BAYESIAN CALIBRATION, PROCESS MODELING AND UNCERTAINTY QUANTIFICATION IN BIOTECHNOLOGY

Laura Marie Helleckes[+,1,2], **Michael Osthege**[+,1,2], **Wolfgang Wiechert**[1,2], **Eric von Lieres**[1,2], **Marco Oldiges**[*,1,2]

[+]Contributed equally
[1]Forschungszentrum Jülich GmbH, 52428 Jülich, Germany
[2]RWTH Aachen University, 52062 Aachen, Germany
[*]Corresponding author

July 1, 2021

ABSTRACT

High-throughput experimentation has revolutionized data-driven experimental sciences and opened the door to the application of machine learning techniques. Nevertheless, the quality of any data analysis strongly depends on the quality of the data and specifically the degree to which random effects in the experimental data-generating process are quantified and accounted for. Accordingly calibration, i.e. the quantitative association between observed quantities with measurement responses, is a core element of many workflows in experimental sciences. Particularly in life sciences, univariate calibration, often involving non-linear saturation effects, must be performed to extract quantitative information from measured data. At the same time, the estimation of uncertainty is inseparably connected to quantitative experimentation. Adequate *calibration models* that describe not only the input/output relationship in a measurement system, but also its inherent measurement noise are required. Due to its mathematical nature, statistically robust calibration modeling remains a challenge for many practitioners, at the same time being extremely beneficial for machine learning applications. In this work, we present a bottom-up conceptual and computational approach that solves many problems of understanding and implementing non-linear, empirical calibration modeling for quantification of analytes and process modeling. The methodology is first applied to the optical measurement of biomass concentrations in a high-throughput cultivation system, then to the quantification of glucose by an automated enzymatic assay. We implemented the conceptual framework in two Python packages, with which we demonstrate how it makes uncertainty quantification for various calibration tasks more accessible. Our software packages enable more reproducible and automatable data analysis routines compared to commonly observed workflows in life sciences. Subsequently, we combine the previously established calibration models with a hierarchical Monod-like differential equation model of microbial growth to describe multiple replicates of *Corynebacterium glutamicum* batch microbioreactor cultures. Key process model parameters are learned by both maximum likelihood estimation and Bayesian inference, highlighting the flexibility of the statistical and computational framework.

***K*eywords** nonlinear calibration · calibration modeling · quantitative measurement · process modeling · ODE modeling · maximum likelihood · Python · Bayesian methods · uncertainty quantification

1 Introduction

1.1 Calibration in life sciences

Calibration modeling is an omnipresent task in experimental science. Particularly the life sciences make heavy use of calibration modeling to achieve quantitative insights from experimental data. The importance of *calibration models* (also known as *calibration curves*) in bioanalytics is underlined in dedicated guidance documents by EMA and FDA [1, 2] that also make recommendations for many related aspects such as method development and validation. While liquid chromatography and mass spectrometry are typically calibrated with linear models [3], a four- or five-parameter logistic model is often used for immuno- or ligand-binding assays [2, 4–6]. The aforementioned guidance documents focus on health-related applications, but there are countless examples where (non-linear) calibration needs to be applied across biological disciplines. From dose-response curves in toxicology to absorbance or fluorescence measurements, or the calibration of *on-line* measurement systems, experimentalists are confronted with the task of calibration.
At the same time, recent advances in affordable liquid-handling robotics facilitate lab scientists in chemistry and biotechnology to (partially) automate their specialized assays (e.g. [7, 8]). Moreover, advanced robotic platforms for parallelized experimentation, monitoring and analytics [8, 9] motivate *on-line* data analysis and calibration for process control of running experiments.

1.2 Generalized computational methods for calibration

Experimental challenges in calibration are often unique to a particular field and require domain knowledge to be solved. At the same time, the statistical or computational aspects of the workflow can be generalized across domains. With the increased amount of available data in high-throughput experimentation comes the need for equally rapid data analysis and calibration. As a consequence, it is highly desirable to develop an automatable, technology-agnostic and easy-to-use framework for quantitative data analysis with calibration models.
From our perspective of working at the intersection between laboratory automation and modeling, we identified a set of requirements for calibration: Data analyses rely more and more on scripting languages such as Python or R, making the use of spreadsheet programs an inconvenient bottleneck. At various levels, and in particular when non-linear calibration models are involved, the statistically sound handling of uncertainty is at the core of a quantitative data analysis.
Before going into detail about the calibration workflow, we would like to highlight its most important aspects and terminology based on the definition of calibration by the *International Bureau of Weights and Measures* (BIPM) [10]:

> **2.39 calibration:** "Operation that, under specified conditions, in a first step, establishes a relation between the quantity values with measurement uncertainties provided by measurement standards and corresponding indications with associated measurement uncertainties and, in a second step, uses this information to establish a relation for obtaining a **measurement result** from an indication."
> **2.9 measurement result:** "[...] A measurement result is generally expressed as a single measured quantity value and a measurement uncertainty."

The "*first step*" from the BIPM definition is the establishment of a relation that we will call *calibration model* henceforth. In statistical terminology, the relationship is established between an *independent* variable (BIPM: *quantity values*) and a *dependent* variable (BIPM: *indications*) and it is important to note that the description of measurement uncertainty is a central aspect of a calibration model. In the application (*"second step"*) of the calibration model, the quantification of uncertainty is a core aspect as well.
Uncertainty arises from the fact that measurements are not exact, but subject to some form of random effects. While many methods assume that these random effects are distributed according to a *normal* distribution, we want to stress that a generalized framework for calibration should not make such constraints. Instead, domain experts should be enabled to choose a probability distribution that is best suited to describe their measurement system at hand.
Going beyond the BIPM definition, we see the application of calibration models two-fold:

- Inference of individual independent quantity values from one or more observations.
- Inferring the parameters of a more comprehensive *process model* from measurement responses obtained from (samples of) the system.

For both applications, uncertainties should be a standard outcome of the analysis. In life sciences, the commonly used estimate of uncertainty is the *confidence interval*. The interpretation of confidence intervals however is challenging, as

it is often oversimplified and confused with other probability measures [11, 12]. Furthermore, their correct implementation for non-linear calibration models, and particularly in combination with complex process models, is technically demanding. For this reason, we use Bayesian *credible intervals* that are interpreted as the range in which an unobserved parameter lies with a certain probability [13]. In 2.3 we go into more details about the uncertainty measures and how they are obtained and interpreted.

Even though high-level conventions and recommendations exist, the task of calibration is approached with different statistical methodology across the experimental sciences. In laboratory automation, we see a lack of tools enabling practitioners to build tailored calibration models while maintaining a generalized approach. At the same time, generalized calibration models have the potential to improve adequacy of complex simulations in the related fields.

While numerous software packages for modeling biological systems are available, most are targeted towards complex biological networks and do not consider calibration modeling or application to large hierarchical datasets. Notable examples are Data2Dynamics [14] or PESTO [15], both allowing to customize calibration models and the way the measurement error is described. However, both tools are implemented in MATLAB and are thus incompatible with data analysis workflows that leverage the rich ecosystem of scientific Python libraries. Here, Python packages such as PyCoTools3 [16] for the popular COPASI software [17] provide valuable functionality, but are limited with respect to custom calibration models, especially in a Bayesian modeling context. To the best of our knowledge, no Python framework exists so far that provides customized calibration models that are at the same time compatible with Bayesian modeling, provide profound uncertainty analysis and modular application with other libraries.

1.3 Aim of this study

This study aims to build an understanding of how *calibration models* can be constructed to describe both *location* and *spread* of measurement outcomes such that uncertainty can be quantified. In two parts, we demonstrate a toolbox for calibration models on the basis of application examples, thus showing how it directly addresses questions typical for quantitative data analysis.

In part one (Section 4.1) we demonstrate how to construct such calibration models based on a reparametrized asymmetric logistic function applied to a photometric assay. We give recommendations for obtaining calibration data and introduce accompanying open-source Python software that implements object-oriented calibration models with a variety of convenience functions.

In part two (Section 4.2) we show how calibration models can become part of elaborate *process models* to accurately describe measurement uncertainty caused by experimental limitations. We introduce a generic framework for refining a template process model into a hierarchical model that flexibly shares parameters across experimental replicates and connects the model prediction with observed data via the previously introduced calibration models. This generic framework is applied to build an ordinary differential equation (ODE) process model for 24 microbial growth curves gained in automated, high-throughput experiments. Finally, we demonstrate how the calibration model can be applied to perform maximum likelihood estimation or Bayesian inference of process model parameters while accounting for non-linearities in the experimental observation process.

Although this paper chooses biotechnological applications, the presented approach is generic and the framework thus applicable to a wide range of research fields.

2 Theoretical Background

2.1 Probability theory for calibration modeling

Probability distributions are at the heart of virtually all statistical and modeling methods. They describe the range of values that a variable of unknown value, also called *random variable*, may take, together with how likely these values are. This work focuses on *univariate* calibration tasks, where a continuous variable is obtained as the result of the measurement procedure. Univariate, continuous probability distributions such as the *Normal* or *Student's-t* distribution are therefore relevant in this context. Probability distributions are described by a set of parameters, such as $\{\mu, \sigma\}$ in the case of a Normal distribution, or $\{\mu, \text{scale}, \nu\}$ in the case of a Student's-t distribution.

To write that a random variable "rv" follows a certain distribution, the $\sim$ symbol is used: $\text{rv} \sim \textit{Student's-t}(\mu, \text{scale}, \nu)$.

The most commonly found visual representation of a continuous probability distribution is in terms of its *probability density function* (PDF, Figure 1), typically written as $p(\text{rv})$.

The term *rv conditioned on d* is used to refer to the probability that an observation of rv takes the value d. It is written as $p(\text{rv} \mid \text{d})$ and corresponds to the value of the PDF at position d.

A related term, the *likelihood* $\mathcal{L}$, takes the inverse perspective and is proportional to the probability of making the observation d, *given* that rv has a certain value (Equation 1). In practice, $\mathcal{L}(\text{rv} \mid \text{d})$ is often easy to access, whereas $p(\text{d} \mid \text{rv})$ is hard to compute analytically. Therefore, most methods for the estimation of model parameters (Section 2.2) exploit the proportionality and just use $\mathcal{L}$.

$$\mathcal{L}(\text{rv} \mid \text{d}) \propto p(\text{d} \mid \text{rv}) \tag{1}$$

When only considering the observed data, the probability of the random variable conditioned on data ($p(rv \mid d)$, can be obtained by normalizing the likelihood by its integral (Equation 2).

$$p(\text{rv} \mid \text{d}) = \frac{\mathcal{L}(\text{rv} \mid \text{d})}{\int \mathcal{L}(\text{rv} \mid \text{d})} \tag{2}$$

In situations where only limited data is available, a Bayesian statistician argues that *prior* information should be taken into account. The likelihood can then be combined with prior beliefs into a *posterior* probability according to Bayes' rule Equation 3.

$$p(\text{rv} \mid \text{d}) = \frac{p(\text{rv}) \cdot \mathcal{L}(\text{rv} \mid \text{d})}{\int p(\text{rv}) \cdot \mathcal{L}(\text{rv} \mid \text{d})} \tag{3}$$

According to Equation 3, the posterior probability $p(\text{rv} \mid \text{d})$ of the random variable rv given the data is equal to the product of prior probability times likelihood, divided by its integral. From the Bayesian perspective, Equation 2 can be understood as a special case of Bayes' rule Equation 3 with flat (uninformative) prior information. For a thorough introduction on Bayesian methods, we refer the interested reader to [18].

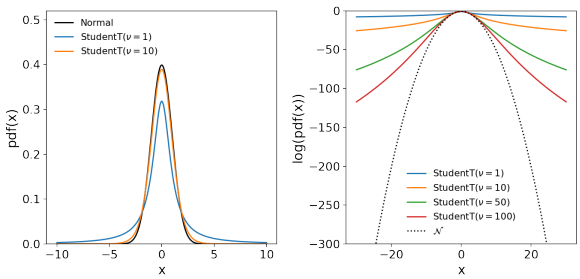

Figure 1: Comparison between Normal and Student-t distribution
In the left chart, the probability density function (PDF) of a Normal distribution, as well as two Student's-t distributions with varying degree of freedom (ν) are shown. Both distributions are parametrized by a location parameter μ that is equal to the mean and mode of these distributions. In addition to μ, the Normal is parametrized by its standard deviation parameter σ, influencing the spread of the distribution. In contrast, the Student's-t distribution has two spread parameters $\{scale, \nu\}$ and is characterized by more probability mass in the tails of the PDF, not approaching 0 as quickly as the PDF of the Normal. With increasing ν, the Student's-t distribution becomes more similar to the Normal distribution. The log probability density (right) of the Normal distribution accelerates has a quadratic dependency on the distance to the mean, whereas the log-PDF of the Student's-t distribution does not go to extreme values as quickly. Because of this property, the Student's distribution causes less numerical problems at extreme values.

2.2 Parameter estimation

A mathematical model ϕ is a function that describes the state of system variables by means of a set of parameters. The model is a representation of the underlying *data generating process*, meaning that the model output from a given set of parameters is imitating the expected output in the real system. From a known list of parameters θ, a model can make predictions of the system variables, in the following denominated as $\vec{y}_{\text{pred}}$. In Machine Learning, this quantity is often called $\hat{\vec{y}}$.

$$\vec{y}_{\text{pred}} = \phi(\vec{\theta}) \tag{4}$$

A predictive model can be obtained when the parameters are estimated from observed experimental data $\vec{y}_{\text{obs}}$. In this process, the experimental data is compared to data predicted by the model. In order to find the prediction matching the data best, different approaches of *parameter estimation* can be applied, sometimes also referred to as *inference* or

informally as *fitting*.
To obtain one parameter vector, optimization of so-called *loss functions* or *objective functions* can be applied. In principle, these functions compare prediction and measurement outcome, yielding a scalar that can be minimized. Various loss functions can be formulated for the optimization process.
In the following, we first consider a special case, least squares estimation, before coming to the generalized approach of maximum likelihood estimation (MLE). The former, which is often applied in biotechnology in the context of linear regression, is specified in the following equation.

$$L = (\vec{y}_{\text{obs}} - \vec{y}_{\text{pred}})^2 \tag{5}$$

Here, the vectors $\vec{y}_{\text{obs}}$ and $\vec{y}_{\text{pred}}$ represent one observed time series and the corresponding prediction. If several time series contribute to the optimization, their differences (residuals) can be summed up:

$$L = \sum_{n=0}^{N} (\vec{y}_{\text{obs, n}} - \vec{y}_{\text{pred, n}})^2 \tag{6}$$

To keep the notation simple, we will in the following use Y_{obs} and Y_{pred} to refer to the set of N time series vectors. However, the vectors might be of different length and thus Y should not be interpreted as a matrix notation. In later chapters, we will see how the Python implementation handles the sets of observations (Section 3.2.4).
Coming back to the likelihood functions introduced in the previous chapter, the residual-based loss functions are a special case of a broader estimation concept, the *maximum likelihood estimation* (MLE):

$$\vec{\theta}_{\text{MLE}} = \underset{\vec{\theta}}{\text{argmax}}\ \mathcal{L}(\vec{\theta} \mid Y_{\text{obs}}) \tag{7}$$

Here, a probability density function is used to quantify how well observation and prediction, the latter represented by the model parameters, match. In case of a Normal-distributed likelihood with constant noise, the result of MLE is the same as a weighted least-squares loss [19]. In comparison to residual-based approaches, the choice of the PDF in a likelihood approach leads to more flexibility, for example covering heteroscedasticity or measurement noise that cannot be described by a Normal distribution.
As introduced in Section 2.1, an important extension of the likelihood approach is Bayes' theorem (Equation 3). Applying this concept, we can perform *Bayesian inference* of model parameters:

$$p(\vec{\theta} \mid Y_{obs}) = \frac{p(\vec{\theta}) \cdot \mathcal{L}(\vec{\theta} \mid Y_{\text{obs}})}{\int p(\vec{\theta}) \cdot \mathcal{L}(\vec{\theta} \mid Y_{\text{obs}})} \tag{8}$$

$$\vec{\theta}_{\text{MAP}} = \underset{\vec{\theta}}{\text{argmax}}\ p(\vec{\theta} \mid Y_{obs}) \tag{9}$$

Similar to MLE, a point estimate of the parameter vector with highest probability can be obtained by optimization (Equation 9), resulting in the *maximum a posteriori* (MAP) estimate. While the MLE is focused on the data-based likelihood, MAP estimates incorporate prior knowledge $p(\vec{\theta})$ into the parameter estimation.
To obtain the full posterior distribution $p(\vec{\theta} \mid Y_{obs})$, which is describing the probability distribution of parameters given the observed data, one has to solve Equation 8. The integral, however, is often intractable or impossible to solve analytically. Therefore, a class of algorithms called *Markov chain Monte Carlo* (MCMC) algorithms is often applied to find numerical approximations for the posterior distribution (for more detail, see Section 3.2.6).
The possibility to not only obtain point estimates, but to obtain a whole distribution describing the parameter vector, is leading to an important concept: uncertainty quantification.

2.3 Uncertainty quantification of model parameters

When aiming for predictive models, it is important to not only estimate one parameter vector, but to quantify how certain the estimation is. In the frequentist paradigm, uncertainty is quantified with *confidence intervals*. When applied correctly, they provide a useful measure, for example in hypothesis testing where the size of a certain effect in a study is to be determined. However, interpretation of the confidence interval can be challenging and it is frequently misinterpreted as the interval that has a 95% chance to contain the true effect size or true mean [11]. However, to obtain intervals with such a simple interpretation, further assumptions on model parameters are required [12].

