u_1) \\ s_{11}(t_1,u_2) & \ldots & s_{1N_p}(t_1,u_2) \\ \vdots & & \vdots \\ s_{11}(t_N,u_2) & \ldots & s_{1N_p}(t_N,u_2) \\ s_{21}(t_1,u_1) & \ldots & s_{2N_p}(t_1,u_1) \\ \vdots & & \vdots \\ s_{21}(t_N,u_1) & \ldots & s_{2N_p}(t_N,u_1) \\ s_{21}(t_1,u_2) & \ldots & s_{2N_p}(t_1,u_2) \\ \vdots & & \vdots \\ s_{21}(t_N,u_2) & \ldots & s_{2N_p}(t_N,u_2) \end{pmatrix} \tag{10}$$

This matrix will be used to directly calculate the FIM:

$$F = S^T Q^{-1} S. \tag{11}$$

Here, Q is the covariance matrix of measurement error [29, 30]. If the measurements are independent, then only the diagonal elements of the matrix are non-zero:

$$Q = \begin{pmatrix} \sigma_1^2(t_1,u_1) & 0 & 0 & \dots & 0 \\ 0 & \sigma_1^2(t_2,u_1) & 0 & \dots & 0 \\ 0 & 0 & \sigma_1^2(t_1,u_2) & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_2^2(t_N,u_2) \end{pmatrix}, \tag{12}$$

where $\sigma_i^2(t_m,u_n)$ is the error of the measurements of observable y_i at time point t_m with input u_n. The error can be either estimated from data or approximated by some error model. For example, one may consider that the error contribution consists of an absolute part that stays constant and a relative part that is proportional to the observable values. Then the diagonal elements of the covariance matrix take form:

$$\sigma_i^2(t_m,u_n) = \gamma_{\text{abs}} + \gamma_{\text{rel}} \cdot y_i(t_m,u_n), \tag{13}$$

where γ_{abs} and γ_{rel} are the coefficients determining the absolute and the relative error contribution, respectively. In case one uses relative sensitivities $\tilde{s}_{ij}(t_m,u_n)$, one needs to use relative measurements errors for the matrix Q:

$$\tilde{\sigma}_i^2(t_m,u_n) = \frac{\sigma_i^2(t_m,u_n)}{y_i^2(t_m,u_n)}. \tag{14}$$

2.3.2 Numerical Sensitivity Calculation

For the calculation of the sensitivities (Eq. 7) we need to supply the derivatives of f with respect to the parameters and state variables. We define $\mathbf{x} = (x_1,x_2,x_3)$ and $\mathbf{p} = (p_1,p_2,p_3)$ with $p_1 = x_{max}$, $p_2 = b$ and $p_3 = T_{\text{min}}$. Firstly, we calculate the mathematical derivatives $\partial f_i/\partial p_j$ and $\partial f_i/\partial x_j$ and then implement the corresponding functions. The first component of the Baranyi-Roberts model reads:

$$\dot{x}_1(t) = f_1(t,\mathbf{x},\mathbf{u},\mathbf{p}) = \mu^{\text{max}} \frac{x_2(t)}{x_2(t)+1} \left(1 - \frac{x_1(t)}{x_1^{\text{max}}}\right) x_1(t), \tag{15}$$

where $\sqrt{\mu^{\text{max}}} = b(T - T_{\text{min}})$. It follows:

$$\frac{\partial \mu^{\text{max}}}{\partial p_1} = 0 \tag{16}$$

$$\frac{\partial \mu^{\text{max}}}{\partial p_2} = 2b(T - T_{\text{min}})^2 = \frac{2\mu^{\text{max}}}{b} \tag{17}$$

$$\frac{\partial \mu^{\text{max}}}{\partial p_3} = -2b^2(T - T_{\text{min}}) = -\frac{2\mu^{\text{max}}}{T - T_{\text{min}}} \tag{18}$$

and consequently:

$$\frac{\partial f_1}{\partial p_1}(t) = \mu^{\max} \quad \frac{x_2(t)}{x_2(t)+1} \quad \frac{x_1(t)}{\left(x_1^{\max}\right)^2}\, x_1(t) \tag{19}$$

$$\frac{\partial f_1}{\partial p_2}(t) = \frac{2\mu^{\max}}{b} \quad \frac{x_2(t)}{x_2(t)+1}\left(1 - \frac{x_1(t)}{x_1^{\max}}\right) x_1(t) \tag{20}$$

$$\frac{\partial f_1}{\partial p_3}(t) = -\frac{2\mu^{\max}}{T - T_{\min}}\frac{x_2(t)}{x_2(t)+1}\left(1 - \frac{x_1(t)}{x_1^{\max}}\right)x_1(t). \tag{21}$$

Similarly, the derivatives $\partial f/\partial x_i$ are given by:

$$\frac{\partial f_1}{\partial x_1}(t) = \mu^{\max} \quad \frac{x_2(t)}{x_2(t)+1}\left(1 - 2\frac{x_1(t)}{x_1^{\max}}\right) \tag{22}$$

$$\frac{\partial f_1}{\partial x_2}(t) = \mu^{\max}\frac{1}{(x_2(t)+1)^2} \quad \left(1 - \frac{x_1(t)}{x_1^{\max}}\right)x_1(t). \tag{23}$$

The readers are encouraged to calculate the components $\partial f_2/\partial p_i$ and $\partial f_2/\partial x_i$ as an exercise. The resulting implemented functions in python can be seen in Code Sample 3.

```
def ode_dfdp(t, x, u, p, ode_args):
    (x1, x2) = x
    (Temp, ) = u
    (x_max, b, Temp_min) = p
    mu_max = b**2 * (Temp - Temp_min)**2
    return [
        [
            mu_max * (x2/(x2 + 1)) * (x1/x_max)**2,
            2 * b * (Temp - Temp_min)**2 * (x2/(x2 + 1))
                * (1 - x1/x_max)*x1,
            -2 * b**2 * (Temp - Temp_min) * (x2/(x2 + 1))
                * (1 - x1/x_max)*x1
        ],
        [
            0,
            2 * b * (Temp - Temp_min)**2 * x2,
            -2 * b**2 * (Temp - Temp_min) * x2
        ]
    ]

def ode_dfdx(t, x, u, p, ode_args):
    (x1, x2) = x
    (Temp, ) = u
    (x_max, b, Temp_min) = p
    mu_max = b**2 * (Temp - Temp_min)**2
    return [
        [
            mu_max * (x2/(x2 + 1)) * (1 - 2*x1/x_max),
            mu_max * 1/(x2 + 1)**2 * (1 - x1/x_max)*x1
        ],
        [
            0,
            mu_max
        ]
    ]
```

Code Sample 3. Derivatives of the function f of the Baranyi-Roberts model ODE.

2.3.3 Define numerical values

After the definition of the structure of the ODEs, we need to specify numerical values. It is good practice, to gather such definitions in the `__main__` method of the python program as it was done in Code Sample 4. We start by defining the parameters of the ODEs (line 64) and afterwards the initial values (line 67). Next, we constrain the optimization routine to find the best possible time points in the interval $t_i \in \left[t_{\text{low}}, t_{\text{high}}\right]$. To do this, we write the times as a dictionary with entries `{"lb":t_low, "ub":t_high, "n":n_times}` where `"lb"` and `"ub"` are the lower and the upper bound while `n` describes the number of discrete time points at which we want to sample the system (line 70). In contrast, if we wanted to specify explicit values for the sampling points, we would have needed to supply a list of time values or a `np.ndarray` directly. One can see this approach in lines 78-80 of the code example. Here, we supply a `np.ndarray` with explicit values, thus fixing the sampling points them for the optimization routine.

```
62  if __name__ == "__main__":
63      # Define parameters
64      p = (2e4, 0.02, -5.5)
65
66      # Define initial conditions
67      x0 = np.array([50, 1])
68
69      # Define interval and number of sampling points for times
70      times = {"lb":0.0, "ub":100.0, "n":2}
71
72      # Define explicit temperature points
73      Temp_low = 2.0
74      Temp_high = 12.0
75      n_Temp = 2
76
77      # Summarize all input definitions in list (only temperatures)
78      inputs = [
79          np.linspace(Temp_low, Temp_high, n_Temp)
80      ]
```

Code Sample 4. The main function will encompass every step in our experimental design approach. In the beginning, we insert the actual values for our model definition.

2.3.4 Defining Explicit Values and Sampling

There are two distinct ways to define inputs of the model. First, one can fix (multiple) explicit values for an input variable

```
inputs = [
    np.linspace(Temp_low, Temp_high, n_Temp)
]
# >>> inputs
# [array([2., 12.])]
```

so that experiment will then be solved for only these values without optimizing them. Secondly, one can also optimize inputs of the system. For example, suppose we want to optimize the temperature at which the experiments are performed, which means we let the optimization algorithm pick the optimal temperature points such that the information gathered from the system is maximized. To sample in the interval `(Temp_low, Temp_high)` with `n_Temp` points the reader can simply write:

```
    inputs = [
        {"lb":Temp_low, "ub":Temp_high, "n":n_Temp}
    ]
    # >>> inputs
    # [{'lb': 2.0, 'ub': 12.0, 'n': 2}]
```

The difference between choosing explicit values and specifying a sampling range may be subtle at first glance, but in turn allows to very easily switch between fixed values and optimized sampling points.

The different variables can be treated individually as needed. Suppose, we have a system with temperatures and humidity as input variables. We could decide to have the temperature optimized, but fix the humidity explicitly, because experimentally one can change the temperature continuously (or in discrete steps) but is restricted regarding the humidity. In our code, we simply would mix explicit and sampled definitions for individual inputs:

```
    inputs = [
        # These values will be sampled
        {"lb":Temp_low, "ub":Temp_high, "n":n_Temp},
        # These are fixed
        np.array([0.4, 0.45, 0.5, 0.55])
    ]
```

2.3.5 Define Model

After we have decided on the numerical values for our model, we need to put everything together. The `FisherModel` class serves as the entry point. Here, we simply supply every previously made definition of variables and methods. When using the syntax as shown in code sample 5, the order of arguments does not matter. However, when only using `FisherModel(x0, 0.0, ...)` , please pay attention to the order of arguments.

```
# Create the FisherModel which serves as the entry point
#  for the solving and optimization algorithms
fsm = FisherModel(
    ode_x0=x0,
    ode_t0=0.0,
    ode_fun=baranyi_roberts_ode,
    ode_dfdx=ode_dfdx,
    ode_dfdp=ode_dfdp,
    times=times,
    inputs=inputs,
    parameters=p
)
```

Code Sample 5. Define the full model.

For a full list of optional arguments, we refer to the documentation of the package.

2.3.6 Optimality Criteria

In the next step we choose the objective function also called optimality criterion. Optimization of the different objectives leads to different experimental design outcomes. The most popular criteria are [3, 31, 32]:

- **D-optimality criterion** maximizes the determinant $\det(F)$ of the FIM. Translated to the parameter space, it means that the volume of the confidence region (see Fig. 2) (or the geometric mean of all errors) is minimized. The size of the confidence region is determined by a confidence level, which is typically chosen 90% or 95%. The confidence level can be interpreted as a probability value that the best parameter fit will belong to this area in case of repeating the experiment and estimations multiple times. The confidence region is usually presented as a confidence ellipsoid. D-optimality is the most widely used criterion and is suitable even in case of such parameter transformations as rescaling.
- **E-optimality criterion** maximizes the smallest eigenvalue $\lambda_{\min}$ of the FIM, which is the same as reducing only the largest estimation error.
- **A-optimality criterion** maximizes the sum of all eigenvalues $\sum_i \lambda_i$, which can be interpreted as minimizing the algebraic mean of all errors.
- **Modified E-optimality criterion** maximizes the ratio between the minimal and maximal eigenvalue $\lambda_{\min}/\lambda_{\max}$ and reduces correlation between two parameters corresponding to these eigenvalues.

Each of the criteria has its pros and cons, so the reader should have a closer look at the various properties of these criteria, for instance, in Franceschini's and Macchietto's paper [7]. The geometrical interpretation of the criteria using the confidence region is shown in Fig. 2.

2.3.7 Optimization

After defining the objective function based on the FIM the next step is to optimize the chosen optimality criteria. Finding the experimental design corresponds to finding the global maximum of the objective, which can be formulated as an optimal control problem [33]. This problem has been widely studied, and multiple numerical solution algorithms for both local and global optimization were introduced [34–37]. In the supplied toolbox *eDPM*, three methods were implemented: Differential evolution, basin-hopping, and the so-called "brute force" method. We give in the following a brief summary of the implemented methods. For more information the reader should turn to the available literature, e.g., [38, 39].

The differential evolution algorithm developed by Storn and Price (1996) [38] is one of the stochastic global optimization methods appropriate for nonlinear dynamic problems. Such types of algorithms have mild computational load but one cannot be sure that the absolute optimum is reached [3]. For differential evolution an initial population of candidate vectors for the experimental design (sampling times and inputs) is randomly chosen from the region of available values. Then each vector mutates by mixing with one of other candidate vectors. To a chosen vector from the initial population D_0, we add a weighted difference between two other randomly chosen vectors from the same set $(D_{\text{rand1}} - D_{\text{rand2}})$. This process is called mutation and a new vector D_m is obtained. The next step is to construct a new trial solution. This is done by choosing the elements of the trial vector either from the initial D_0 or the mutated D_m vectors. For each new element of trial vector, a random number drawn uniformly from the interval [0, 1) and compared to the so-called recombination constant. If this random number is less than the recombination constant, then the trial vector element is chosen from the vector D_m, otherwise from D_0. The degree of mutation can be controlled by changing the recombination constant; the larger this constant is, the more often vector elements are chosen from D_m. Subsequently, the objective function is calculated using the trial vector and the result is compared to result obtained using the initial solution D_0; and the best of them is chosen for the next generation. This procedure is repeated for every solution candidate of the initial population, by means of which the new generation is build [40]. The process of population mutation is repeated till a stopping criterion is reached, e.g., the maximum number of generations (steps) is reached or the standard deviation of the candidate vectors is below a certain threshold [41]. This method is rather simple and straightforward, does not require any gradient calculation and is easy to parallelize. Another advantages include its fast convergence and robustness at numerical optimization [42].

The second implemented method is basin-hopping developed by David Wales and Jonathan Doye [39]. This iterative algorithm combines Monte-Carlo and local optimization and works as follows: During the iteration step the design vector, i.e., vector of measurement times and inputs, is subject to a random perturbation and then to local minimization. After this, the step is either accepted or rejected. As in a standard Monte-Carlo method, the decision is made using the Metropolis criterion for the objective function [43]. For a deeper understanding of the Monte-Carlo minimization, the reader can refer to the papers [44, 45].

And lastly, a simple brute force method was implemented as well. It is a grid search algorithm calculating the objective function value at each point of a multidimensional grid in a chosen region. This method is suitable for discrete optimization with limited number of grid values. However, the downside of this technique is its slowness, inefficiency and long computational times, noticeable for higher numbers of possible discrete solutions [46]. The papers of Banga *et al.* [35, 47, 48] list additional information about this topic.