In Bayesian inference, prior distribution provide these necessary assumptions and the posterior can be used for uncertainty quantification. As a consequence, Bayesian *credible intervals* can indeed be interpreted as the range in which an unobserved parameter lies with a certain probability [13]. The choice of probability level (e.g. 90 %) or interval bounds is arbitrary. Consequently, there are many equally valid flavors of credible intervals. The most important ones are:

- **Highest posterior density** intervals (HDI) are chosen such that the width of the interval is minimized
- **Equal-tailed** intervals (ETI) are chosen such that the probability mass of the posterior below and above the interval are equal
- **Half-open** credible intervals specify the probability that the parameter lies on one side of a threshold

In the scope of this paper, we will solely focus on the Bayesian quantification of parameter uncertainty. Note that uncertainty of parameters should not be confused with the measurement uncertainty mentioned in the context of calibration in Section 1.2, which will be further explained in the following section.

2.4 Calibration models

Coming back to the BIPM definition of calibration (Section 1.1), we can now associate aspects of that definition with the statistical modeling terminology. In Figure 2 (left), the blue axis "independent" variable corresponds to the "quantity values" from the BIPM definition. At every value of the independent variable, the calibration model (green) describes the probability distribution (green slices) of measurement responses. This corresponds to the "indications with associated measurement uncertainties" from the BIPM definition.
Neither the formal definition, nor the conceptual framework presented in this study impose constraints on the kind of probability distribution that describes the measurement responses. Apart from the Normal distribution, a practitioner may choose a Student's-t distribution if outliers are a concern. The Student's-t distribution has a v parameter that influences how much probability is attributed to the tails of the distribution (Figure 1), or in other words how likely it is to observe extreme values. Depending on the measurement system at hand, a Lognormal, Gamma, Weibull, Uniform or other continuous distributions may be appropriate. Also discrete distributions such as the Poisson, Binomial or Categorical may be chosen to adequately represent the observation process.
For some distributions, including Normal and Student's-*t*, the parameters may be categorized as *location* parameters affecting the median or *spread* parameters affecting the variance, while for many other distributions the commonly used parameterization is less not as independent. The parameters of the probability distribution that models the measurement responses must be described as functions of the independent variable. In the example from Figure 2, a Student's-t distribution with parameters $\{\mu, \text{scale}, \nu\}$ is used. Its parameter μ is modeled with a logistic function, the scale parameter as a 1st order polynomial of μ and ν is kept constant. It should be emphasized that the choice of probability distribution and functions to model its parameters is completely up to the domain expert.

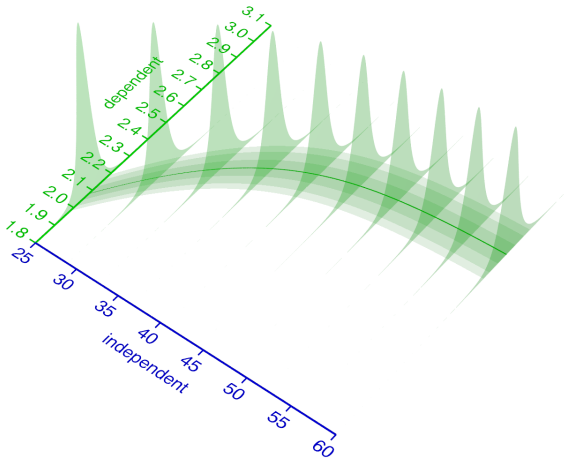

Figure 2: Relationship of independent and dependent variable
The distribution of measurement responses (dependent variable) can be modeled as a function of the independent variable. This measurement response probability distribution (here: Student's t) is parametrized by its parameters the mean μ (solid green line) and spread parameters σ and ν. Some or all of the distributions parameters are modeled as a function of the independent variable.

When coming up with the structure of a calibration model, domain knowledge about the measurement system should be considered, particularly for the choice of probability distribution. An exploratory scatter plot can help to select an adequate function for the location parameter of the distribution (μ in case of a Normal or Student's-t). A popular choice for measurement systems that exhibit saturation kinetics is the (asymmetric) logistic function. Many other measurement systems can be operated in a "linear range", hence a 1st order polynomial is an equally popular model for the location parameter of a distribution. To describe the spread parameters ($\sigma, scale, \nu, \ldots$), a 0th (constant) or 1st order (linear) polynomial function of the location parameter is often a reasonable choice.
After specifying the functions in the calibration model, the practitioner must fit the model (Section 2.2) and decide to stick with the result, or modify the functions in the model. This iteration between model specification and inspection, is a central aspect of modeling. A general recommendation is to find the simplest model that is in line with domain knowledge about the measurement system, while minimizing the *lack-of-fit*.
The term *lack-of-fit* is used to describe systematic deviation between the model fit and data. It refers not only to the trend of location and spread parameters, but also to the kind of probability distribution. A *residual plot* is often instrumental to diagnose lack-of-fit and discriminate it from purely random noise in the observations. In Figure 3, different calibration models (top), residuals (middle) and the spread of data points along the predicted probability distribution (bottom) illustrate how to diagnose a lack-of-fit. A well-chosen model (D) is characterized by the random spread of residuals without systematic deviation and the equivalence of the modeled and observed distribution. When enough calibration data points are available, the modeled and observed distributions can be compared via the occupancy of percentiles.

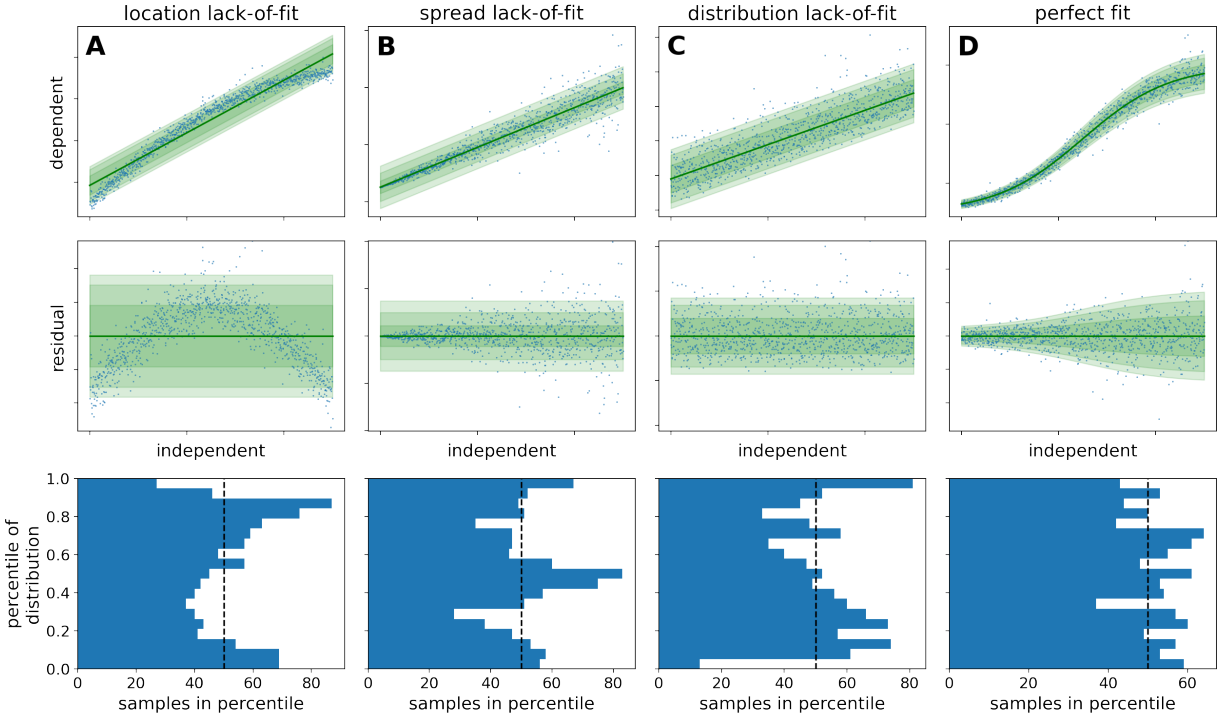

Figure 3: Diagnostic plots of model fits
Linear and logistic models were fitted to synthetic data to show three kinds of *lack-of-fit* error (columns 1-3) in comparison to a perfect fit (column 4). The underlying structure of the data and model is as follows: **A:** Homoscedastic linear model, fitted to homoscedastic nonlinear data **B:** Homoscedastic linear model, fitted to heteroscedastic linear data **C:** Homoscedastic linear model, fitted to homoscedastic linear data that is Lognormal-distributed **D:** Heteroscedastic logistic model, fitted to heteroscedastic logistic data The raw data (blue dots) and corresponding fit is visualized in the top row alongside a density band that corresponds to the regions of highest likelihood according to the model. The residual plots in the middle row show the distance between the data and the modeled location parameter (green line). The bottom row shows how many data points fall into the percentiles of the predicted probability distribution. Whereas the lack-of-fit cases exhibit systematic under- and over-occupancy of percentiles, only in the perfect fit case all percentiles are approximately equally occupied.

Whereas the BIPM definition uses the word *uncertainty* in multiple contexts, we prefer to always use the term to describe *uncertainty in a parameter*, but never to refer to measurement noise. In other words, the parameter uncertainty

can often be reduced by acquiring more data, whereas measurement noise is inherent and constant. In the context of calibration models, the methods for uncertainty quantification (Section 2.3) may be applied to the calibration model parameters, the independent variable, or both. Uncertainty quantification of calibration model parameters can be useful when investigating the structure of the calibration model itself, or when optimization does not yield a reliable fit. Because the independent variable is in most cases the parameter of interest in the application of a calibration model, the quantification of uncertainty about the independent variable is typically the goal. To keep the examples easy and understandable, we fix calibration model parameters at their maximum likelihood estimate; however, we would like to point out that `calibr8` does not make this restriction.

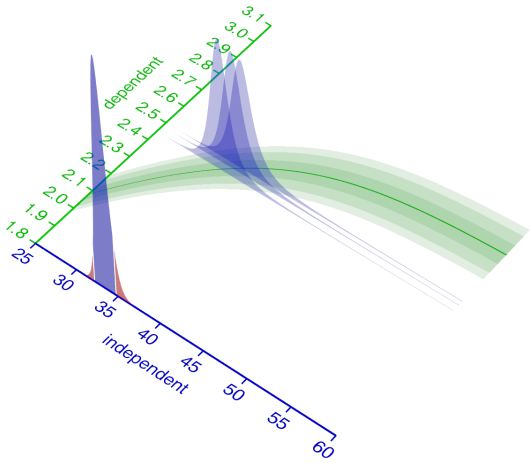

Figure 4: Uncertainty about the independent variable
An intuition for inferring the independent variable from an observed dependent variable is to cut (condition) the green probability distribution model at the observed value (blue slices) and normalize its area to 1. The resulting (blue) slice is a potentially asymmetric probability distribution that describes the likelihood of the observation, given the independent variable. Its maximum (the maximum likelihood estimate) is the value of the independent variable that best describes the observation. For multiple observations, the probability density function for the independent variable corresponds to the product of the PDFs of the observations. The red shoulders mark the regions outside of the 90 % equal-tailed interval.

In Figure 4, the green likelihood bands on the ground of the 3D plot represent a calibration model with fixed parameters. To quantify the independent variable with associated Bayesian uncertainty, it must be considered as a random variable. Accordingly, $p(\mathrm{rv}_{\mathrm{independent}} \mid \mathrm{d})$ from either a likelihoodist (Equation 2) or Bayesian (Equation 3) perspective is the desired outcome of the uncertainty quantification.
Given a single observed dependent variable, the likelihoodist $p(\mathrm{rv}_{\mathrm{independent}} \mid d)$ (Equation 2) corresponds to the normalized cross-section of the likelihood bands at the observed dependent variable (Figure 4, blue slices). With multiple observations, $p(\mathrm{rv}_{\mathrm{independent}} \mid \mathrm{d})$ becomes the product (superposition) of the elementwise likelihoods (Figure 4, blue slice at the axis). For a Bayesian interpretation of $p(\mathrm{rv}_{\mathrm{independent}} \mid \mathrm{d})$ (Equation 3), the blue likelihood slice is superimposed with an additional prior distribution (not shown). More practical details on uncertainty quantification of the independent variable in a calibration model are given in Section 4.

2.5 Process models

Most research questions are not answered by inferring a single variable from some observations. Instead, typical questions target the comparison between multiple conditions, the value of an *latent* (unobservable) parameter, or the inter- and extrapolation of a temporally evolving system. For example, one might extract a latent parameter that constitutes a key performance indicator, or make decisions based on predictions (extrapolation) of new scenarios. Data analysis for all of these and many more scenarios is carried out with models that are tailored to the system or process under investigation. Such models are typically derived from theoretical (textbook) understanding of the process under investigation and in terms of SI units, but are not concerned with the means of making observations. Henceforth, we use the term *process model* (ϕ_{textpm}) to describe such models and discriminate them from calibration models (ϕ_{textcm}) that are explicitly concerned with the observation procedure.
Whereas calibration models are merely univariate input/output relationships of a measurement system, process models may involve many parameters, hierarchy, multivariate predictions or more complicated functions such as ordinary or partial differential equations (ODEs, PDEs). For example, they may predict a temporal evolution of a system

with differential equations, sharing some parameters between different conditions, while keeping others local. In life-sciences, time series play a central role, hence our application example is also concerned with a temporally evolving system.
Nevertheless, calibration models ϕ_{textcm} and process models ϕ_{textpm} are models, and the methods for estimation of their parameters (Section 2.2) as well as uncertainty quantification (Section 2.3) apply to both. As described in Section 2.3, the likelihood $\mathcal{L}$ is the ultimate all-rounder tool in parameter estimation. The key behind our proposed discrimination between calibration and process models is the observation that a calibration model can serve as a modular likelihood function for a process model (Equation 10).

$$\begin{aligned} Y_{\text{pred}} &= \phi_{\text{pm}}(\vec{\theta}_{\text{pm}}) \\ \mathcal{L}(\vec{\theta}_{\text{pm}} \mid Y_{\text{obs}}) &= \mathcal{L}(Y_{\text{pred}} \mid Y_{\text{obs}}) \\ \mathcal{L}(Y_{\text{pred}} \mid Y_{\text{obs}}) &\propto p(Y_{\text{obs}} \mid \phi_{\text{cm}}(Y_{\text{pred}}, \vec{\theta}_{\text{cm}})) \end{aligned} \tag{10}$$

Conceptually separating between calibration models and process models has many advantages for the data analysis workflow in general. After going into more detail about the implementation of calibration models and process models in Section 3, we will demonstrate their application and combination in Section 4.

3 Material and methods

3.1 Experimental workflows

3.1.1 Automated microbioreactor platform

All experiments were conducted on a so-called automated microbioreactor platform. In our setup, a BioLector Pro microbioreactor system (m2p-labs GmbH, Baesweiler, Germany), is integrated into a Tecan Freedom EVO liquid handling robot (Tecan, Männedorf, Switzerland). The BioLector pro is a device to quasi-continuously observe biomass, pH and dissolved oxygen (DO) during cultivation of microorganisms in specialized microtiter plates (MTPs). These rectangular plates comprise multiple reaction cavities called "wells", usually with volumes in microliter or milliliter scale. The BioLector allows to control temperature and humidity while shaking the plates at adjustable frequencies between 800 and 1500 rpm.
The liquid handler, which allows to take samples for *at-line* measurements during cultivation, is surrounded by a laminar flow hood to ensure sterile conditions for liquid transfer operations. Next to the BioLector Pro, various other devices are available on the platform, including an Infinite® M Nano+ microplate photometer (Tecan, Männedorf, Switzerland), a cooling carrier and a Hettich Rotanta 460 robotic centrifuge (Andreas Hettich GmbH & Co. KG, Tuttlingen, Germany). The overall setup is similar to the one described by Unthan *et al.* 2015 [8]. The automated platform enables to perform growth experiments with different microorganisms, to autonomously take samples of the running process and to perform bioanalytical measurements, *e.g.* quantification of glucose. It is thus a device for miniaturised, automated bioprocess cultivation experiments.
In this work, we used pre-sterilized, disposable 48-well FlowerPlates® (MTP-48-B, m2p-labs GmbH, Baesweiler, Germany) covered with a gas-permeable sealing film with a pierced silicone layer for automation (m2p-labs GmbH, Baesweiler, Germany). The biomass was quasi-continuously detected via scattered light [20] at gain 3 with 4 minutes cycle time to obtain backscatter measurements. DO and pH were not measured since they are not relevant for the application examples. Both cultivation and biomass calibration experiments were conducted in the BioLector Pro at 30 °C, 3 mm shaking diameter, 1400 rpm shaking frequency, 21% headspace oxygen concentration and $\geq$ 85% relative humidity.

3.1.2 Strain, media preparation and cell banking and cultivation

The wild-type strain *Corynebacterium glutamicum* ATCC 13032 [21] was used in this study. If not stated otherwise, all chemicals were purchased from Sigma–Aldrich (Steinheim, Germany), Roche (Grenzach-Wyhlen, Germany) or Carl Roth (Karlsruhe, Germany) in analytical quality.
Cultivations were performed with CGXII defined medium with the following final amounts per liter of distilled water: 20 g D-glucose, 20 g $(NH_4)_2SO_4$, 1 g K_2HPO_4, 1 g KH_2PO_4, 5 g urea, 13.25 mg $CaCl_2 \cdot 2\,H_2O$, 0.25 g $MgSO_4 \cdot 7\,H_2O$, 10 mg $FeSO_4 \cdot 7\,H_2O$, 10 mg $MnSO_4 \cdot H_2O$, 0.02 mg $NiCl_2 \cdot 6\,H_2O$, 0.313 mg $CuSO_4 \cdot 5\,H_2O$, 1 mg $ZnSO_4 \cdot 7\,H_2O$, 0.2 mg biotin, 30 mg protocatechuic acid. 42 g/L MOPS were used as buffering agent and the pH was adjusted to 7.0 using 4 M NaOH.
A working cell bank (WCB) was prepared from a shake flask culture containing 50 mL of the described CGXII medium

and 10 % (v/v) brain heart infusion (BHI) medium (37 g/L). It was inoculated with 100 µl cryo culture from a master cell bank stored at -80°C. The culture was incubated for approximately 16 hours in an unbaffled shake flask with 500 ml nominal volume at 250 rpm, 25 mm shaking diameter and 30 °C. The culture broth was then centrifuged at $4000 \times g$ for 10 minutes at 4 °C and washed once in 0.9% sterile NaCl solution. After centrifugation, the pellets were resuspended in a suitable volume of NaCl solution to yield a suspension with an optical density at 600 nm (OD_{600}) of 60. The suspension was then mixed with an equal volume of 50% (w/v) sterile glycerol, resulting in cryo cultures of $OD_{600} \approx 30$. Aliquots of 1 mL were quickly transferred to low-temperature freezer vials, frozen in liquid nitrogen and stored at -80°C.

3.1.3 Algorithmic planning of dilution series

All calibration experiments require a set of *standards* (reference samples) with known concentrations, spanning across sometimes multiple orders of magnitude. Traditionally such standards are prepared by manually pipetting a serial dilution with a 2x dilution factor in each step. This can result in a series of standards whose concentrations are evenly spaced on a logarithmic scale. While easily planned, a serial dilution generally introduces inaccuracies that accumulate with an increasing number of dilution steps. It is therefore desirable to plan a dilution series of reference standards such that the number of serial dilution steps is minimized.

To reduce the planning effort and allow for a swift and accurate preparation of the standards, we devised an algorithm that plans liquid handling instructions for preparation of standards. Our `DilutionPlan` algorithm considers constraints of a $(R \times C)$ grid geometry, well volume, minimum and maximum transfer volumes to generate pipetting instructions for human or robotic liquid handlers.

First, the algorithms reshapes a length $R \cdot C$ vector of sorted target concentrations into the user specified $(R \times C)$ grid typically corresponding to a microtiter plate. Next, it iteratively plans the transfer- and filling volumes of grid columns subject to the volume constraints. This column-wise procedure improves the compatibility with multi-channel manual pipettes, or robotic liquid handlers. Diluting from a stock solution is prioritized over the (serial) dilution from already diluted columns. The result of the algorithm are (machine readable) pipetting instructions to create $R \cdot C$ single replicates with concentrations very close to the targets. We open-sourced the implementation as part of the `robotools` library [22].

As the accuracy of the calibration model parameter estimate increases with the number of calibration points, we performed all calibrations with the maximum number of observations that the respective measurement system can make in parallel. The calibration with 96 glucose and 48 biomass concentrations is covered in the following chapters.

3.1.4 Glucose assay calibration

For the quantification of `D`-glucose, the commercial enzymatic assay kit "Glucose Hexokinase FS" (DiaSys Diagnostic Systems, Holzheim, Germany) was used. For the master mix, four parts buffer and one part enzyme solution were mixed manually. The master mix was freshly prepared for each experiment and incubated at room temperature for at least 30 minutes prior to the application for temperature stabilization. All other pipetting operations were performed with the robotic liquid handler. For the assay, 280 µL master mix were added to 20 µL analyte in transparent 96-well flat bottom polystyrol plates (Greiner Bio-One GmbH, Frickenhausen, Germany) and incubated for 6 minutes, followed by absorbance measurement at 365 nm. To treat standards and cultivation samples equally, both were diluted by a factor of 10 (100 µL sample/standard + 900 µL diluent) as part of the assay procedure.

As standards for calibration, 96 solutions with concentrations between 0.075 and 50 g/L were prepared from fresh CGXII cultivation medium (Section 3.1.2) with a 50 g/L concentration of `D`-glucose. The `DilutionPlan` algorithm (Section 3.1.3) was used to plan the serial dilution procedure with glucose-free CGXII media as the diluent, resulting in 96 unique concentrations, evenly distributed on a logarithmic scale. Absorbance results from the photometer were parsed with a custom Python package and paired with the concentrations from the serial dilution series to form the calibration dataset used in Section 4.1.2. 83 of the 96 concentration/absorbance pairs lie below 20 g/L and were used to fit a linear model in Section 4.1.1.

3.1.5 Biomass calibration

Calibration data for the biomass/backscatter calibration model Figure 9 was acquired by measuring 48 different biomass concentrations at cultivation conditions (Section 3.1.2) in the BioLector Pro. 100 mL *C. glutamicum* WT culture was grown overnight on 20 g/L glucose CGXII medium (Section 3.1.2) in two unbaffled 500 mL shake flasks with 50 mL culture volume each (N=250 rpm, r=25 mm). The cultures were harvested in the stationary phase, pooled, centrifuged and resuspended in 25 mL 0.9 $\%_{w/v}$ NaCl solution. The highly concentrated biomass suspension was transferred into a magnetically stirred 100 mL trough on the liquid handler, for automated serial dilution with logarithmically evenly spaced dilution factors from $1\times$ to $1000\times$. The serial dilution was prepared by the robotic liquid handler in a 6×8 (48-well square) deep well plate (Agilent Part number 201306-100) according to the `DilutionPlan` (Section 3.1.3).

6x 800 µL of biomass stock solution were transferred to previously dried and weighed 2 mL tubes, immediately after all transfers of stock solution to the 48 well plate had occurred. The 2 mL tubes were frozen at -80 °C, lyophilized over night, dried again at room temperature in a desiccator over night and finally weighted again to determine the biomass concentration in the stock solution.
After a column in the 48 well plate was diluted with 0.9 $\%_{\mathrm{w/v}}$ NaCL solution, the column was mixed twice by aspirating 950 µL at the bottom of the wells and dispensing above the liquid surface. The transfers for serial dilutions (columns 1 and 2) and to the 48 well FlowerPlate were executed right after mixing to minimize the effects of biomass sedimentation as much as possible. The FlowerPlate was sealed with a gas-permeable sealing foil (product number F-GP-10, m2p-labs GmbH, Baesweiler, Germany) and placed in the BioLector Pro device. The 1 h BioLector process for the acquisition of calibration data was programmed with shaking frequency profile of 1400, 1200, 1500, 1000 rpm while maintaining 30 °C chamber temperature and measuring backscatter with gain 3 in cycles of 3 minutes.
The result file was parsed with a custom Python package and backscatter measurements made at 1400 rpm shaking frequency were extracted. A log(independent) asymmetric logistic calibration model was fitted as described in Section 4.1.2. The linear calibration model for comparison purposes (Figure 14) was implemented with its intercept fixed to the background signal predicted by the asymmetric logistic model ($\mu_{\mathrm{BS}}(0\ \frac{g}{L})$). It was fitted to a subset of calibration points approximately linearly spaced at 30 different biomass concentrations from 0.01 to 15 g/L.

3.1.6 Microbial growth experiment

Cultivations with *C. glutamicum* were performed in the automated microbioreactor platform (Section 3.1.1) under the described conditions. CGXII medium with 20 g/L glucose and without BHI was used as cultivation medium. To start the growth experiments, the liquid handler was used to transfer 20 µL of a WCB aliquot into the first column of FlowerPlate wells, which were pre-filled with 780 µL medium. These wells were run as a preculture. When precultures reached a backscatter readout of 15, which corresponds to a cell dry weight of approximately 10 g/L, the inoculation of the main culture wells was triggered. 780 µL medium were distributed into each main culture well (columns 2-8) and allowed to warm up for approximately 15 minutes. Preculture wells A01 and B01 were pooled and 20 µL culture broth was transferred to each main culture well, resulting in 800 µL final volume. The theoretical biomass concentration at the start of the main cultures is 0.25 g/L accordingly. This strategy was used to avoid a lag-phase with non-exponential growth.
Backscatter measurements of biomass concentration were acquired continuously, while main culture wells were harvested at predetermined time points to measure glucose concentrations in the supernatant. The time points were chosen between 0 and 15 hours after the inoculation of main cultures to cover all growth phases. For harvesting, the culture broth was transferred to a 1 mL deep-well plate by the liquid handler. The plate was centrifuged at $3190 \times g$ at 4 °C for 5 minutes and the supernatant was stored on a 1 mL deep well plate chilled to 4 °C. The glucose assay was performed after all samples were taken.

3.2 Computational methods

All analyses presented in this study were performed with recent versions of Python 3.7, PyMC3 ==3.11.2 [23], ArviZ >=0.9 [24], PyGMO >=2.13 [25], matplotlib >=3.1 [26], NumPy >=1.16 [27], pandas >=0.24 [28, 29], SciPy >=1.3 [30] and related packages from the Python ecosystem. For a full list of dependencies and exact versions see the accompanying GitHub repository and supporting information.

The two packages presented in this study, `calibr8` and `murefi` , may be installed via semantically versioned releases on PyPI. Source code, documentation and detailed release notes are available through their respective GitHub projects [31, 32].

3.2.1 Asymmetric logistic function

The asymmetric, five-parameter logistic function (also known as *Richard's curve*) was previously shown to be a good model for many applications [33], but it is often defined in a parameterization (Equation 11) that is non-intuitive. Some parametrizations even introduce a sixth parameter to make the specification more intuitive, but this comes at the cost of structural non-identifiability [34, 35]. Furthermore, in the most commonly found parametrization (Equation 11), one parameter is constrained to be strictly positive. We also found that structural non-identifiability between the parameters makes it difficult to define an initial guess and bounds to reliably optimize a model based on this parametrization.

$$
\begin{aligned}
f(x) &= L_L + \frac{L_U - L_L}{(1 + e^{-B(m-x)})^{1/v}} \\
L_L, L_U, B, m &\in \mathbb{R} \\
v &\in \mathbb{R}_{>0}
\end{aligned}
\tag{11}
$$

To make calibration model fitting more user friendly, we reparameterized the commonly used form such that all five parameters are intuitively interpretable and structurally independent Figure 5. With our reparameterization (Equation 12), the 5-parameter asymmetric logistic function is parameterized by lower limit $L_L \in \mathbb{R}$, upper limit $L_U \in \mathbb{R}$, inflection point x-coordinate $I_x \in \mathbb{R}$, slope at inflection point $S \in \mathbb{R}$ and an asymmetry parameter $c \in \mathbb{R}$. At $c = 0$, the y-coordinate of the inflection point lies centered between L_L and L_U. I_y moves closer to L_U when $c > 0$ and accordingly closer to L_L when $c < 0$ (Figure 5, black and gray curves). An interactive version of Figure 5 can be found in a Jupyter notebook in the `calibr8` GitHub repository ([31]).

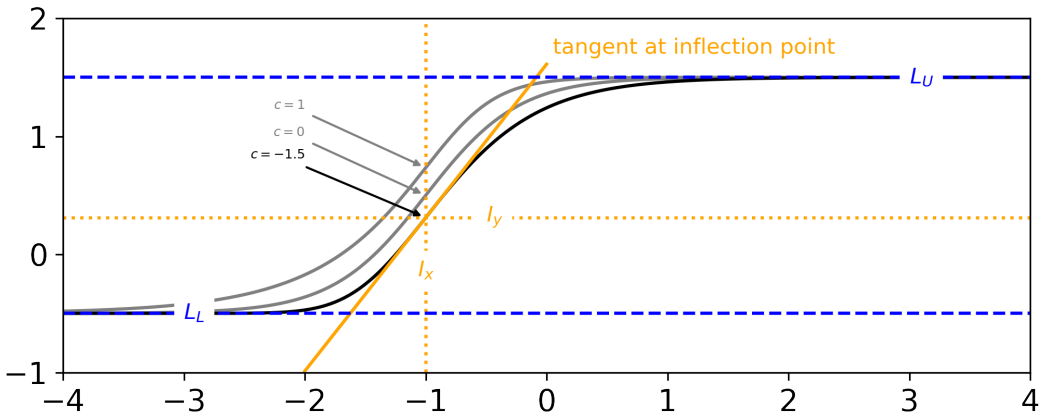

Figure 5: Reparametrized Asymmetric Logistic Function
When parametrized as shown in Equation 12, each of the 5 parameters can be manipulated without influencing the others. Note that, for example, the symmetry parameter c can be changed without affecting the x-coordinate of the inflection point (I_x), or the slope S at the inflection point (gray vs. black).

For readability and computational efficiency, we used `SymPy` [36] to apply common subexpression elimination to Equation 12 and our implementation respectively (Code 6). The step wise derivation from Equation 11 to Equation 12 is shown in Appendix A.1 and in a Jupyter notebook in the `calibr8` GitHub repository ([31]).

$$
\begin{aligned}
f(x) &= L_L + \frac{L_U - L_L}{(e^{s_2 \cdot (s_3 \cdot (I_x - x) + \frac{c}{s_2})} + 1)^{s_1}} \\
s_0 &= e^c + 1 \\
s_1 &= e^{-c} \\
s_2 &= s_0^{(s_0 \cdot s_1)} \\
s_3 &= \frac{S}{L_U - L_L} \\
L_L, L_U, I_x, S, c &\in \mathbb{R}
\end{aligned}
\tag{12}
$$

3.2.2 `calibr8` package for calibration models and modular likelihoods

With `calibr8` we present a lightweight Python package that specializes on the definition and modular implementation of non-linear calibration models for calibration and modeling workflows.
The `calibr8` application programming interface (API) was designed such that all calibration models are implemented as classes that inherit from `calibr8.CalibrationModel`, which implements properties and methods that are common

to all calibration models (Figure 6). The common interface simplifies working with calibration models in a data analysis or modeling workflow. For example, the `CalibrationModel.objective` can be used to create objective functions to optimize the model parameters. The objective relies on the `loglikelihood` method to compute the sum of log-likelihoods from independent and dependent variables. It uses the `predict_dependent` method internally to obtain the parameters of the probability distribution describing the dependent variables, conditioned on the independent variable.

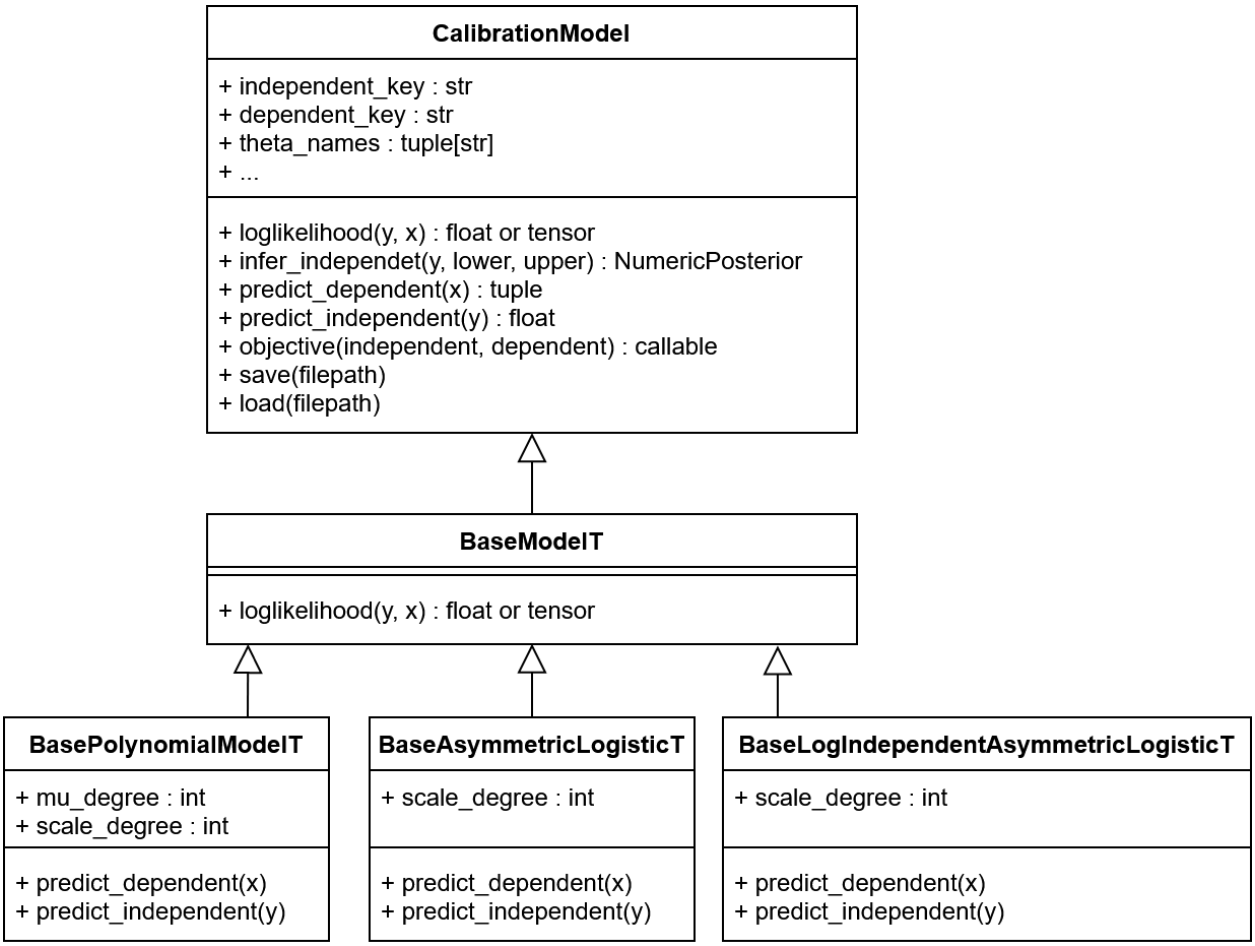

Figure 6: `calibr8` Class Diagram
All `calibr8` models inherit from the same `CalibrationModel` class that defines attributes, properties and method signatures that are common to all calibration models. Some methods, like `save()` or `objective()` are implemented by `CalibrationModel` directly, whereas others are implemented by the inheriting classes. Specifically the `loglikelihood` and the `predict_*` methods depend on the choice of the domain expert. With a suite of `Base*T` classes, `calibr8` provides base classes for models based on Students-*t* distributed observations. A domain expert may start from any of these levels to implement a custom calibration model for a specific application.