2.3.8 Optimization Code

To run the optimization procedure the following function can be used:

```
fsr = find_optimal(fsm)
```

This code will produce the output shown in Fig. 4 and 5. The user can choose between any of the three previously explained optimization procedures. Any optimization argument of the aforementioned routines, can be specified. The default arguments of the `"scipy_differential_evolution"` optimization method are given by the scipy default arguments [40]. For our specific use-case, we advise to experiment with the options presented in Code Sample 6. A full list of optional arguments can be seen

```
fsr = find_optimal(
    # Required argument: The model to optimize
    fsm,
    # The optimization method
    optimization_strategy="scipy_differential_evolution",
    # Our custom options
    criterion=fisher_determinant,
    relative_sensitivities=True,
    # Options from scipy.optimize.differential_evolution
    recombination=0.7,
    mutation=(0.1, 0.8),
    workers=-1,
    popsize=10,
    polish=False,
)
```

Code Sample 6. Define the optimization conditions and calculate the resulting Experimental design.

in the scipy documentation [49]. In addition, there are some interesting optimization options such as the optional arguments `relative_sensitivities,criterion`, which are responsible for using relative sensitivities $\frac{dy}{dp}\frac{p}{y}$ and specifying the optimality criterion as explained in the previous section. Please view the full documentation for explanation. The resulting class `fsr` contains all definitions, current values and information on the optimization process. Note that our toolbox optimizes and provides the timepoints with an assumption that the initial timepoint t_0 is always measured. For the parameter estimation the initial values $\mathbf{x}(t_0)$ of the system are required. Hence, the reader should keep that in mind and include the initial time t_0 as a measurement point in the final OED.

2.3.9 Identifiability

Before using the optimal experimental design generated in the preceding section, the reader needs to check if the parameters of the system are identifiable, i.e., to examine if it is possible to obtain a unique solution for the parameters from the optimization. It can happen that a subset of the parameters is non-identifiable. The non-identifiability can be due to the model structure or observables (structural non-identifiability) or insufficient data (practical non-identifiability) [50–52]. Structural non-identifiability should be avoided as it results in at least one parameter which cannot be determined. Fortunately, there is a quick and easy way to check whether the system is structural non-identifiable by calculating the rank of the sensitivity matrix [53, 54]. For an identifiable system, the rank should coincide with the number of estimated parameters. This method allows for the analysis of the local structural identifiability for already chosen inputs and times. Hence, we can check if the final result of the optimization routine is able to identify every parameter by this method. This does not replace a systematic inspection the model but at least we can validate our results. If a non-identifiability is detected, the structure of the model or number of measurements does not allow determination of the involved parameters. This means we need to adjust our description and check all supplied parameters for their validity. On the other hand, there are also *a priori* methods, e.g., using Lie group theory, that only require model definition [51]. However, due to the extensiveness of the topic we constrain ourselves to the method discussed above. Practical non-identifiability results in large confidence intervals [23, 53, 55]. Methods such as profile-likelihood [51] are able to test for these cases but they require experimental data.

2.3.10 Identifiability Code

Using the *eDPM* package, the reader can quite easily check if the resulting experimental design is structurally identifiable. For this, one can call the function that compares the rank of the sensitivity matrix with the number of parameters as in Code Sample 7. The function returns `True`, if the identifiability condition is reached, and `False` otherwise. If this test is not passed, the reader should either reconsider the model structure including the definition of parameters or increase the number of numerically optimized measurement conditions (inputs or times). From a mathematical point of view, we need at least as many measurement points as we have parameters in our system. Note that the described test above only allows to exclude structural

```
    # Check the structural identifiability
    check_if_identifiable(fsr)
```

Code Sample 7. Check the identifiability the optimizaton result.

identifiability but to obtain reasonable confidence intervals for parameter estimates the practical identifiability should be considered as well. Therefore, in reality more experiments are needed to increase the accuracy of the model. Thus, we suggest that the reader considers the optimization result not as a finished design but as a reference pointing to the minimal requirements and the most crucial conditions (inputs and times) for the parameter estimations.

2.3.11 Plotting, Json, etc.

Our package also provides the option to save results into Json file and automatically plot results for the ode solutions, observables and sensitivities (see Code Sample 8).

```
    plot_all_observables(fsr)
    plot_all_sensitivities(fsr)
    json_dump(fsr, "baranyi.json").
```

Code Sample 8. Plot the observables and sensitivities calculated for the OED and save the result in the Json format.

3 Results

3.1 Baranyi-Roberts Model - Single Species

As a summary of this tutorial, we would like to present the resulting output of our package. To this end we study the growth of a bacterial colony consisting of a single species and describe it mathematically using the Baranyi-Roberts model given by Eqs. (3),(4). We aim to estimate the parameters of the model and determine optimal measurement conditions. The observable is the bacterial count, i.e., the first component of the state variable vector $y = x_1$. As an additional constraint we need to consider that the available climate chambers can only operate the temperature in the range from 2 to 12 degrees, and the experiment should not take longer than 100 hours. The parameter values and initial values of the system are taken in accordance with the literature [56] and presented in Table 2. To model realistic uncertainties, we choose the error model given by Eq. (13) with $\gamma_{\text{abs}} = 0.3$ and $\gamma_{\text{rel}} = 0.1$ for all the measurement points. We identify the three parameters of our system by choosing 4 independent measurement conditions consisting of two temperature-points and two time-points. In our experimental design optimization routine, we use the D-optimality criterion, relative sensitivities, and the differential evolution optimization method. The resulting OED for the described system is presented in Fig. 3, 7. The identifiability test confirms the validity of our model setup (Fig. 6). For the system described above the minimal requirements for passing the identifiability test is using a setup with at least two temperatures and two measurement times per temperature. The choice of minimal number of measurement times can also be justified when looking at Fig. 8. Here we see that one measurement time is not enough as the determinant is numerically very close to zero indicating a non-identifiability of this system configuration. Fig. 7 shows that each of the chosen optimal time points corresponds to the maximum of at least one local sensitivity curve. This maximizes the determinant of the Fisher information matrix.

3.2 Two-Species Resource Competition

As a second example we discuss a slightly more complicated system and extend the previous example of bacterial growth to a system where two species are present. The species interact by competitive inhibition because they depend on a common nutrient resource. Analogously to the previous example, we denote the concentration of the two different species as x_1 and y_1, respectively. The system can be described by the following set of ODEs:

$$\begin{cases} \dot{x}_1 = \alpha_x R x_1 \\ \dot{y}_1 = \alpha_y R y_1 \\ \dot{R} = -\frac{R}{n_{\max}}(\alpha_x x_1 + \alpha_y y_1) \end{cases} \tag{24}$$

where α_x, α_y are the time and temperature dependent growth rates for population x_1, y_1, and R represents the nutrient pool concentration, respectively. Using the conservation quantity $x_1 + y_1 + n_{\max}R = n_{\max}$ this system can be reduced to:

$$\begin{cases} \dot{x}_1 = \alpha_x x_1 \left(1 - \frac{x_1+y_1}{n_{\max}}\right) \\ \dot{y}_1 = \alpha_y y_1 \left(1 - \frac{x_1+y_1}{n_{\max}}\right). \end{cases} \tag{25}$$