Through its inheritance-based design, the `calibr8.CalibrationModel` gives the domain expert full control over the choice of trend functions and probability distributions. Conveniently, `calibr8` already implements functions such as `polynomial`, `logistic` and `asymmetric_logistic`, as well as base classes for commonly found models. By leveraging these base models, the implementation of a user-defined calibration model reduces to just a few lines of code (Code 1 and Code 2).
The implementations depicted in Figure 6 are fully compatible with `aesara.Variable` inputs, resulting in `TensorVariable` outputs. Aesara is a graph computation framework that auto-differentiates computation graphs written in Python and compiles functions that evaluate with high performance [37]. This way, the `loglikelihood` function of a `CalibrationModel` can be auto-differentiated and compiled, to facilitate efficient computation with optimization or gradient-based MCMC sampling algorithms (Section 3.2.6). For more details about the implementation, please refer to the documentation and code of the `calibr8` package ([31]).

Convenience features To facilitate modeling workflows, `calibr8` implements convenience functions for optimization (`fit_scipy`, `fit_pygmo`) and creation of diagnostic plots (`calibr8.plot_model`) as shown in Figure 8 and Figure 9. As explained in Section 2.4 the residual plot on the right of the resulting figure is instrumental to judge the quality of the model fit.
Standard properties of the model, estimated parameters and calibration data can be saved to a JSON file via the `CalibrationModel.save` method. The saved file includes additional information about the type of calibration model and the `calibr8` version number (e.g. Code 9) to support good versioning and data provenance practices. When the `CalibrationModel.load` method is called to instantiate a calibration model from a file, standard properties of the new instance are set and the model type and `calibr8` version number are checked for compatibility.

3.2.3 Numerical inference

To numerically infer the posterior distribution of the independent variable, given one or more observations, `infer_independent` implements a multi-step procedure. The three outputs of this procedure are **A** a vector of posterior probability evaluations, densely resolved around the locations of high probability mass, and **B** the bounds of the equal-tailed, and highest-density intervals (ETI, HDI) corresponding to a user specified credible interval probability. In the first step, the likelihood function is integrated in the user specified interval $[\text{lower}, \text{upper}]$ with `scipy.integrate.quad`. Second, we evaluate its cumulative density function (CDF) at 10 000 locations in $[\text{lower}, \text{upper}]$ and determine locations closest to the $\text{ETI}^{99.999\ \%}$. Next, we re-evaluate the CDF at 100 000 locations in the $\text{ETI}^{99.999\ \%}$ to obtain it with sufficiently high resolution in the region of highest probability. Both ETI and HDI with the (very close to) user specified `ci_prob` are obtained from the high resolution CDF. Whereas the ETI is easily obtained by finding the CDF evaluations closest to the corresponding lower and upper probability levels, the HDI must be determined through optimization (Equation 13).

$$\text{HDI} = [a, a+d] = \underset{a,d}{\operatorname{argmin}} \begin{cases} \infty & \text{if } \text{CDF}(a+d) - \text{CDF}(a) < \texttt{ci_prob} \\ d & \text{otherwise} \end{cases} \tag{13}$$

3.2.4 `murefi` package for building multi-replicate ODE models

Models of biochemical processes are traditionally set up to describe the temporal progression of an individual system, such as a reaction vessel. Experimental data, however, is commonly obtained from multiple reaction vessels in parallel, often run under different conditions to maximize information gain. This discrepancy between the prototypical model of the biological system and the heterogeneous experimental data to be fitted is typically resolved by replicating the model for all realizations of the biological system in the dataset. Along with the replication of the model, some model parameters may be kept *global*, while others can be *local* to a subset of the replicates, for example due to batch effects or different start conditions.
With a Python package we call `murefi` (multi-replicate fitting), we implemented data structures, classes and auxiliary functions that simplify the implementation of models for such heterogeneous time series datasets. It seamlessly integrates with `calibr8` to construct likelihood-based objective functions for optimization or Bayesian inference. To enable the application of efficient optimization or sampling algorithms, the use of automatic differentiation to obtain gradients of the likelihood w.r.t. input parameters is highly desirable. Various methods for automatic differentiation of ODE models are available, but their efficiency is closely connected to the implementation and size of the model [38]. In `murefi` we implemented support for `sunode` [39], a recently implemented Python wrapper around the SUNDIALS suite of nonlinear and differential/algebraic equation solvers [40]. When used in the context of a PyMC3 model, a process model created with `calibr8` and `murefi` can therefore be auto-differentiated, solved, optimized and MCMC-sampled with particularly high computational efficiency.

Structure of time series data and models To accommodate for the heterogeneous structure of time series experiments in biological applications, we implemented a data structure of three hierarchy levels. The `murefi.Timeseries` object represents the time and state vectors $\vec{t}$, $\vec{y}$ of a single state variable or observation time series. To allow association of state and observed variables via `calibr8` calibration models, the `Timeseries` is initialized with `independent_key` and `dependent_key`. Multiple `Timeseries` are bundled to a `murefi.Replicate`, which represents either the observations obtained from one reaction vessel, or the predictions made by a process model. Consequently, the `murefi.Dataset` aggregates replicates of multiple reaction vessels, or the model predictions made for them (Figure 7 center). To allow for a separation of data preprocessing and model fitting in both time and implementation, a `murefi.Dataset` can be saved as and loaded from a single HDF5 file [41, 42].
To describe a reaction system by a system of ordinary differential equations, a new class is implemented by subclassing the `murefi.BaseODEModel` convenience type. In the constructor of the class, the names and order of parameters and

state variables are defined, whereas the differential equations are implemented in a `dydt` class method. An example is shown in Code 3 with the implementation of the Monod kinetics for microbial growth.

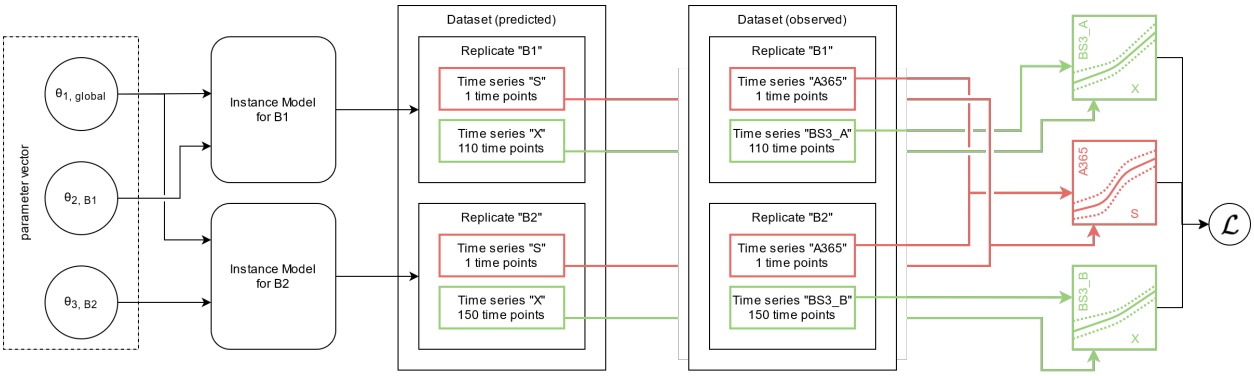

Figure 7: Data structures and computation graph of `murefi` models
Elements in a comprehensive parameter vector are mapped to replicate-wise model instances. In the depicted example, the model instances for both replicates "B1" and "B2" share $\theta_{1,global}$ as the first element in their parameter vectors. The second model parameter θ_2 is local to the replicates, hence the full parameter vector (left) is comprised of three elements. Model predictions are made such that they resemble the structure of the observed data, having the same number of time points for each predicted time series. An objective function calculating the sum of log-likelihoods is created by associating predicted and observed time series via their respective calibration models. By associating the calibration models based on the dependent variable name, a calibration model may be re-used across multiple replicates, or kept local if, for example, the observations were obtained by different methods.

Parameter mapping and objective function In addition to a `murefi` model instance, a `murefi.Dataset` and calibration models, a `murefi.ParameterMapping` must be defined to facilitate the creation of an objective function. This mapping specifies whether parameters are local or global and the rules with which they are shared between replicates. The `ParameterMapping` may be represented as a table, assigning each element of replicate-wise parameter vectors to constants or names of parameters in a comprehensive parameter vector. In Figure 7, the parameter mapping is depicted by arrows mapping elements of a 3-element comprehensive parameter vector to 2-element parameter vector of the replicate-wise models. A table-representation of the parameter mapping used to fit the Monod model in Section 4.2 is shown in Table 2.
Model predictions are made such that the time points of the predicted time series match those of the observed data (Figure 7, center). Based on the `(in)dependent_key`, the predicted and observed `Timeseries` can be associated with each other and passed to the corresponding `CalibrationModel.loglikelihood` method to calculate $\mathcal{L}(\vec{\theta} \mid Y_{\text{obs}})$. Note that this procedure conveniently allows for calibration models to be shared by multiple replicates, as well as making observations of one state variable with more than one analytical method.
An objective function performing the computation depicted in Figure 7 can be created with a single call to a convenience function. For compute-efficient optimization and specifically Bayesian parameter estimation, the elements in the parameter vector can also be *Aesara* tensors, resulting in the creation of a symbolic computation graph. The computation graph can not only be statically compiled, but also auto-differentiated, if all operations in the graph are also auto-differentiable. This is already the case for standard `calibr8` calibration models and is also available for `murefi` -based process models when the `sunode` [39] package is installed.

3.2.5 Optimization

In this work optimization algorithms are involved at multiple steps of the workflow. Unless otherwise noted we used *scipy.optimize.minimize* with default settings to obtain the MLEs of calibration and process models. Our implementation to compute HDIs (Section 3.2.3) uses `scipy.optimize.fmin`, as we found that the convergence with the underlying Nelder-Mead simplex algorithm was more reliable than with gradient-based optimizers from *scipy.optimize.minimize*. Initial guesses, as well as parameter bounds for maximum-likelihood optimization, were motivated from prior assumptions or exploratory plots of the data. Based on the intuitive parametrization of the asymmetric logistic (Section 3.2.1) we specified initial guesses for calibration models such that the model prediction from the guessed parameter vector was at least in the same order of magnitude as the data. For maximum likelihood estimation of process model parameters, the guessed parameters were motivated from prior assumptions. Likewise, we specified bounds to be realistic both biologically and based on exploratory scatter plots of the data.

3.2.6 MCMC sampling

In contrast to optimization, MCMC sampling follows a very different paradigm. Whereas in maximum likelihood estimation the likelihood function is iteratively evaluated to find its maximum, Bayesian inference aims to approximate the posterior probability distribution according to Equation 8.
Most *sampling algorithms* draw the posterior samples in the form of a Markov chain with a *equilibrium distribution* that matches the posterior probability distribution. While early MCMC algorithms, such as Random-walk Metropolis [43] are conceptually simple and easy to implement, they are computationally ineffective on problems with more than just a handful of dimensions [44, 45]. Instead of implementing inefficient algorithms by hand, practitioners can rely on state of the art libraries for Bayesian inference. These libraries apply automatic transformations, provide diagnostic tools and implement much more efficient sampling algorithms that often use gradients $\frac{d\mathcal{L}}{d\theta}$ for higher computational efficiency.
Probabilistic programming languages/libraries (PPL), such as PyMC3 [46], Pyro [47], Stan [48] or Tensorflow Probability [49] use automatic differentiation and typically implement at least one of the gradient-based sampling algorithms Hamiltonian Monte Carlo (HMC) or No-U-Turn Sampling (NUTS) [45]. PyMC3, the most popular Python-based PPL, implements both gradient-based (HMC, NUTS) as well as gradient-free algorithms, such as Differential Evolution MCMC (DE-MCMC) [50], DE-MCMC-Z [44] or elliptical slice sampling [51] in Python, allowing easy integration with custom data processing and modeling code. In this study, PyMC3 was used to sample posterior distributions with either DE-MCMC-Z (`pymc3.DEMetropolisZ`) or NUTS.

MCMC sampling of the process model Whereas in DE-MCMC, proposals are informed from a random pair of other chains in a "population", the DE-MCMC-Z version selects a pair of states from its own history, the "Z"-dimension. Compared to DE-MCMC, DE-MCMC-Z yields good results with less chains that can run independently. The `pymc3.DEMetropolisZ` sampler differs from the original DE-MCMC-Z in a tuning mechanism by which a `tune_drop_fraction` of by default 90% of the samples are discarded at the end of the tuning phase. This trick reliably cuts away unconverged "burn-in" history, leading to faster convergence.
`pymc3.DEMetropolisZ` was applied to sample the process model in Section 4.2.3. MCMC chains were initialized at the MAP to accelerate convergence of the DE-MCMC-Z sampler in the tuning phase. 50 000 tuning iterations per chain were followed by 500 000 iterations to draw posterior samples for further analysis. The `DEMetropolisZ` settings remained fixed at ($\lambda = \frac{2.38}{\sqrt{2 \cdot d}}$ (default), $\epsilon = 0.0001$) for the entire procedure.
The $\hat{R}$ diagnostic from ArviZ [24] was used to check for convergence (all $\hat{R} \approx 1$, Appendix A.3).

3.2.7 Visualization techniques

Plots were prepared from Python with a combination of `matplotlib` [26], ArviZ and PyMC3. We used POV-Ray to produce Figure 2 and Figure 4 and https://diagrams.net for technical drawings. Probability densities were visualized with the `pymc3.gp.utils.plot_gp_dist` helper function that overlays many polygons corresponding to percentiles of a distribution, creating the colorful bands seen in Figure 15 and others. Posterior predictive samples were obtained by randomly drawing observations from the calibration model, based on independent values sampled from the posterior distribution. If not stated otherwise, the densities plotted for MCMC prediction results were obtained from at least 1000 posterior samples. The pair-plot of 2-dimensional kernel density estimates of posterior marginals (Figure 17) was prepared with ArivZ.

4 Results and discussion

4.1 Application: Implementing (non-)linear calibration models with `calibr8`

A common application of calibration models in life sciences are enzymatic assays, where the quantification of glucose is one out of many popular examples. In this section, data from a glucose assay is used as a demonstration case for building calibration models with `calibr8` . First, the linear range of the assay is described by the corresponding linear calibration model to then explore an extended concentration range by implementing a calibration model with logistic trend of the location parameter. We examine a second calibration example that is nonlinear in its nature, namely the backscatter/biomass relationship of measurements with a BioLector Pro device (Section 3.1.1), to then demonstrate how uncertainty estimates for biomass concentrations can be easily obtained with `calibr8` .

4.1.1 Linear calibration model

To acquire data for the establishment of a calibration model, 96 glucose standards between 0.001 and 50 g/L were subjected to the glucose assay. A frequent approach to calibration modeling in life sciences is to identify the linear

range of an assay and to discard measurements outside this range. From a previous adaptation of the glucose assay for automation with liquid handling robotics, the linear range was expected to be up to 2 g/L (Holger Morschett, personal communication, 2019). Since samples are diluted by a factor of 10 before the assay, 83 glucose standards with concentrations below 20 g/L remain for a linear calibration model.
As described in Section 2.4, calibration models use a probability distribution to describe the relation between independent variable and measurement outcome, both subject to random noise. In this example, we chose a Student-*t* distribution, thus the change of location parameter μ over the independent variable determines the trend of the calibration model. `calibr8` provides a convenience class `BasePolynomialModelT` that was used to implement a glucose calibration model with linear trend (Code 1). For the spread parameter `scale`, we also chose a linear function dependent on μ to account for increasing random noise in dependency of the absorbance readout of the assay. Both can easily be adapted by changing the respective parameters `mu_degree` and `scale_degree` passed to the constructor of the convenience class. The degree of freedom ν in a `BasePolynomialModelT` is estimated from the data as a constant.