Based on the one species case for the growth rates we use [20, 22]

$$\alpha_x(t,T) = b_x^2 (T - T_{\min,x})^2 \frac{x_2(t)}{x_2(t)+1} \tag{26}$$

$$\alpha_y(t,T) = b_y^2 (T - T_{\min,y})^2 \frac{y_2(t)}{y_2(t)+1} \tag{27}$$

Combining equations (25, 27), we get the system of four differential equations:

$$\begin{cases} \dot{x}_1 = b_x^2 (T - T_{\min,x})^2 \frac{x_2}{x_2+1} x_1 (1 - \frac{x_1+y_1}{n_{\max}}) \\ \dot{x}_2 = b_x^2 (T - T_{\min,x})^2 x_2 \\ \dot{y}_1 = b_y^2 (T - T_{\min,y})^2 \frac{y_2}{y_2+1} y_1 (1 - \frac{x_1+y_1}{n_{\max}}) \\ \dot{y}_2 = b_y^2 (T - T_{\min,y})^2 y_2 \end{cases} \tag{28}$$

where x_2, y_2 are the concentration of the quantities determining the critical substance (nutrient) needed for growth of the species x_1 and y_1, respectively. The values of the parameter vector $\mathbf{p} = (n_{\max}, b_x, T_{\min,x}, b_y, T_{\min,y})$ and the vector of the initial values $\mathbf{x}(t_0) = (x_1(t_0), x_2(t_0), y_1(t_0), y_2(t_0))$ are presented in Table 3. In comparison to the previous example, we now choose the count of the two bacteria species x_1, y_1 as observables. Meanwhile, other optimization arguments were taken the same as in the previous example i.e., we optimize measurement times and temperatures. The new OED is shown in Fig. 9. The presented design shows that the least needed number of experiments is two with temperatures of 2°C and 12°C. In each experiment two observables that correspond to the concentrations of two bacteria types and two time points were measured. Note that for both temperatures at least two time-points are required for identifiability. Thus, we can estimate the parameters by performing experiments according to this design.

4 Conclusion

We introduced the concepts of experimental design and identifiability and their practical applications to systems described by ODEs. In addition, we implemented the numerical calculations in simple python code which we provide within our software *eDPM*. Our toolbox provides a simple way to apply methods of mathematical statistics and optimization. We demonstrated how to use these methods by applying them to the well-known Baranyi-Roberts model for microbial growth with one and two species. These results show us how parameters of the model can now be estimated most precisely by designing experiments around the predicted measurement conditions. In summary, model-based experimental design can accelerate and simplify the planning of efficient experiments for parameter estimation in microbiology.

Figures

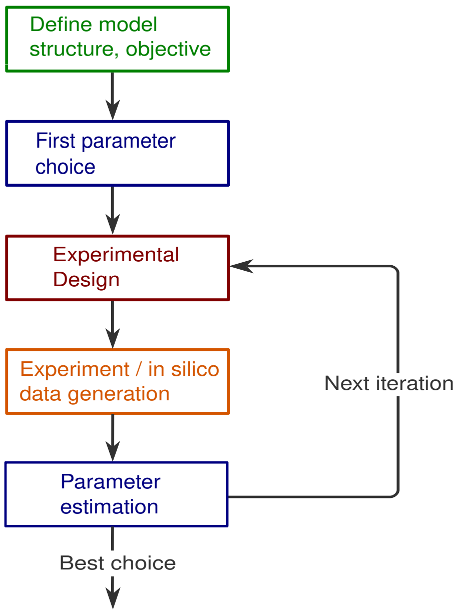


Fig. 1. The workflow of the iterative process for model-based experimental design for parameter estimation.

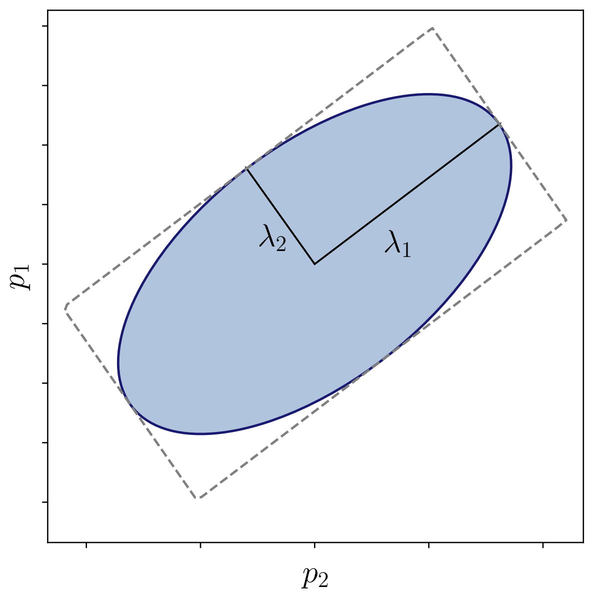


Fig. 2. The confidence ellipsoid projection to the (p_1, p_2) parameter space. The ellipsoid shows the geometrical meaning of the optimality criteria. The center of the ellipsoid represents the estimated parameter values. The radii are the uncertainties of the estimates associated with different eigenvalues of the FIM λ_1, λ_2 ($\lambda_1 < \lambda_2$). The D-optimality aims to minimize the volume of the ellipsoid, the E-optimality minimizes the largest radius, A-optimality minimizes the perimeter of the rectangle that encloses the ellipse (dashed gray line), and, finally, modified E-optimality tends to make the ellipse as spherical as possible.