Code 1: Implementation of glucose/absorbance calibration model using convenience type

```
class LinearGlucoseCalibrationModelV1(base.BasePolynomialModelT):
    def __init__(self, *, independent_key:str='S', dependent_key:str='A365'):
        super().__init__(
            independent_key=independent_key,
            dependent_key=dependent_key,
            mu_degree=1,
            scale_degree=1
        )
```

The calibration model resulting from MLE of location and spread parameters was plotted with another `calibr8` convenience function (Figure 8 A-C). The plot shows the calibration model and measurement data (Figure 8 A), the same relation with a logarithmic x-axis (Figure 8 B) and the relative residuals of data and model predictions (Figure 8 C). As it is often recommended for biological assays, the glucose concentrations of the dilution series were evenly spaced on a logarithmic scale [52, 53], thus ensuring a higher accuracy of the model in the low-concentrated area (Figure 8 B). To evaluate the quality of the linear calibration model, the residuals of data and model prediction were analyzed (Figure 8 C). Overall, the residuals lie between $\pm 5\%$ of the observed values, demonstrating the high precision of the data. For concentrations higher than 0.6 g/L, an s-shaped trend is observed in the residuals, meaning that data first lies below and then above the linear model prediction. This indicates a lack-of-fit as described in Section 2.4. However, the discrepancy might also be caused by errors in the serial dilution that was pipetted with the robotic liquid handler, resulting in deviations from the expected linear relation. Moreover, it can be seen that the relative spread of residuals is quite constant, meaning that the absolute deviation increases with higher concentrations (Figure 8 C). Although the linearly increasing `scale` parameter accounts for this rise of the underlying random noise, it can be seen that it is slightly overestimated by the model since all data points above 2 g/L lie within a 90 % probability interval.
In comparison to simple linear regression, which is often evaluated by the coefficient of determination R^2 alone, the demonstrated diagnostics allow to judge whether the choice of model is appropriate. In this case, a more sophisticated model for the spread of the Student-*t* distribution could be chosen to reduce the lack-of-fit. Moreover, all data points lying above 20 g/L were not considered so far to allow for a linear model. In the following, we will therefore modify the existing calibration model to include a logistic function for the location parameter.

4.1.2 Logistic calibration model

Although linear calibration models are useful in many cases, some relations in datasets are non-linear in their nature. Moreover, restricting analytical measurements to an approximately linear range instead of calibrating all concentrations of interest can be limiting. If the order of magnitude of sample concentrations is unknown, this leads to laborious dilution series or repetition of measurements to ensure that the linear range is met. In contrast, non-linear calibration models allow to describe complex relationships and, in case of biological assays, to reduce these time- and material-consuming workflows.
Many recommendations for experimental design in calibration can be found in literature (e.g. [52]). Having determined the range of interest for the calibration model, it should be exceeded in both directions if possible, thus ensuring that the relevant concentrations are well-covered. This way, all model parameters, including limits where applicable, can be identified from the observed data. Afterwards, the expected relationship between dependent and independent variable is to be considered. Since the glucose assay readout is based on absorbance in a plate reader (Section 3.1.4), which has a lower and upper detection limit, a saturation effect at high glucose concentrations is expected. In our demonstration example, glucose concentrations of up to 50 g/L were targeted to cover relevant concentration for cultivation (Section 4.2) and at the same time to exceed the linear range towards the upper detection limit.
Sigmoidal shapes in calibration data, *e.g.* often observed for immunoassays, can be well-described by logistic func-

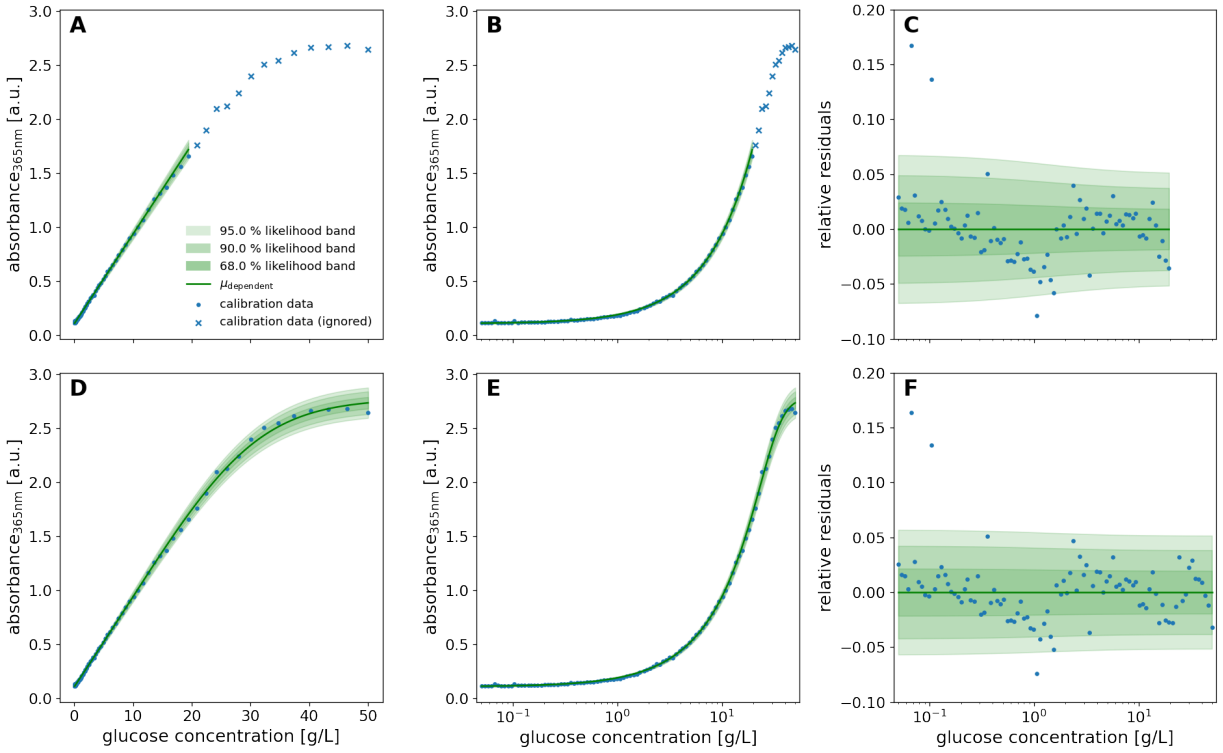

Figure 8: Linear (top) and logistic (bottom) calibration model of glucose assay
A calibration model comprising linear functions for both the location parameter μ_{A365} and the scale parameter of a Student-*t* distribution was fitted to calibration data of glucose standard concentrations (0.05-20 g/L) and absorbance readouts by maximum likelihood estimation (**A-C**). The calibration data used to fit the linear model is the 0.05-20 g/L subset of standards that were spaced evenly on a log-scale up to 50 g/L (**B, E**). Likewise, a calibration model with a 5-parameter asymmetric logistic function for the μ parameter of the Student-*t* distribution was fitted to the full 0.05-50 g/L calibration dataset (**D-E**). In both models, the scale parameter was modeled as a 1st-order polynomial function of μ and the degree of freedom ν as a constant. Standard concentrations were spaced evenly on a log-scale between 0.05 and 20 g/L (**B, E**). The extended range of calibration standard concentrations up to 50 g/L reveals a saturation kinetic of the glucose assay (**A, D**) and depending on the glucose concentration, the residuals (*C, F*) with respect to the modeled location parameter are scattered by approximately 5 %. Modeling the scale parameter of the distribution as a 1st-order polynomial function of μ describes the broadening of the distribution at higher concentrations (C).

tions [33]. In the `calibr8` package, a generalized logistic function with 5 parameters is used in an interpretable form (Section 3.2.1). It was used to implement a calibration model where the location parameter μ is described by a logistic function dependent on the glucose concentration. A respective base class `BaseAsymmetricLogisticT` is provided by `calibr8` (Appendix A.1). Using the whole glucose dataset up to 50 g/L, parameters of the new calibration model were estimated (Figure 8 D-F).
Overall, the logistic trend of the location parameter matches the extended calibration data well (Figure 8 D, E). Since the scale parameter of the Student-*t* distribution is modeled as a linear function dependent on μ, the width of the likelihood bands approaches a limit at high glucose concentrations (Figure 8 F). For concentrations greater than 3 g/L, no residuals lie outside of the 90 % probability interval, indicating that the distribution spread is overestimated as it was before. Importantly, a direct comparison between the two calibration models (Figure 8 C, F) reveals a high similarity in the reduced range (< 20 g/L). This demonstrates how a non-linear model extends the range of concentrations available for measurement and modeling while improving the quality of the fit. For the glucose assay, truncating to a linear range thus becomes obsolete.
While non-linear models were so far shown to be useful to extend the usable concentration range of an assay, other applications do not allow to linearly approximate a subrange of measurements. Such an example is the online detection of biomass in growth experiments, where the non-invasive backscatter measurement of a BioLector Section 3.1.1 does not allow for dilution of the cell suspension during incubation. To model the distribution of backscatter observations as a function of the underlying biomass concentration, a structure similar to the glucose

calibration model was chosen. In contrast, the location parameter μ was modeled by a polynomial function of the *logarithmic* cell dry weight (CDW). The final CDW/backscatter calibration model was implemented using the `calibr8.BaseLogIndependentAsymmetricLogisticT` convenience class Code 2.

Code 2: Implementation of CDW/backscatter calibration model using convenience type

```
class BioLectorCDWBackscatterModelV1(calibr8.BaseLogIndependentAsymmetricLogisticT):
    def __init__(self, *, independent_key:str='X', dependent_key:str='BS'):
        super().__init__(
            independent_key=independent_key,
            dependent_key=dependent_key,
            scale_degree=1
        )
```

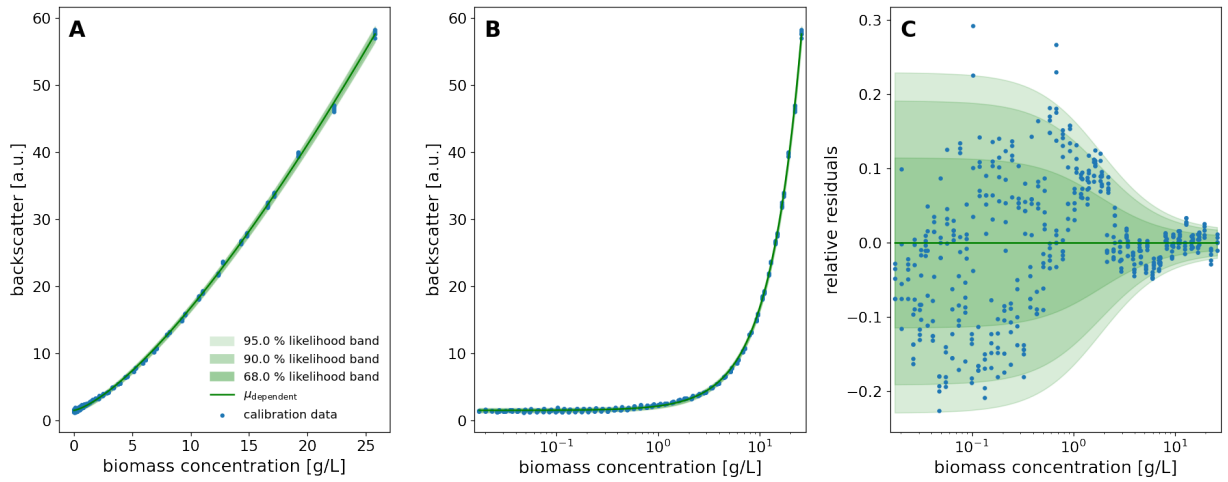

Figure 9: Calibration model of biomass-dependent backscatter measurement
Backscatter observations from two independent calibration experiments (1400 rpm, gain=3) on the same BioLector Pro were pooled. A non-linearity of the backscatter/CDW relationship is apparent already from the data itself (**A**). The evenly spaced calibration data (**B**) are well described with little lack-of-fit error (**C**). At low biomass concentrations the relative spread of the measurement responses starts at ca. 20 % and reduces to approximately 2 % at concentrations above 10 g/L.

Two independent experiments were conducted to obtain calibration data as described in Section 3.1.5. The model was fitted to pooled data using the `calibr8.fit_pygmo` convenience function. As shown in Figure 9, the model accurately describes the nonlinear correlation between biomass concentration and observed backscatter measurements in the BioLector Pro device (Figure 9 A, B). Non-linearity is particularly observed for biomass concentrations below 10 g/L (Figure 9 A). Moreover, the residual plot (Figure 9 C) mainly shows a random distribution; solely residuals between 1 and 3 g/L indicate a lack-of-fit. To assess the potential influence, the resulting uncertainty in estimated biomass concentrations has to be considered, which will be further discussed in Section 4.1.3. Overall, the chosen logistic calibration model describes the calibration data well and is thus useful to transform backscatter measurements from the BioLector device into interpretable quantitative biomass curves.
In summary, this section illustrated how calibration models can be built conveniently with `calibr8` and showed that the asymmetric logistic function is suitable to describe many relationships in biotechnological measurement systems. Having demonstrated how concentration/readout relations can be described by different calibration models, a remaining question is how to apply those calibration models. An important use-case is to obtain an estimate of concentrations in unknown samples, where uncertainty quantification is a crucial step.

4.1.3 Uncertainty quantification on independent variables

After establishing a calibration model, the practitioner can in most cases consider the parameters of the model as fixed. Characterization of measurement reproducibility is thereby externalized into the calibration procedure, where random noise is inherently described by the spread of a probability distribution. The calibration model can then be put into application for the quantification of the independent variable from new observations. As introduced before, not only a single value of the independent variable is desired, but also a measure of uncertainty about it.

Quantifying the uncertainty in the independent variable as a continuous probability density is not only intuitive to visually interpret (Section 2.4), but also flexible with respect to the question of interest. To quantify the uncertainty numerically, various kinds of credible intervals (Section 2.3) can be obtained. For example, one might estimate the equal-tailed interval in which the independent variable lies with 90 % probability, or alternatively the probability that it lies above a certain threshold.
In `calibr8` , the `CalibrationModel.infer_independent` method is used to perform the uncertainty quantification from one or more observations. Internally, it uses the `loglikelihood` method of the calibration model and numerically integrates the sum of log-likelihoods over a user-specified range of plausible independent variables (Section 3.2.3). The resulting `calibr8.NumericPosterior` is equivalent to Equation 14, where the prior $p(x)$ is specifying the plausible range.

$$p(x \mid \vec{y}_{obs}) = \frac{\mathcal{L}(x \mid \vec{y}_{obs}) \cdot p(x)}{\int_{-\infty}^{\infty} \mathcal{L}(x \mid \vec{y}_{obs}) \cdot p(x)\, dx} = \frac{\mathcal{L}(x \mid \vec{y}_{obs}) \cdot p(x)}{\int_{a}^{b} \mathcal{L}(x \mid \vec{y}_{obs})\, dx} \tag{14}$$
$$\text{where } p(x) = \text{Uniform}(a, b)$$

For convenience, the `CalibrationModel.infer_independent` method automatically determines median and credible interval (ETI and HDI) bounds. It determines vectors for the independent variable and the conditional probability density that can be plotted without further processing.

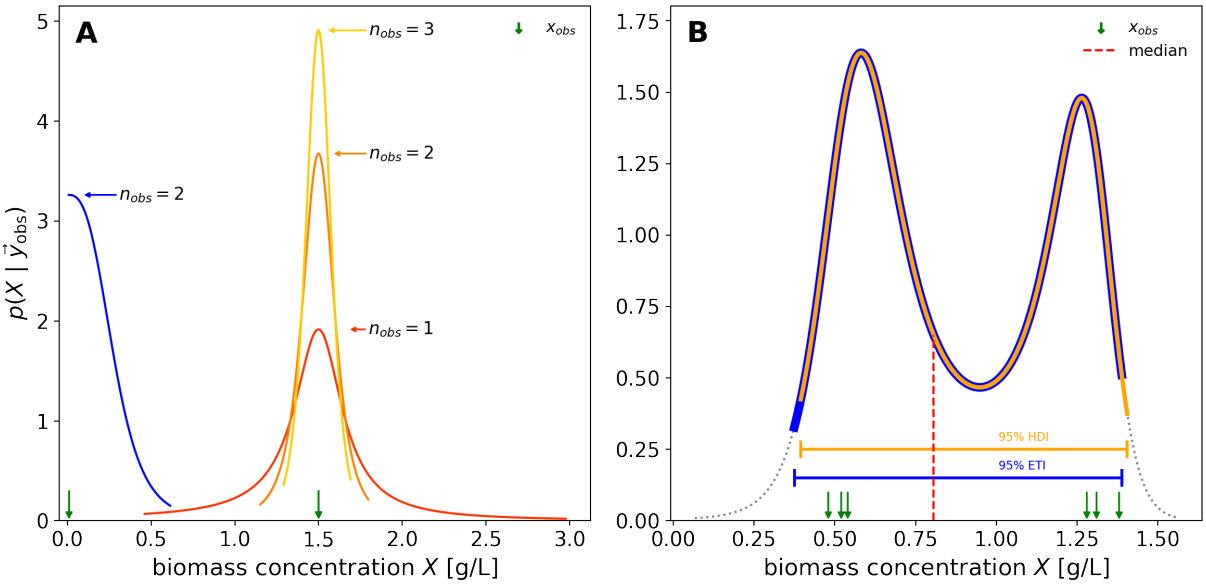

Figure 10: Independent variable PDFs in various observation scenarios
Posterior densities inferred from various numbers of observations corresponding to different biomass concentrations are shown (**A**). The ends of the drawn lines in **A** indicate the 95 % equal-tailed interval. Near biomass concentrations of 0, the posterior density is asymmetric (**A, blue**), indicating that very low concentrations can not be distinguished. As the number of observations grows, the probability mass is concentrated and the ETIs shrink (**A, oranges**). The choice of a Student-*t* distribution model can lead to a multi-modality of the inferred posterior density when observations lie far apart (**B**). For asymmetric distributions, the median (dashed line) does not necessarily coincide with a mode and equal-tailed and highest-density intervals (ETI, HDI) can be different. Maximum likelihood estimates from individual observations, as obtained via `predict_independent` are shown as arrows. Note: $\vec{y}_{obs}$ and the model's ν parameter were chosen at extreme values for illustrative purposes.