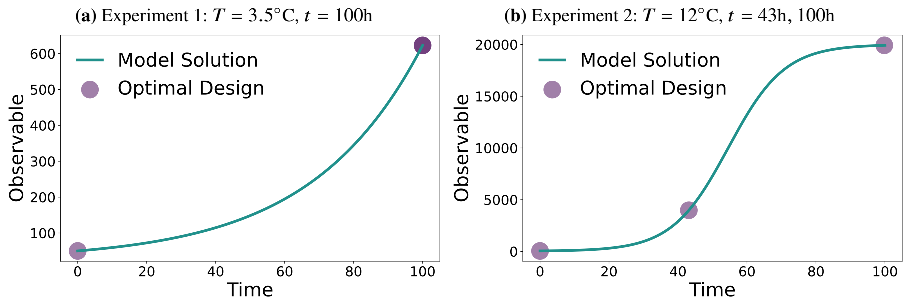
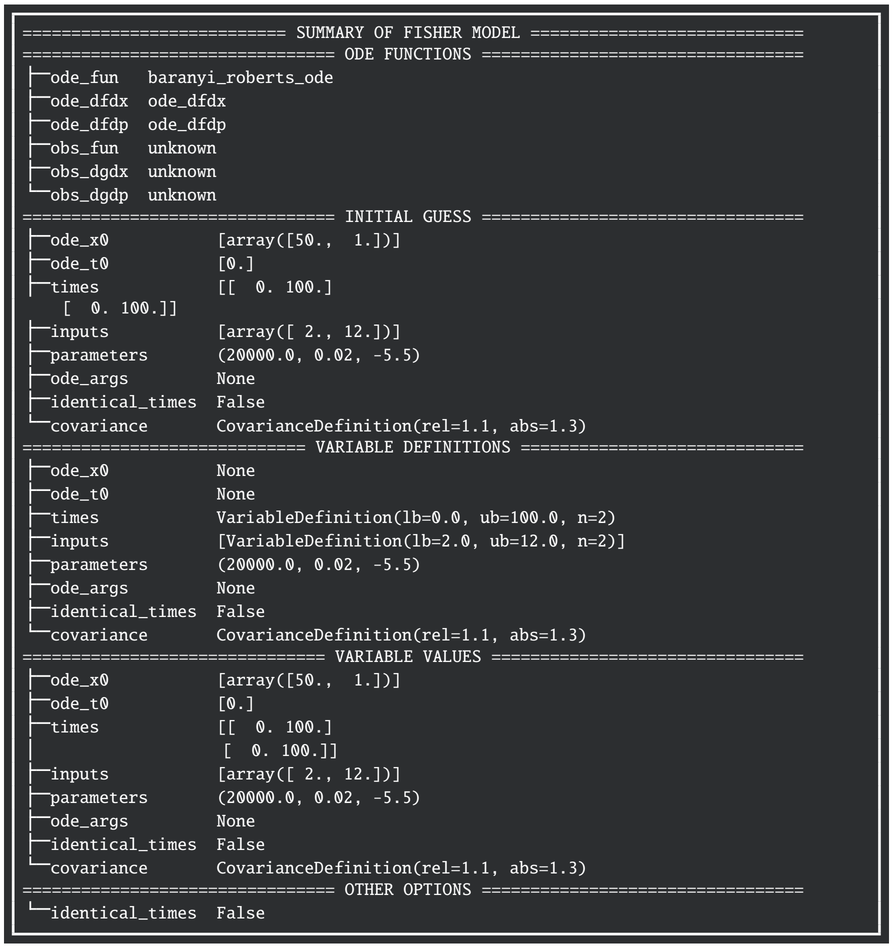
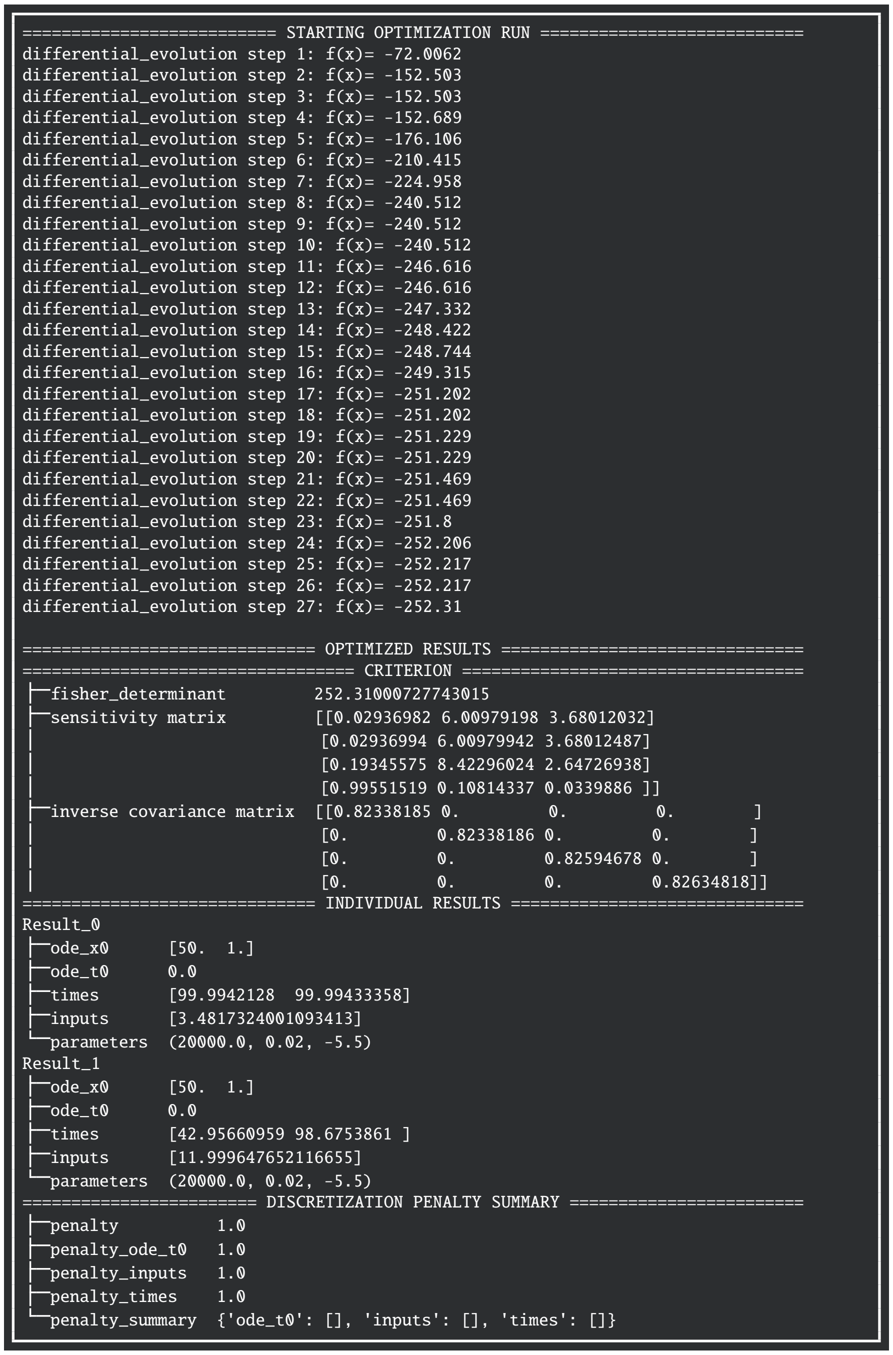


Fig. 3. The example of the output of the experimental design optimization procedure for the Baranyi-Roberts model. The line plot presents the model solution for the observable, and the circles determine the suggested by experimental design time points. The optimal design is proposed for one observable which is the total bacterial count of the species x_1 and consists of two time series where samples are stored at temperatures (a) T_1 = 3.5°C and (b) T_2 = 12°C. The corresponding measurement times are (a) t_{11} = 100h and (b) t_{21} = 43h, t_{22} = 100h. The initial time $t_0 = 0$ is included in the experimental design by definition.



```
========================== SUMMARY OF FISHER MODEL ============================
=============================== ODE FUNCTIONS ================================
├─ode_fun   baranyi_roberts_ode
├─ode_dfdx  ode_dfdx
├─ode_dfdp  ode_dfdp
├─obs_fun   unknown
├─obs_dgdx  unknown
└─obs_dgdp  unknown
=============================== INITIAL GUESS ================================
├─ode_x0           [array([50.,  1.])]
├─ode_t0           [0.]
├─times            [[  0. 100.]
   [  0. 100.]]
├─inputs           [array([ 2., 12.])]
├─parameters       (20000.0, 0.02, -5.5)
├─ode_args         None
├─identical_times  False
└─covariance       CovarianceDefinition(rel=1.1, abs=1.3)
============================ VARIABLE DEFINITIONS ============================
├─ode_x0           None
├─ode_t0           None
├─times            VariableDefinition(lb=0.0, ub=100.0, n=2)
├─inputs           [VariableDefinition(lb=2.0, ub=12.0, n=2)]
├─parameters       (20000.0, 0.02, -5.5)
├─ode_args         None
├─identical_times  False
└─covariance       CovarianceDefinition(rel=1.1, abs=1.3)
============================== VARIABLE VALUES ===============================
├─ode_x0           [array([50.,  1.])]
├─ode_t0           [0.]
├─times            [[  0. 100.]
│                   [  0. 100.]]
├─inputs           [array([ 2., 12.])]
├─parameters       (20000.0, 0.02, -5.5)
├─ode_args         None
├─identical_times  False
└─covariance       CovarianceDefinition(rel=1.1, abs=1.3)
=============================== OTHER OPTIONS ================================
└─identical_times  False
```