In Figure 10, various inferences obtained with `infer_independent` are illustrated with a biomass calibration model. For example, observations in the lower or upper saturation of the measurement system typically result in one-sided probability densities, and repetitive observations result in a narrowing of the distribution (Figure 10, A).
When the calibration model assumes the possibility of outliers (Student-*t* distributed measurement responses), the observation of drastically different measurement responses can translate into a multi-modal posterior belief in the independent variable. The intuition behind this multi-modality is that a subset of observations are "outliers" from the perspective of the remaining observations and vice versa. In the example shown in Figure 10, the three observations

around 0.5 could be "outliers", or the ones around 1.3, but from the data alone both are equally likely. Hence the posterior belief in the biomass concentration is bimodal.
The Bayesian, or likelihood-based perspective on uncertainty in the independent variable (Equation 14) allows for quantification of uncertainty even with single observations, close to zero, or close to saturation limits of the measurement system. Calibration models built with `calibr8` are easy to set up, visualize and diagnose and can thus be flexibly integrated into existing data analysis workflows of various domains. Moreover, the set-up in a versatile, object-oriented programming language such as Python allows to use `calibr8` in high-throughput, automated experiments where hundreds of calibration models must be fitted. Next, we will build upon the presented biomass and glucose calibration models and demonstrate how combining them with a bioprocess model enables to gain insight into the growth phenotype of a biotechnological model organism, *C. glutamicum*.

4.2 Application 2: Process modeling of bacterial growth

A real-world experimental procedure is often not a textbook example, but rather a heterogeneous dataset, *e.g.* comprising multiple measurement types or varying process conditions. We use the term *process model*, as introduced in Section 2.5, to describe the complete underlying chemical or biological process, but not the experimental observations that are made of it. These input/output relations of the measurement system are explicitly described by calibration models. In this application example, we demonstrate how object-oriented `calibr8` calibration models from Section 4.1 can be combined with an ODE bioprocess model to describe a heterogeneous dataset of bacterial growth curves.
The simplest experimental setup to obtain a bacterial growth curve is a so-called batch cultivation. Under laboratory conditions, such batch cultivations can be performed in a variety of cultivation systems such as shake flasks, bioreactors or microbioreactors. From a data science perspective, the cultivation systems differ mostly by how many independent cultivations are performed in parallel and by the kind and number of observations made per cultivation. In the domain of bioprocess development, a large number of cultivations must be conducted to find best-performing producer strains and media compositions. For these applications, microbioreactors offer an increased cultivation throughput combined with non-invasive *on-line* measurements of pH, dissolved oxygen tension (DO) and, in case of the BioLector, also biomass [54]. However, all three signals are obtained optically and must be calibrated against the true variable of interest (Section 3.1.1, Section 3.1.5). Furthermore, confounding factors are known for all three measurement methods, mandating special rigor in the design and analysis of quantitative experiments. For example, the optode-based pH and DO measurements can be influenced by media components, or the backscatter signal by morphology changes.
To facilitate a simple application example, we grew *Corynebacterium glutamicum* in a BioLector Pro device (Section 3.1.2). This bacterium is a well-known industrially applied microorganism that exhibits textbook-like exponential growth kinetics when grown on carbon sources such as glucose [55]. A preculture was grown in the BioLector wells A01 and B01 and used to automatically inoculate 28 main culture wells (A02 through F08). We thus avoided a lag phase of adaptation at the beginning of the growth curve, which greatly simplifies the process model (Section 3.1.2). As we will see later on, the pipetting error of the robotic liquid handler at the small inoculation volume must be considered when setting up the process model, highlighting the need to adapt the data analysis to the peculiarities of the experiment. Before going into the details of the process model for this application example, we would like to emphasize that the same modeling techniques can be applied to other domain specific examples.

4.2.1 Building an ODE process model for bacterial growth experiments

The simplest model for microbial growth is the Monod kinetics differential equation model of substrate-limited exponential growth [56]. Similar to how the famous Michaelis-Menten kinetics describe enzymatic reaction rates, the Monod kinetics model the specific growth rate as a function of substrate concentration. Under the assumptions of homogeneous mixing, unlimited nutrient supply and constant ambient conditions, the Monod model can be applied to batch cultivations of bacterial, fungal, plant or cell cultures that grow with a maximum growth rate μ_{max} until a substrate, typically a carbon source, is depleted.
The Monod model (Equation 15) has five parameters including the initial conditions for substrate concentration S_0 and biomass concentration X_0. The maximum growth rate μ_{max} specifies the specific exponential growth rate that the organism can achieve under the modeled conditions. The actual specific growth rate $\mu(t)$ is modeled as a function of μ_{max}, the current substrate concentration S and a parameter K_S that corresponds to the substrate concentration at which $\mu(t) = \frac{\mu_{\text{max}}}{2}$. The last parameter Y_{XS}, called biomass yield, describes the amount of substrate consumed per unit of formed biomass.

$$\begin{aligned} \frac{dX}{dt} &= \mu_{\text{max}} \cdot X \cdot \frac{S}{K_S + S} \\ \frac{dS}{dt} &= -Y_{\text{XS}} \cdot \frac{dX}{dt} \\ S, X, \mu_{\text{max}}, K_S, Y_{\text{XS}} &\in \mathcal{R}_{>0} \end{aligned} \tag{15}$$

The experiment to be modeled in this application example was devised such that Monod-like growth behavior of *C. glutamicum* wild-type could be expected (Section 3.1.2). We grew 28 parallel batch cultures that were sampled to measure glucose concentrations in addition to the high resolution backscatter time series. The resulting dataset comprises 28 *replicates*, each with backscatter time series of varying length and a time series of length 1 for the glucose absorbance readout. Building upon our Python package `murefi` for flexible **mu**lti-**re**plicate **fi**tting, we loaded the raw observations into a `murefi.Dataset` object (Section 3.2.4). The package was designed to simplify the definition and parameter estimation of process models that describe all replicates in a dataset simultaneously.
To build such elaborate process models with `murefi`, the user must specify the process model corresponding to a single replicate, as well as a set of rules that describe how parameters of this model are shared across replicates. The Monod kinetics in this application example were implemented in just a few lines of code by subclassing from `murefi.BaseODEModel` (Code 3).

Code 3: Implementation of Monod ODE model using `murefi.BaseODEModel` convenience type

```
class MonodModel(BaseODEModel):
    def __init__(self):
        super().__init__(
            theta_names=('S0', 'X0', 'mu_max', 'K_S', 'Y_XS'),
            independent_keys=['S', 'X']
        )

    def dydt(self, y, t, theta):
        S, X = y
        mu_max, K_S, Y_XS = theta
        dXdt = mu_max * S * X / (K_S + S)
        dSdt = -1 / Y_XS * dXdt

        return [
            dSdt,
            dXdt,
        ]
```

For heterogeneous datasets, the rules for sharing process model parameters across replicates can be complex and hard to implement and most modeling workflows require the practitioner to often change the parametrization. In `murefi` , the `ParameterMapping` class supports the modeler by specializing in the tedious translation of parameter sharing rules into a function (`.repmap(...)`) that takes a single parameter vector and transforms it into replicate-specific parameter vectors. At the same time, it provides mechanisms for specifying fixed parameters, initial guesses and bounds on the parameters. Reading a spread sheet with parameters into Python is an easy way of initializing the `ParameterMapping` (Figure 11).

Unique names specify that a parameter is only estimated from the indicated replicate (*e.g.* X0_A02) while shared names correspond to global parameters (*e.g.* S0). For the application example at hand, a parameter mapping was defined such that the parameter X_0 is local to each replicate while S_0, μ_{max} and Y_{XS} are shared across all replicates. For the Monod substrate affinity constant K_S, literature reports values of approximately 0.00005-0.1 g/L for *Escherichia coli* [57]), while no data is available for *C. glutamicum*. Because it is practically non-identifiable at the resolution of our dataset, K_S was fixed to an arbitrary, but numerically harmless value of 0.02 g/L. In Figure 11, this is expressed by the numerical column entries.
A likelihood function for parameter estimation was created using the `murefi.objectives.for_dataset` convenience function (Code 4). The objective is independent of the parameter estimation paradigm and was applied for optimization via MLE (Section 4.2.2) and sampling by MCMC (Section 4.2.3) in the scope of this work.

4.2.2 Estimating ODE process model parameters by maximum likelihood

First, we determined maximum likelihood estimates of the process model parameters through optimization. In few lines of code, the calibration models from Section 4.1 and dataset are loaded (Code 4, ll. 2-4), the process model is

rid	S0	X0	mu_max	K_S	Y_XS
A02	S0	X0_A02	mu_max	0.02	Y_XS
A03	S0	X0_A03	mu_max	0.02	Y_XS
A04	S0	X0_A04	mu_max	0.02	Y_XS
A05	S0	X0_A05	mu_max	0.02	Y_XS
A06	S0	X0_A06	mu_max	0.02	Y_XS

Figure 11: Tabular `DataFrame` representation of a parameter mapping
With columns corresponding to the parameter names of a naive Monod process model, the parametrization of each replicate, identified by a *replicate ID (rid)* is specified in a tabular format. Parameter identifiers that appear multiple times (*e.g.* S0) correspond to a parameter shared across replicates. Accordingly, replicate-local parameters names simply do not appear multiple times (*e.g.* X0_A06). Numeric entries are interpreted as fixed values and will be left out of parameter estimation. Columns do not need to be homogeneously fixed/shared/local, but parameters can only be shared within the same column.

instantiated (Code 4, l. 1) and the `ParameterMapping` is specified with bounds and guesses (Code 4, ll. 7-21). The `objective` (Code 4, ll. 22-27) can directly be used for an optimization algorithm (Code 4, ll. 28-32), in this case one from the popular Python library `scipy`. A table with MLE parameters can be found in Appendix A.3.

Code 4: MLE of process model parameters

```
1  model = MonodModel()
2  dataset = murefi.load_dataset("cultivation_dataset.h5")
3  cm_biomass = BioLectorCDWBackscatterModelV1.load("biomass_cm_logistic.json")
4  cm_glucose = LogisticGlucoseCalibrationModelV1.load("glucose_cm_logistic.json")
5  df_mapping = pandas.read_excel("parameter_mapping.xlsx", index_col='rid')
6
7  theta_mapping = murefi.ParameterMapping(
8      df_mapping,
9      bounds={
10         "S0": (15, 20),
11         "X0": (0.01, 0.4),
12         "mu_max": (0.4, 0.5),
13         "Y_XS": (0.3, 1)
14     },
15     guesses={
16         "S0": 17,
17         "X0": 0.01,
18         "mu_max": 0.4,
19         "Y_XS": 0.5
20     }
21 )
22 objective = murefi.objectives.for_dataset(
23     dataset=dataset,
24     model=model,
25     parameter_mapping=theta_mapping,
26     calibration_models=[cm_glucose, cm_biomass]
27 )
28 mle_result = scipy.optimize.minimize(
29     objective,
30     x0=theta_mapping.guesses,
31     bounds=theta_mapping.bounds
32 )
```

Figure 12 shows the observations alone (A) and combined with MLE results (B) for glucose (absorbance) and biomass (backscatter). The replicates were sampled at different times to measure glucose concentrations; the end of a time series is indicated by an arrow and the replicate name (Figure 12, A). Overall, the backscatter time series show a very high reproducibility, which demonstrates the effect of pooling precultures before inoculation (Section 3.1.2). The model describes the observations so accurately that they can only be distinguished in the inset plot (Figure 12, B). Here, a small

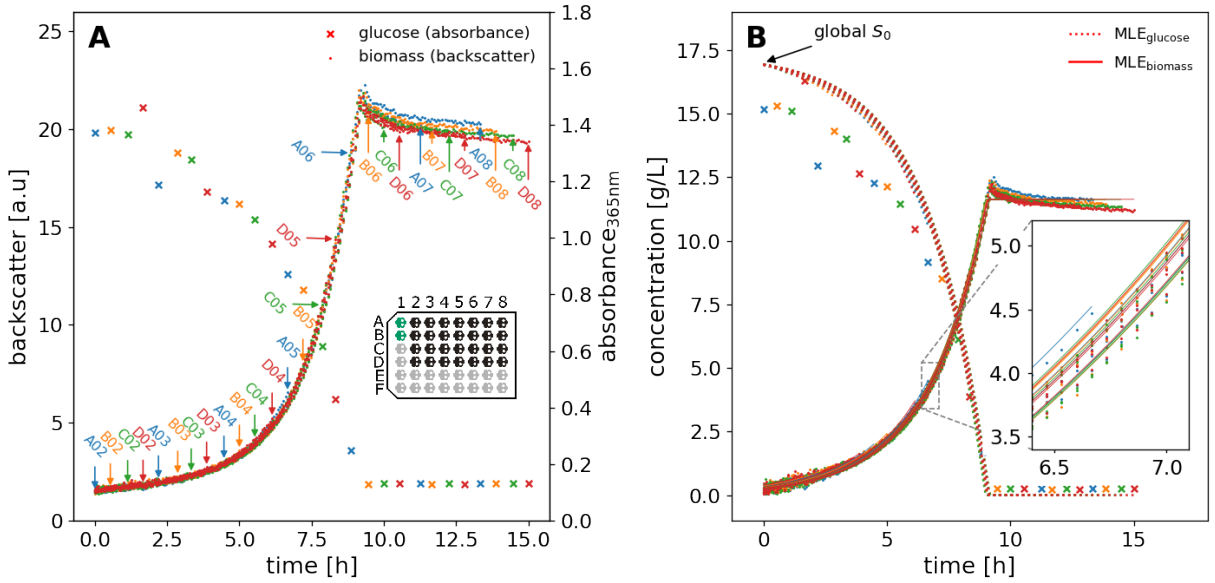

Figure 12: Measurements and maximum likelihood estimate of *C. glutamicum* growth Monod model
Original measurement responses of *on-line* biomass (backscatter) and *at-line* endpoint glucose assay measurements (absorbance) are shown in (**A**). Glucose measurements were obtained by sacrificing culture wells, hence each backscatter time series terminates at the time of glucose assay observations. The time and well ID of sacrifices are marked by arrows, colored by row in the cultivation FlowerPlate. The inset plot shows a typical FlowerPlate layout. The preculture wells (data not shown) are highlighted in green, main cultures in black.
In **B**, the observations and MLE predictions of the ODE process model are shown in SI units. Observations were transformed from original units using the `predict_independent` method of the respective calibration model. Whereas all curves start at the same global initial substrate concentration S_0, each well has individual initial biomass concentrations, resulting in the time shifts visible in the zoomed-in inset plot. Biomass observations in the inset plot (•) correspond to the median posterior inferred from each backscatter observation individually.

difference between different replicates can be observed, which is caused by different initial biomass concentrations due to inevitable pipetting errors in the automated inoculation of the main cultures. It becomes evident that replicate-wise X_0 parameters were necessary to account for this effect. The different initial biomasses are also visible from the spread of data points at the beginning of the growth curve (Figure 12, B). For the biomass, the only period of systematic deviation between model prediction and observations is at the time of entry into the stationary phase, the phase where substrate is depleted and growth stops. Here, the biomass signal overshoots while the Monod kinetics predict a rapid change to a constant signal. This effect in the growth curve of *C. glutamicum* is also known from other experiments with the BioLector [58] and cannot be accounted for by the otherwise useful textbook process model.
The glucose data shows more deviation, but follows the expected textbook behaviour of exponential decay (Figure 12, B). Interestingly, the predictions for glucose concentrations at the end of cultivation lie slightly above $0\ \frac{g}{L}$, showing that the corresponding calibration model is not describing this range of concentrations well. The deviation could be caused by other components in the used cultivation medium that distort the measurement compared to calibration with fresh medium as diluent. However, this was not further investigated since the substrate data has little influence on the parameter estimation compared to the high-resolution backscatter measurements.
From a first inspection of MLE results, we can see that the simple Monod process model describes the high-resolution data very well. For more insight, we will take a look at the parameter estimation, correlations and systematic deviations using a Bayesian approach.