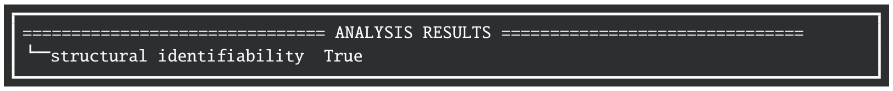
Fig. 4. Command-line output of *eDPM* of all parameters, inputs, ODE functions and initial values needed to run the optimization method.



```
========================== STARTING OPTIMIZATION RUN ===========================
differential_evolution step 1: f(x)= -72.0062
differential_evolution step 2: f(x)= -152.503
differential_evolution step 3: f(x)= -152.503
differential_evolution step 4: f(x)= -152.689
differential_evolution step 5: f(x)= -176.106
differential_evolution step 6: f(x)= -210.415
differential_evolution step 7: f(x)= -224.958
differential_evolution step 8: f(x)= -240.512
differential_evolution step 9: f(x)= -240.512
differential_evolution step 10: f(x)= -240.512
differential_evolution step 11: f(x)= -246.616
differential_evolution step 12: f(x)= -246.616
differential_evolution step 13: f(x)= -247.332
differential_evolution step 14: f(x)= -248.422
differential_evolution step 15: f(x)= -248.744
differential_evolution step 16: f(x)= -249.315
differential_evolution step 17: f(x)= -251.202
differential_evolution step 18: f(x)= -251.202
differential_evolution step 19: f(x)= -251.229
differential_evolution step 20: f(x)= -251.229
differential_evolution step 21: f(x)= -251.469
differential_evolution step 22: f(x)= -251.469
differential_evolution step 23: f(x)= -251.8
differential_evolution step 24: f(x)= -252.206
differential_evolution step 25: f(x)= -252.217
differential_evolution step 26: f(x)= -252.217
differential_evolution step 27: f(x)= -252.31

============================== OPTIMIZED RESULTS ===============================
================================== CRITERION ===================================
 ├─fisher_determinant         252.31000727743015
 ├─sensitivity matrix         [[0.02936982 6.00979198 3.68012032]
 │                             [0.02936994 6.00979942 3.68012487]
 │                             [0.19345575 8.42296024 2.64726938]
 │                             [0.99551519 0.10814337 0.0339886 ]]
 ├─inverse covariance matrix  [[0.82338185 0.         0.         0.        ]
 │                             [0.         0.82338186 0.         0.        ]
 │                             [0.         0.         0.82594678 0.        ]
 │                             [0.         0.         0.         0.82634818]]
============================== INDIVIDUAL RESULTS ==============================
Result_0
 ├─ode_x0      [50.  1.]
 ├─ode_t0      0.0
 ├─times       [99.9942128  99.99433358]
 ├─inputs      [3.4817324001093413]
 └─parameters  (20000.0, 0.02, -5.5)
Result_1
 ├─ode_x0      [50.  1.]
 ├─ode_t0      0.0
 ├─times       [42.95660959 98.6753861 ]
 ├─inputs      [11.999647652116655]
 └─parameters  (20000.0, 0.02, -5.5)
======================== DISCRETIZATION PENALTY SUMMARY ========================
 ├─penalty          1.0
 ├─penalty_ode_t0   1.0
 ├─penalty_inputs   1.0
 ├─penalty_times    1.0
 └─penalty_summary  {'ode_t0': [], 'inputs': [], 'times': []}
```

Fig. 5. Command-line output which summarizes of the optimization routine and its results.



```
============================== ANALYSIS RESULTS ==============================
 └─structural identifiability  True
```

Fig. 6. Command-line output showing the result of the identifiability check for the experimental design provided in Figure 3. The output shows that the experimental design gives structurally identifiable result and allows parameter estimation of the system.

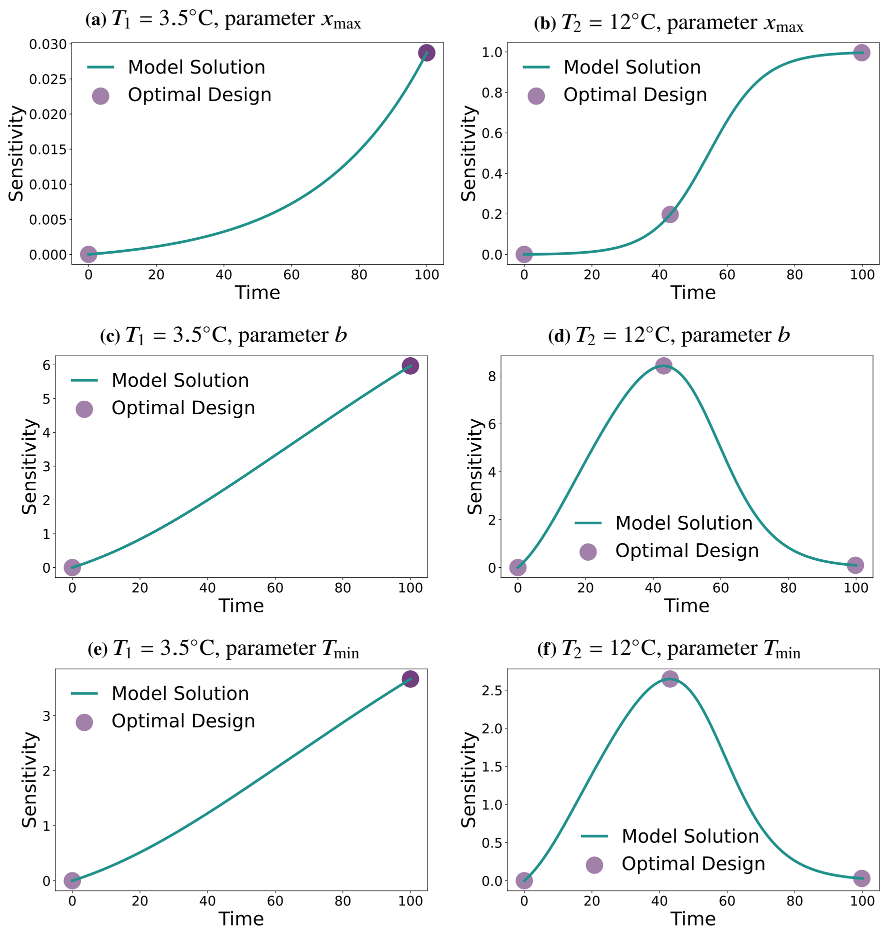


Fig. 7. The example of the sensitivities of the total bacterial count calculated for the experimental design optimization procedure for the Baranyi-Roberts model. The line plot presents the model solution for the observable, and the circles determine the suggested by experimental design time points. In (a), (c), (e) the sensitivity curves for the experimental series at storage temperature T_1 = 3.5°C are shown for parameters $x_{\max}$, b, $T_{\min}$, respectively. As well as, (b), (d), (f) subfigures present the sensitivities for the T_2 = 12°C experiment for parameters $x_{\max}$, b, $T_{\min}$, respectively.

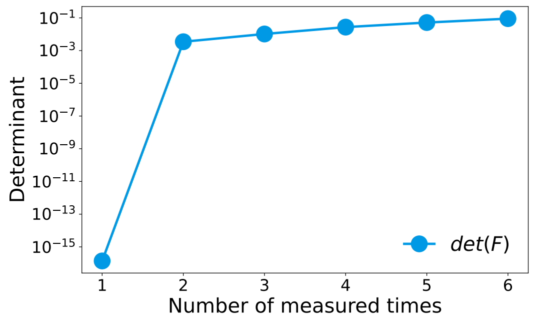


Fig. 8. The determinant of the FIM as a function of the number of measurement times for the experimental design consisted of two measured temperatures.