4.2.3 Hierarchical Bayesian ODE models with `calibr8` and `murefi`

The results presented in the previous chapter show that the Monod model, when combined with non-linear calibration models for the observations, can describe the observed biological process with high accuracy. However, the precision of the parameter set obtained by the maximum likelihood method is still unknown. Particularly, when decisions are made from model-based inferences and predictions, the uncertainty about these variables is a key factor.
The combination of (forward) sensitivity analysis with Gaussian error propagation could be applied to learn about

the precision of the maximum likelihood estimate. Instead of maximum likelihood optimization of a parameter set, Bayes' rule can be used to infer a posterior probability distribution of parameters. In comparison to the maximum likelihood method, the Bayesian approach allows to incorporate prior knowledge and inherently quantifies uncertainty and parameter correlations. Bayesian posteriors can in some (rare) cases be obtained analytically, or numerically as shown in Section 4.1.3. However, in most practical applications Markov chain Monte Carlo (MCMC) algorithms are applied. MCMC offers convergence guarantees as the number of iterations approaches infinity and can give satisfactory results with competitive computational performance when modern algorithms are used.

To build a Bayesian process model, one must explicitly state prior beliefs in the model parameters in the form of probability distributions. For our hierarchical Monod model application example, we must specify prior beliefs in the ODE parameters μ_{max}, Y_{XS} and initial conditions S_0 and $X_{0,\text{well}}$. Prior distributions for these parameters were specified to reflect biologically reasonable, but uninformative assumptions about the experiment Equation 16. The initial substrate concentration S_0 was expected at approximately 20 $\frac{g}{L}$ with a conservative 10 % relative error. For *Corynebacterium glutamicum* wild-type, our priors for biomass yields with $\text{HDI}^{95\ \%}_{Y_{\text{XS}}} = [0.5,\ 0.7]\ \frac{g_{\text{CDW}}}{g_{\text{glucose}}}$ and for maximum growth rates with $\text{HDI}^{95\ \%}_{\mu_{\text{max}}} = [0.2,\ 0.6]\ h^{-1}$ are uninformative and based on literature [59]. Our process model describes initial biomass concentrations on a per-well basis (Section 4.2.1), but can still infer the mean initial biomass concentration $X_{0,\mu}$ as a *hyperprior* by modeling well-specific offsets w.r.t. the group mean as $X_{0,well} = X_{0,\mu} \cdot F_{\text{offset,well}}$. Through $X_{0,\mu}$ the priors for all initial biomass concentrations $\vec{X_0}$ are parametrized by a common parameter, allowing each individual $X_{0,\text{well}}$ to vary while concentrating around their common group mean. For more intuition and details about Bayesian hierarchical modeling in particular, we refer to [60].

The experiment was programmed to inoculate main cultures to approximately 0.25 g/L (Section 3.1.2), therefore the prior for $X_{0,\mu}$ was centered at 0.25 g/L with a 10 % relative error. Our prior belief in the well-specific relative offset $F_{\text{offset,well}}$ was also modeled by a Lognormal distribution with mean 0, corresponding to the expectation that the offset is centered around 1 in the transformed space. A standard deviation of 20 % was chosen to account for random and systematic inaccuracy of the automated liquid handler at the low pipetting volume of 20 µL [58].

$$
\begin{aligned}
X_{0,\mu} &\sim \text{Lognormal}(\mu = \log(0.25), \sigma = 0.1) \\
\vec{F_{\text{offset}}} &\sim \text{Lognormal}(\mu = 0, \sigma = 0.2) \\
\vec{X_0} &\sim X_{0,\mu} \cdot \vec{F}_{\text{offset}} \\
\\
S_0 &\sim \text{Lognormal}(\mu = \log(20),\ \sigma = 0.1) \\
Y_{\text{XS}} &\sim \text{Beta}(\mu = 0.6,\ \sigma = 0.05) \\
\mu_{\text{max}} &\sim \text{Beta}(\mu = 0.4,\ \sigma = 0.1) \\
\\
Y_{\text{pred, well}} &\sim \phi_{\text{process model}}(S_0,\ X_{0,\text{well}},\ \mu_{\text{max}},\ Y_{\text{XS}}) \\
\mathcal{L}(\theta_{\text{pm}} \mid Y_{\text{obs}}) &= p(Y_{\text{obs}} \mid \theta_{\text{em}}(Y_{\text{pred}}))
\end{aligned}
\tag{16}
$$

When modeling with `calibr8` and `murefi` , this specification of prior beliefs is the only overhead compared to the MLE method. The API of both packages was designed to be fully compatible with the probabilistic programming library PyMC3, such that `calibr8` and `murefi` models can become fully Bayesian with little programming effort. Concretely, the objective function created by `murefi` accepts `Aesara` tensors (*e.g.* PyMC3 random variables) as inputs, resulting in a symbolic `TensorVariable` likelihood instead of a numeric one. The PyMC3 model for the hierarchical ODE process model in our application example builds upon the previously established `objective` function (Code 4, l. 22). The model code (Code 5) resembles the mathematical notation of the same model shown in Equation 16.

Code 5: Specification of complete process model in PyMC3

```
1  with pymc3.Model() as pmodel:
2      # Specify a hyperprior on the initial biomass group mean:
3      # + centered on the planned inoculation density (0.25 g/L) in main cultures
4      # + with a 10 % standard deviation to account for pipetting errors
5      X0_mu = pymc3.Lognormal('X0_mu', mu=numpy.log(0.25), sd=0.10)
6
7      # Model the relative offset of initial biomass between each well and
8      # the group mean with a relative pipetting error of 20 %
```

```
    F_offset = pymc3.Lognormal('F_offset', mu=0, sd=0.20, shape=(N_wells,))

    # Thereby, the initial biomass in each well is the product
    # of group mean and relative offset:
    X0 = pymc3.Deterministic('X0', X0_mu * F_offset)

    # Combine the priors into a dictionary
    theta = {
        'S0': pymc3.Lognormal('S0', mu=numpy.log(20), sigma=0.10),
        'Y_XS': pymc3.Beta('Y_XS', mu=0.6, sd=0.05),
        'mu_max': pymc3.Beta('mu_max', mu=0.4, sd=0.1),
        # unpack the vector of initial biomasses into individual scalars
        **{
            f'X0_{well}': X0[w]
            for w, well in enumerate(wells)
        }
    }
    # Re-use the objective function from the MLE model code
    L = objective(theta)
```

After the PyMC3 process model was defined, its parameters were estimated by MCMC as described in Section 3.2.6. Two-dimensional marginals of the posterior samples obtained from MCMC sampling are shown in Figure 13 for two replicates and in Figure 17 for the whole dataset.

The pair plot visualization of the posterior reveals that some model parameters are strongly correlated with each other. Among those strong correlations are the pair of initial substrate concentration S_0 and biomass yield Y_{XS}. Interestingly, even in the very narrow HDIs of $X_{0,\text{well}}$ and μ_{max}, correlations were found, which is particularly clear for replicate D06. An interpretation is that when the initial biomass concentration is estimated at a smaller value, the maximum growth rate of cells must be higher to reach the same biomass level. The correlation is thus a natural consequence of the underlying process. Similarly, a lower initial substrate concentration results in a higher yield.
From a modeling point of view, the plot reveals how identifiable the model parameters are from the data. Furthermore, strong correlations, as observed for Y_{XS} and S_0, can be problematic for some optimization or MCMC sampling algorithms. In this case, the applied algorithm DE-Metropolis-Z [44] proved beneficial to sample the 32-dimensional parameter space with highly correlated parameters (Figure 13, top right). Interestingly, the strength of the correlation depends on the amount of data that was available for a particular replicate (Figure 17). The more data available, the stronger the correlation between X_0 and μ_{max}; this can also be observed for wells D04 and D06. The parameter estimates by MCMC are also tabulated in Appendix A.3.
In the lower part of Figure 13, the observations as well as model predictions with parameters sampled from the posterior are shown. Each line in the density plot corresponds to one set of parameters sampled with the MCMC algorithm. The small width of the density bands express how well the parameters could be estimated from the data, which is in accordance to the pair plot above. The violins around the substrate data visualize the uncertainty of glucose concentration inferred with the calibration model alone, instead of using the process model with all evidence. The violin is wider than the posterior band from the process model accordingly. Similar to the the MLE results, it becomes obvious that the Monod model estimate is well-suited to describe the biological dataset. With `calibr8` and `murefi`, building and sampling the Bayesian model needs a similar effort as MLE and the user can focus on structural requirements rather than cumbersome implementation. To assess the benefits of the Bayesian model in more detail, the role of different calibration models, the residuals and the hierarchical parameter X_0 are investigated in more detail in the next section.

4.2.4 Process and model insight through Bayesian uncertainty quantification

From the process model fit and the uncertainty estimates in particular, conclusions about the choice of model and the underlying biological process can be drawn. First, to emphasize that the elaborate non-linear calibration model was required, we compare the process model fits obtained with a non-linear versus a linear calibration model. The more traditional linear biomass/backscatter correlation was fitted to calibration data as described in Section 3.1.5 and used to fit the D06 replicate from our dataset. For comparison, the asymmetric logistic calibration model from Section 4.1.1 was used to estimate parameters of the same process model and data.

On a first glance, the fit of the Monod process model using the linear biomass calibration model looks like a good description of the data (Figure 14 A), but does not hold up to closer inspection. The residual plots (B, C) reveal that using the linear calibration model results in systematically larger residuals of the process model, compared to using the logistic calibration model. A thorough inspection of the linear calibration model itself (D) also reveals that it already has a lack-of-fit of the location parameter (green line), similar to the depiction in Figure 3. We would like to point

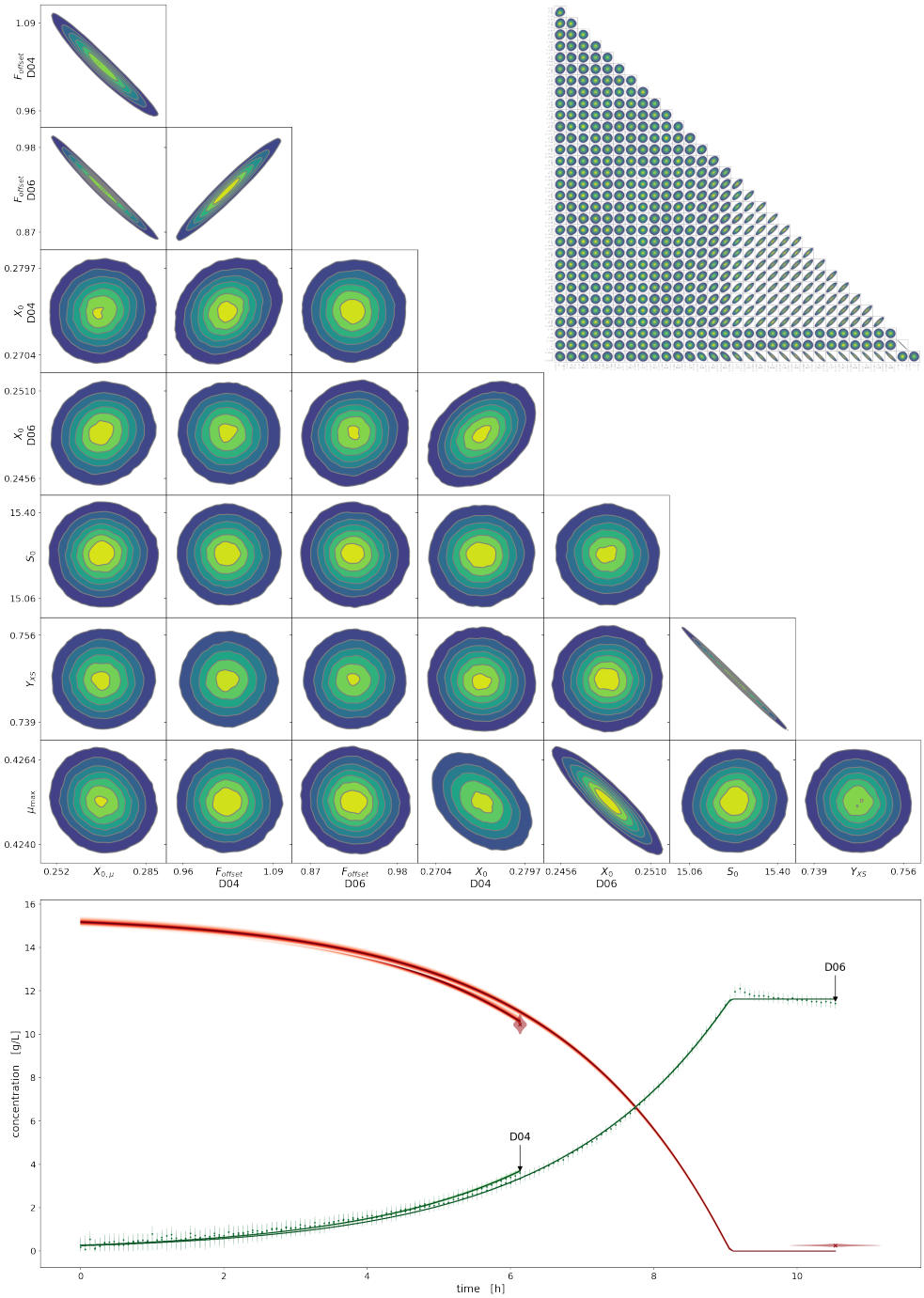

Figure 13: Parameter correlations, data and posterior distributions of growth curves
Each kernel density estimate (KDE) in the top half shows a 2-dimensional cross-section of the full posterior, visualizing correlations between some of the model parameters. For example, the topmost KDE shows that the posterior samples of $F_{\text{offset,D04}}$ are correlated with $X_{0,\mu}$. Axis labels correspond to the lower- and upper-bound of 90 % HDIs. The large pair plot shows just the marginals that are relevant for the replicates D04 and D06, whereas the small pair plot shows the dimensions for all parameters (high resolution in appendix). In the bottom half of the figure, the kinetics of replicates D04 and D06 are drawn. The red (substrate) and green (biomass) densities correspond to the distribution of predictions obtained from posterior samples, as described in Section 3.2.7. The red violins visualize the posterior inferred from single glucose measurement responses without the use of the process model. Likewise, the green vertical bars on the biomass concentrations show the 90% HDI.

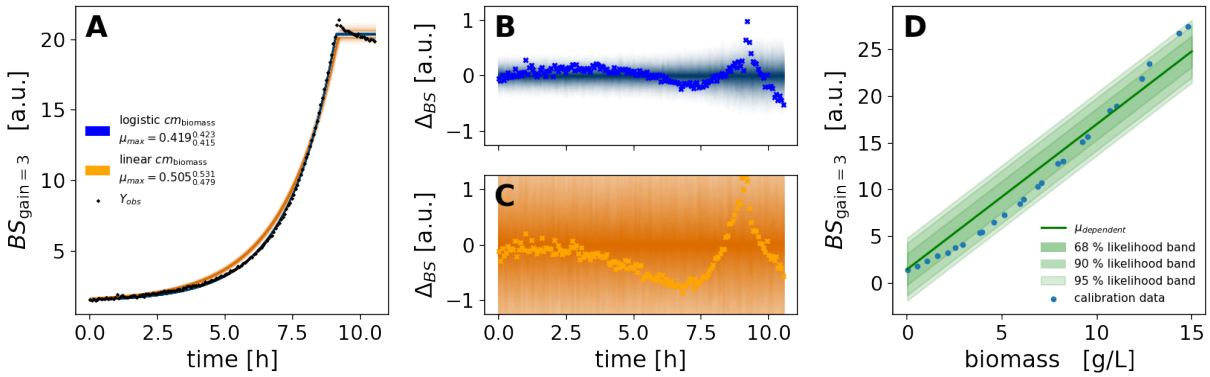

Figure 14: Comparison of Monod model fit with linear error model
Two Monod kinetic process models were fitted to the same observations from culture well D06 utilizing either a linear calibration model for the biomass/backscatter relationship (**D**, orange) or the previously established logistic model (blue). In **A** the posterior distribution of backscatter observations (density bands) is overlaid with actual backscatter observations. A linear calibration model with fixed intercept (Section 3.1.5) **D** was fitted to the subset of calibration data points up to 15 g/L such that it covers the range of biomass concentrations expected in the experiment. Residual plots of the observations compared to the posterior predictive distribution of backscatter observations (**B, C**) show that the fit obtained with the logistic calibration model (blue) has much less *lack-of-fit* compared to the one with the linear model (orange). Note that the backscatter residuals of ± 1 % are small compared to the amplitude of the absolute values going from close to 0 to approximately 20. The discrepancy between the two models is also evident from the 90 % HDI of the maximum growth rate μ_{max} of $[0.415,\ 0.423]\ h^{-1}$ in the logistic and $[0.479,\ 0.531]\ h^{-1}$ in the linear case.

out that also the maximum growth rate estimated from a process model with linear biomass/backscatter calibration ($\text{HDI}^{90\ \%}_{\mu_{\text{max}}} = [0.479,\ 0.531]$) is systematically overestimated compared to the one with the logistic model ($\text{HDI}^{90\ \%}_{\mu_{\text{max}}} = [0.415,\ 0.423]$). Regarding the choice of calibration model for the biomass/backscatter relationship, we conclude that the linear model should no longer be used, as it results in biased parameter estimates.