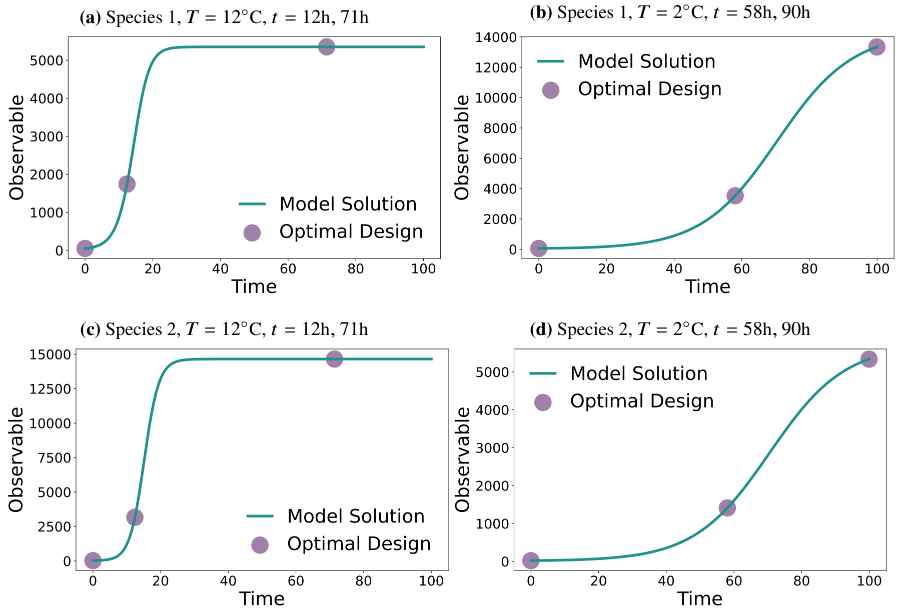


Fig. 9. The optimal experimental design for the Baranyi-Roberts model with two different species (28) is proposed for two observables: the bacterial count of the first species x_1 and of the second species x_2. (a), (c) show the first experimental series held at the temperature $T_1 = 12°\text{C}$ and with measurement times $t_{11} = 12\text{h}$, $t_{12} = 71\text{h}$ for observables x_1 and x_2, respectively. (b), (d) show the second experimental series held at the temperature $T_2 = 2°\text{C}$ and with measurement times $t_{21} = 58\text{h}$, $t_{22} = 90\text{h}$ for observables x_1 and x_2, respectively. The line plot presents the model solution for the observable, and the circles determine the suggested by experimental design time points. The initial time $t_0 = 0$ is included in the experimental design by definition.

Tables

Description	Formula	Code
ODE	$\mathbf{f}$	`def ode_fun(t, x, u, p, ode_args)`
	$\partial\mathbf{f}/\partial\mathbf{x}$	`def ode_dfdx(t, x, u, p, ode_args)`
	$\partial\mathbf{f}/\partial\mathbf{p}$	`def ode_dfdp(t, x, u, p, ode_args)`
Observable (optional)	$\mathbf{g}$	`def obs_fun(t, x, u, p, ode_args)`
	$\partial\mathbf{g}/\partial x$	`def obs_dgdx(t, x, u, p, ode_args)`
	$\partial\mathbf{g}/\partial p$	`def obs_dgdp(t, x, u, p, ode_args)`
Initial value	$\mathbf{x}_0$	`x0`
Initial time	t_0	`t0`
Time points	t_i	`t`
Inputs	$\mathbf{u}$	`u`
Parameters	$\mathbf{p}$	`p`
Other arguments		`ode_args`

Table 1. Summary of user-defined functions and variables.

Variable	Value	Units
$n_{\max}$	$2 \cdot 10^4$	cfu/g
b	0.2	$°C^{-1}h^{-1/2}$
$T_{\min}$	-5.5	°C
$x_1(t_0)$	50.0	cfu/g
$x_2(t_0)$	1.0	1

Table 2. The list of the parameter and ODE's initial values of the system that fully define the system (3),(4). The values correspond to estimated parameters for Pseudomonas spp. in poultry under aerobic conditions [56].

Variable	Value	Units
$n_{\max}$	$2 \cdot 10^4$	cfu/g
b_x	0.03	$°C^{-1}h^{-1/2}$
$T_{\min,x}$	-8.0	°C
b_y	0.04	$°C^{-1}h^{-1/2}$
$T_{\min,y}$	-5.5	°C
$x_1(t_0)$	50.0	cfu/g
$x_2(t_0)$	1.0	1
$y_1(t_0)$	20.0	cfu/g
$y_2(t_0)$	1.0	1

Table 3. The list of the parameter and ODE's initial values of the system that fully define the system (28).

Supporting information

References

1. Kristel Bernaerts, Els Dens, Karen Vereecken, Annemie H. Geeraerd, Arnout R. Standaert, Frank Devlieghere, Johan Debevere, and Jan F. Van Impe. Concepts and Tools for Predictive Modeling of Microbial Dynamics. *Journal of Food Protection*, 67(9):2041–2052, September 2004.
2. Eva Van Derlinden, Laurence Mertens, and Jan F. Van Impe. The impact of experiment design on the parameter estimation of cardinal parameter models in predictive microbiology. *Food Control*, 29(2):300–308, 2013.
3. J.R. Balsa-Canto, E. Banga. Computing optimal dynamic experiments for model calibration in Predictive Microbiology. *Journal of Food Process Engineering*, 31, 2008.
4. Clemens Kreutz and Jens Timmer. Systems biology: Experimental design. *The FEBS journal*, 276(4):923–942, February 2009.
5. I. Stamati, S. Akkermans, F. Logist, E. Noriega, and J. Van Impe. Optimal experimental design for discriminating between microbial growth models as function of suboptimal temperature: From in silico to in vivo. *Food Research International*, 89:689–700, 2016.
6. A. C. Atkinson. Developments in the Design of Experiments, Correspondent Paper. *International Statistical Review / Revue Internationale de Statistique*, 50(2):161–177, 1982.
7. Gaia Franceschini and Sandro Macchietto. Model-based design of experiments for parameter precision: State of the art. *Chemical Engineering Science*, 63(19):4846–4872, 2008.
8. Jianyong Sun, Jonathan M Garibaldi, and Charlie Hodgman. Parameter Estimation Using Meta-Heuristics in Systems Biology: A Comprehensive Review. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, March 2011.
9. Carlos Vilas, Ana Arias-Méndez, Míriam R García, Antonio A Alonso, and E Balsa-Canto. Toward predictive food process models: A protocol for parameter estimation. *Critical Reviews in Food Science and Nutrition*, 27:1–14, May 2016.
10. D. Telen, F. Logist, E. Van Derlinden, I. Tack, and J. Van Impe. Optimal experiment design for dynamic bioprocesses: A multi-objective approach. *Chemical Engineering Science*, 78:82–97, 2012.
11. Filip Logist, Boris Houska, Moritz Diehl, and Jan F. Van Impe. Robust multi-objective optimal control of uncertain (bio)chemical processes. *Chemical Engineering Science*, 66(20):4670–4682, 2011.
12. K J Versyck, K Bernaerts, A H Geeraerd, and J F Van Impe. Introducing optimal experimental design in predictive modeling: A motivating example. *International Journal of Food Microbiology*, 51(1):39–51, October 1999.

13. Míriam R. García, Carlos Vilas, Juan R. Herrera, Marta Bernárdez, Eva Balsa-Canto, and Antonio A. Alonso. Quality and shelf-life prediction for retail fresh hake (Merluccius merluccius). *International Journal of Food Microbiology*, 208:65–74, 2015.
14. Eva Balsa-Canto, David Henriques, Attila Gábor, and Julio R. Banga. AMIGO2, a toolbox for dynamic modeling, optimization and control in systems biology. *Bioinformatics*, 32(21):3357–3359, November 2016.
15. Jeff F. Zhang, Nikos E. Papanikolaou, Theodore Kypraios, and Christopher C. Drovandi. Optimal experimental design for predator–prey functional response experiments. *Journal of The Royal Society Interface*, 15(144):20180186, July 2018. Publisher: Royal Society.
16. Alberto Giovanni Busetto, Alain Hauser, Gabriel Krummenacher, Mikael Sunnåker, Sotiris Dimopoulos, Cheng Soon Ong, Jörg Stelling, and Joachim M. Buhmann. Near-optimal experimental design for model selection in systems biology. *Bioinformatics*, 29(20):2625–2632, October 2013.
17. Guido van Rossum and Fred L. Drake. *The Python Language Reference*. Number Pt. 2 in Python Documentation Manual / Guido van Rossum; Fred L. Drake [Ed.]. Python Software Foundation, Hampton, NH, release 3.0.1 [repr.] edition, 2010.
18. Jupyter Team. Jupyter Notebook. https://jupyter.org.
19. J Pleyer and P Gaindrik. eDPM - experimental Design for Predictive Microbiology. `https://github.com/Spatial-Systems-Biology-Freiburg/eDPM`. Accessed: 21.09.2023.
20. József Baranyi and Terry A. Roberts. A dynamic approach to predicting bacterial growth in food. *International Journal of Food Microbiology*, 23(3-4):277–294, November 1994.
21. K Grijspeerdt and P Vanrolleghem. Estimating the parameters of the Baranyi model for bacterial growth. *Food Microbiology*, 16(6):593–605, December 1999.
22. D A Ratkowsky, J Olley, T A McMeekin, and A Ball. Relationship between temperature and growth rate of bacterial cultures. *Journal of Bacteriology*, 149(1):1–5, January 1982. Publisher: American Society for Microbiology.
23. Clemens Kreutz, Andreas Raue, Daniel Kaschek, and Jens Timmer. Profile likelihood in systems biology. *The FEBS journal*, May 2013.
24. Attila Gábor and Julio R. Banga. Robust and efficient parameter estimation in dynamic models of biological systems. *BMC Systems Biology*, 9(1):74, December 2015.
25. A Ly, M Marsman, J Verhagen, RPPP Grasman Journal of Mathematical, and 2017. A tutorial on Fisher information. *Elsevier*, 80:40–55, October 2017.
26. Roy Frieden and Robert A Gatenby. *Exploratory Data Analysis Using Fisher Information*. Springer Science & Business Media, London, May 2010.
27. H. T. Banks, Sava Dediu, Stacey L. Ernstberger, and Franz Kappel. Generalized sensitivities and optimal experimental design. 18(1):25–83, April 2010. Publisher: De Gruyter Section: Journal of Inverse and Ill-posed Problems.
28. J D Stigter, D Joubert, and J Molenaar. Observability of Complex Systems: Finding the Gap. *Scientific Reports*, pages 1–9, November 2017.
29. A. Cintrón-Arias, H. T. Banks, A. Capaldi, and A. L. Lloyd. A sensitivity matrix based methodology for inverse problem formulation. 17(6):545–564, August 2009. Publisher: De Gruyter Section: Journal of Inverse and Ill-posed Problems.
30. E Balsa-Canto, J R Banga, and A A Alonso. Computational procedures for optimal experimental design in biological systems. *IET systems biology*, 2(4):163–172, July 2008.
31. Holger Dette. Designing Experiments with Respect to 'Standardized' Optimality Criteria. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(1):97–110, 1997. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9868.00056.
32. E. Walter and L. Pronzato. Qualitative and quantitative experiment design for phenomenological models—A survey. *Automatica*, 26(2):195–213, March 1990.
33. D. Espie and S. Macchietto. The optimal design of dynamic experiments. *AIChE Journal*, 35(2):223–229, 1989. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690350206.
34. William R. Esposito and Christodoulos A. Floudas. Global Optimization for the Parameter Estimation of Differential-Algebraic Systems. *Ind. Eng. Chem. Res.*, 39(5):1291–1310, May 2000. Publisher: American Chemical Society.

35. Julio R. Banga, Eva Balsa-Canto, Carmen G. Moles, and Antonio A. Alonso. Improving food processing using modern optimization methods. *Trends in Food Science & Technology*, 14(4):131–144, 2003.
36. M. M. Ali, A. Törn, and S. Viitanen. A Numerical Comparison of Some Modified Controlled Random Search Algorithms. *Journal of Global Optimization*, 11(4):377–385, December 1997.
37. T.P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000. Conference Name: IEEE Transactions on Evolutionary Computation.
38. Rainer Storn and Kenneth Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 4(11):341–359, 1997.
39. David J. Wales and Jonathan P. K. Doye. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *J. Phys. Chem. A*, 101(28):5111–5116, July 1997.
40. The SciPy community. Documentation scipy.optimize.differential_evolution. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html`. Accessed: 26.09.2023.
41. K. Zielinski, P. Weitkemper, R. Laur, and R. Laur K.-D. Kammeyer, K. Zielinski. Examination of stopping criteria for differential evolution based on a power allocation problem. In *10th International Conference on Optimization of Electrical and Electronic Equipment*, Brasov, Romania, May 2006.
42. B. V. Babu and S. A. Munawar. Differential evolution strategies for optimal design of shell-and-tube heat exchangers. *Chemical Engineering Science*, 62(14):3720–3739, July 2007.
43. The SciPy community. Documentation scipy.optimize.basinhopping. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html`. Accessed: 26.09.2023.
44. Z Li and H A Scheraga. Monte Carlo-minimization approach to the multiple-minima problem in protein folding. *Proceedings of the National Academy of Sciences*, 84(19):6611–6615, October 1987. Publisher: Proceedings of the National Academy of Sciences.
45. I. Beichl and F. Sullivan. The Metropolis Algorithm. *Computing in Science & Engineering*, 2(1):65–69, January 2000. Conference Name: Computing in Science & Engineering.
46. The SciPy community. Documentation scipy.optimize.brute. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.brute.html`. Accessed: 26.09.2023.
47. Julio R. Banga and Warren D. Seider. Global Optimization of Chemical Processes using Stochastic Algorithms. In C. A. Floudas and P. M. Pardalos, editors, *State of the Art in Global Optimization: Computational Methods and Applications*, Nonconvex Optimization and Its Applications, pages 563–583. Springer US, Boston, MA, 1996.
48. Julio R. Banga, Eva Balsa-Canto, Carmen G. Moles, and Antonio A. Alonso. Dynamic optimization of bioprocesses: Efficient and robust numerical strategies. *Journal of Biotechnology*, 117(4):407–419, June 2005.
49. Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, March 2020.
50. Joseph H. A. Guillaume, John D. Jakeman, Stefano Marsili-Libelli, Michael Asher, Philip Brunner, Barry Croke, Mary C. Hill, Anthony J. Jakeman, Karel J. Keesman, Saman Razavi, and Johannes D. Stigter. Introductory overview of identifiability analysis: A guide to evaluating whether you have the right type of data for your modeling purpose. *Environmental Modelling & Software*, 119:418–432, September 2019.

51. Franz-Georg Wieland, Adrian L. Hauber, Marcus Rosenblatt, Christian Tönsing, and Jens Timmer. On structural and practical identifiability. *Current Opinion in Systems Biology*, 25:60–69, March 2021.
52. Eric Walter and Luc Pronzato. On the identifiability and distinguishability of nonlinear parametric models. *Mathematics and Computers in Simulation*, 42(2):125–134, October 1996.
53. Hongyu Miao, Xiaohua Xia, Alan S. Perelson, and Hulin Wu. On Identifiability of Nonlinear ODE Models and Applications in Viral Dynamics. *SIAM Rev.*, 53(1):3–39, January 2011. Publisher: Society for Industrial and Applied Mathematics.
54. Johannes D. Stigter and Jaap Molenaar. A fast algorithm to assess local structural identifiability. *Automatica*, 58:118–124, August 2015.
55. Andrea Holmberg. On the practical identifiability of microbial growth models incorporating Michaelis-Menten type nonlinearities. *Mathematical Biosciences*, 62(1):23–43, November 1982.
56. Radovan Gospavic, Judith Kreyenschmidt, Stefanie Bruckner, Viktor Popov, and Nasimul Haque. Mathematical modelling for predicting the growth of Pseudomonas spp. in poultry under variable temperature conditions. *International Journal of Food Microbiology*, 127(3):290–297, October 2008.