Having chosen a suitable calibration model for the variables, the choice of the Monod model itself can be investigated. Figure 15 shows the high-resolution biomass data and predictions from MCMC on a logarithmic y-scale (Figure 15, A) as well as the residuals in backscatter units (Figure 15, B). In the left subplot, the data was transformed to biomass concentrations with the logistic biomass calibration model. The orange intervals represent the $\text{HDI}^{90\ \%}_{\text{biomass}}$ inferred from a single observation using only the calibration model. In contrast, the blue density represents the posterior of the process model, which contains all observations. Naturally, the posterior from all evidence, combined through the process model, is much narrower than the posterior from any single observation. The plot reveals that the exponential growth assumed by the Monod model is generally suitable for the growth on glucose, since the blue density is describing the trend of observations well.
To evaluate a lack-of-fit, the residual plot (Figure 15, B) should be considered. Here, the residuals between the process model posterior and the observed backscatter are shown in black, the respective posterior predictive distribution of measurement responses (Section 3.2.3) is shown in green. The posterior predictive is the distribution of measurement responses that the model predicts. First, biomass concentrations are drawn from the posterior distribution. At each biomass concentration, another sample is taken from the Student-*t* distribution predicted by the biomass calibration model.
First of all, a large deviation that cannot be explained with the uncertainty of the estimate can be observed after 8 hours. Looking at the data, *e.g.* in Figure 14, it can be seen that it accounts for the previously described overshoot of the backscatter signal at the beginning of the stationary phase (Section 4.2.2). This phenomenon cannot be explained by the Monod model, which assumes a constant biomass concentration after substrate depletion. Further investigations are needed to identify whether the change is morphological, *e.g.* a shrinking of cells to due carbon source depletion, or a decrease of biomass, *e.g.* by cell lysis.
Before 8 hours, an s-shaped systematic deviation can be observed, meaning that the observations first lie above and then below the prediction. Apart from the influence of the overshoot, which distorts the fit, this might be explained by a different growth rate. It was previously shown that *C. glutamicum* exhibits a higher specific growth rate on protocatechuic acid (PCA), which is a component of the cultivation medium CGXII [59]. Upon depletion of PCA after the first hours of cultivation, the growth rate decreases accordingly. This is not accounted for in the Monod kinetics, which describe an effectively constant growth rate at substrate concentrations much higher than the K_S value. To cover

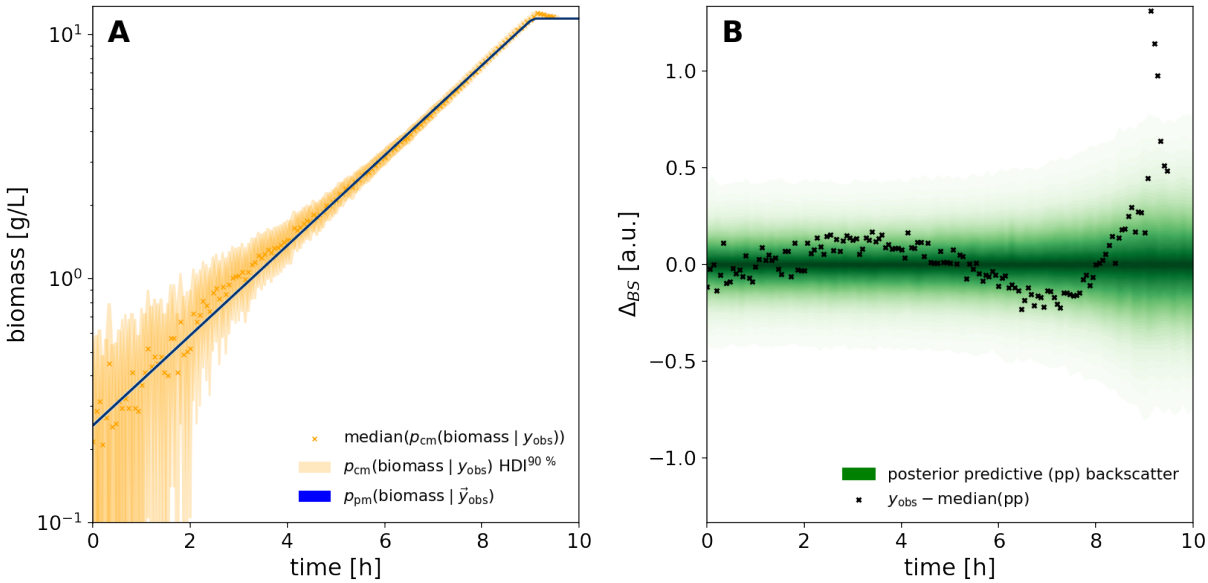

Figure 15: Predictions, observations and residuals of Monod model fitted to backscatter data
A: Through a logarithmic y-axis, the plot A shows that both process model (blue density) and the $\text{HDI}^{90\ \%}_{\text{biomass}}$ obtained from the biomass calibation model with individual observations (orange) describe an exponentially increasing biomass concentration up to approximately 9 hours. B: The residuals between prediction and observed backscatter (black) and the posterior predictive backscatter distribution (green density) show that the *lack-of-fit* is consistently less than ± 0.25 backscatter units with the exception of a fluctuation at the time of substrate depletion.

this effect, PCA must be measured, *e.g.* by sampling and liquid chromatography, and a more elaborate process models with several substrates must be utilized. Nevertheless, the very simple Monod kinetics describe the overall growth behaviour well and residuals are low.

In Figure 12, we have seen that the time differences in the exponential phases between replicates are well explained by the well-wise initial biomass concentrations $\vec{X}_0$. The choice of a hierarchical process models is further evaluated in Figure 16, which shows the estimated $\vec{X}_0$ with uncertainties for all replicates. For replicates with more evidence (longer time series), the posterior probability for their initial biomass concentration is concentrated in a narrow interval, whereas X_0 in wells with little evidence was estimated with considerably more uncertainty. The posterior for the group mean $X_{0,\mu}$ is concentrated at $\text{HDI}^{90\ \%}_{X_{0,\mu}} = [0.251,\ 0.286]\ \frac{g}{L}$, close to the theoretical concentration ($0.25\ \frac{g}{L}$) expected from the experimental design.
Overall, the well-wise modeling of initial biomass concentrations as well as the separate modeling of replicates allowed us to account for inevitable differences between wells, while inferring the key process model parameters from all data. The combination of `calibr8` and `murefi` made it possible to construct a process models of our application example with little code and apply both optimization (MLE) and Bayesian inference (MCMC) without needing to change any implementation details (Code 4, Code 5). Our application example showed that Bayesian inference with ODE-based process models to 28 parallel cultures with hundreds of observations is not only technically feasible, but also accessible without deep understanding of probabilistic programming frameworks.
As implied in the famous quote by George E.P. Box – *"All models are wrong, but some are useful."* – also our Monod kinetics process model does not describe every aspect of the data, but is a powerful tool to quantify key process parameters under uncertainty. From its (in)accuracies, we can gain insight into the bioprocess and generate new hypotheses about the biological process or measurement system that are yet to be understood. In our case, the uncertainty quantification of process model parameters can become the cornerstone of bioprocess development by facilitating robust and intuitive statistical analysis or Bayesian decision-making.

4.3 Comparison with existing modeling software

A multitude of statistical software tools exist, many of which can be used for data analyses similar to the ones presented in this work. The technical complexity of performing such analyses, however, depends strongly on the technical

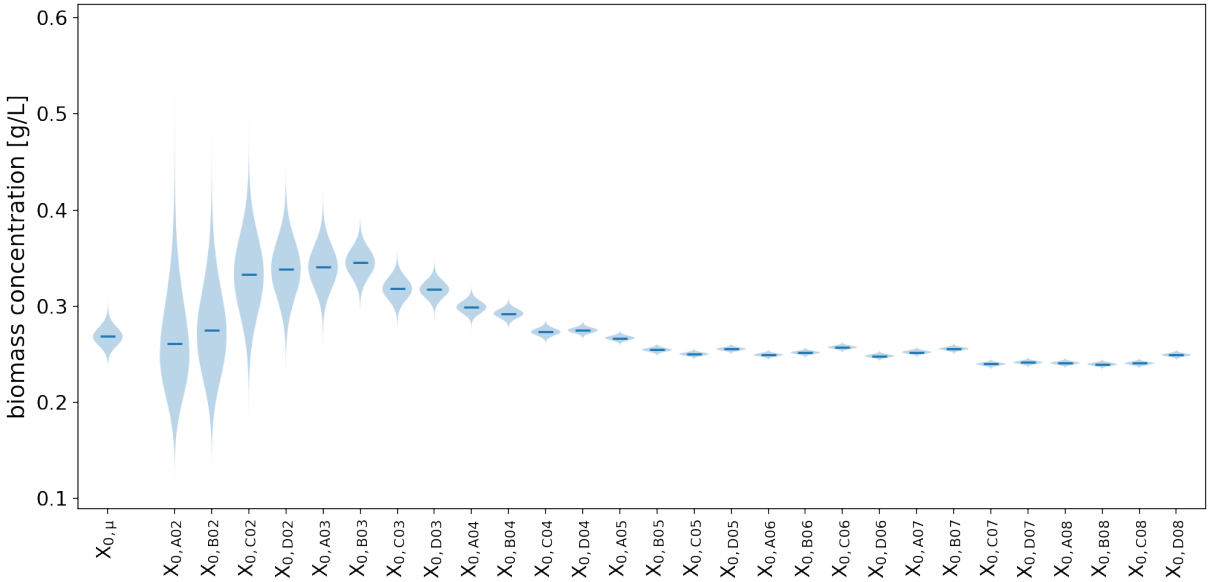

Figure 16: Posterior group mean and well-specific initial biomass concentrations X_0
Variability between the growth curves in separate wells is described by well-specific initial biomass concentrations $X_{0,well}$. Their posterior probability distribution is wide if the well was sacrificed early (left) and narrows down with the number of observed timepoints (right). Their common hyper-prior (a.k.a group mean prior) $X_{0,\mu}$ for the mean of each $X_{0,well}$ was updated to a posterior with $\text{HDI}^{90\,\%}_{X_{0,\mu}} = [0.250,\ 0.288]\ \frac{g}{L}$.

capabilities of the software package. A comparison to relevant packages with similar scope and use-cases is given in Table 1.

For higher-throughput analyses and flexibility in the data analysis workflow, the user interface of statistical analysis software is particularly important. Most tools provide interfaces for popular scripting languages such as Python, R or MATLAB, but the model definition is in some cases delegated to a domain-specific programming language (DSL). For a broad application of calibration models, it is important that they are modular. Software like COPASI considers calibration only in the context of the ODE model and likelihoods cannot be customized. With modeling toolboxes such as Data2Dynamics or PESTO, custom calibration models and likelihoods can be realized, but they must be implemented manually as part of the objective function. This does not only require advanced expertise, but is also more error prone than working with a PPL directly. In contrast, `calibr8` separates calibration modeling entirely from the process modeling workflow, thereby becoming a valuable toolbox for calibration tasks even without process modeling. Together with `murefi`, this modular design allows to seamlessly use custom likelihood models in advanced ODE process models, a feature that we have not found with other software.

An important criterion for usability of calibration software is the required expertise. Packages that implement the foundations of model construction, auto-differentiation and definition of probability densities reduce the mathematical complexity and allow users with little technical expertise to perform advanced statistical analyses. `calibr8` and `murefi` are beginner-friendly, which is also evident from the simplicity of code examples [61, 62] compared to other tools [63, 64].

Bayesian analysis through MCMC methods is available through most modeling packages. Efficient, gradient-based state-of-the-art MCMC algorithms however are only readily available with probabilistic programming languages such as PyMC3 or Stan because they provide the necessary auto-differentiation of models. Finally, experimental replicates or hierarchical structures require replication and nesting of ODE models. Instead of manually expanding the differential equation system to match these requirements, templating approaches as they are used in `murefi` or COPASI can facilitate rapid model construction.

Table 1: Comparison with related software packages
DSL: Domain-Specific Language, GUI: Graphical User Interface

	User interfaces	Modularity of likelihood model	Required expertise	MCMC	ODE model construction	License
murefi, calibr8	Python	Modular	Low	Yes, with auto-diff	Templated	AGPLv3
PyMC3 [46]	Python	Manual	Medium	Yes, with auto-diff	Manual	Apache 2.0
COPASI, PyCoTools3 [16, 17]	GUI, Python	No	Medium	No	Templated	Artistic 2.0, LGPL
Data2Dynamics [14]	MATLAB, DSL	Manual	Medium	Yes	Manual	Not specified
PESTO [15]	MATLAB	Manual	High	Yes	Manual	BSD-3
Stan [48]	DSL	Manual	High	Yes, with auto-diff	Manual	BSD-3
brms [65]	R, Formula-based	Modular	Low	Yes, with auto-diff	N/A	GPLv2
JMP [66]	GUI, HTTP (plugin)	No	Medium	No	N/A	Proprietary

5 Conclusions

In this paper, we introduced the general concept of *calibration models* and presented `calibr8`, an object-oriented Python toolbox that is applicable to both analytical calibration and inference of process models. Our open-source software allows to easily implement and analyze calibration models by providing a number of convenience functions, for example an asymmetric logistic function with an intuitive parametrization and a function to obtain the most important diagnostic plots in one line of code. It thus gives users without a background in statistics access to quantitative linear and non-linear calibration models, as well as Bayesian uncertainty quantification. Furthermore, the implementation through a suite of extendable Python classes allows advanced modelers to customize the technique to a variety of applications. In comparison to existing software, the unique combination of modular likelihood functions from `calibr8` with objectives and (hierarchical) datasets from `murefi` enables a fully Bayesian, Pythonic approach to calibration and process modeling that could so far only be achieved by cumbersome manual implementation or combination of various libraries.

In our work, we demonstrated how the versatile asymmetric logistic calibration model can be applied to bioanalytical calibration tasks. Furthermore, we showed how combining the concept of calibration models with process models allows to gain process insight into a biological process. Especially in combination with `murefi`, our package to set up multi-replicate models, `calibr8` is suitable for high-throughput experimentation because of the flexible interface that allows to analyze data via optimization or MCMC. Uncertainty quantification is covered within the scope of the toolbox and enables easy identification of critical parameters. By making Bayesian inference of ODE models easy to implement, `calibr8` and `murefi` bridge the gap between bioprocess modeling and an entire portfolio of methods, such as Bayesian model comparison or decision-making.

Well-chosen calibration models eradicate the effect of systematic errors in measurements and allow the practitioner to focus a data analysis on the underlying process. In our application example, the non-linear biomass calibration model was required to identify lack-of-fit in the Monod model based on growth behaviour alone. We also identified the biomass overshoot at the beginning of the stationary phase as an interesting target for further investigation, *e.g.* by automated microscopy of cells during cultivation.

`calibr8` greatly reduces the workload of calibration tasks. For example, the systematic, model-based approach allows the user to quantify batch effects between calibration experiments; repetition of calibration measurements could thus

be highly reduced. With `calibr`, we provide a versatile toolbox that we believe to be beneficial not only for the biotechnology community, but for various calibration tasks in experimental sciences.

6 Acknowledgements

First developments of data structures for multi-replicate modeling were made by Michael Osthege in the Theoretical Systems Biology group of Prof. Roland Eils at the German Cancer Research Center under the supervision of Dr. Stefan Kallenberger. The conceptual framework for modular representation of calibration models was devised and implemented by Laura Helleckes and Michael Osthege. Experiments were designed, programmed and conducted by Laura Helleckes and Michael Osthege, as was the data analysis. Eric von Lieres, Marco Oldiges and Wolfgang Wiechert reviewed the manuscript, organized funding and were responsible for supervision and project coordination. This work was funded by the German Federal Ministry of Education and Research (BMBF, Grand. No. 031B0463A) as part of the project "Digitalization In Industrial Biotechnology", DigInBio. Further funding was received from the Enabling Spaces Program "Helmholtz Innovation Labs" of the German Helmholtz Association to support the "Microbial Bioprocess Lab – A Helmholtz Innovation Lab".

References

[1] European Medicines Agency. *Guideline on bioanalytical method validation*. 2015. URL: https://www.ema.europa.eu/en/bioanalytical-method-validation.

[2] U.S. Department of Health et al. *Bioanalytical Method Validation - Guidance for Industry*. 2018. URL: https://www.fda.gov/media/70858/download.

[3] Francisco Raposo. "Evaluation of analytical calibration based on least-squares linear regression for instrumental techniques: A tutorial review". In: *TrAC Trends in Analytical Chemistry* 77 (2016), pp. 167–185.

[4] John W. A. Findlay et al. "Validation of immunoassays for bioanalysis: a pharmaceutical industry perspective". In: *Journal of pharmaceutical and biomedical analysis* 21.6 (2000), pp. 1249–1273.

[5] Binodh DeSilva et al. "Recommendations for the bioanalytical method validation of ligand-binding assays to support pharmacokinetic assessments of macromolecules". In: *Pharmaceutical research* 20.11 (2003), pp. 1885–1900.

[6] Darshana Jani et al. "Recommendations for use and fit-for-purpose validation of biomarker multiplex ligand binding assays in drug development". In: *The AAPS journal* 18.1 (2016), pp. 1–14.

[7] Elizabeth B Cogan, G Bruce Birrell, and O Hayes Griffith. "A robotics-based automated assay for inorganic and organic phosphates". In: *Analytical biochemistry* 271.1 (1999), pp. 29–35.

[8] Simon Unthan et al. "Bioprocess automation on a Mini Pilot Plant enables fast quantitative microbial phenotyping". In: *Microbial cell factories* 14.1 (2015), p. 32.

[9] Andreas Knepper et al. "Robotic platform for parallelized cultivation and monitoring of microbial growth parameters in microwell plates". In: *Journal of laboratory automation* 19.6 (2014), pp. 593–601.

[10] International Bureau of Weights and Measures. *International vocabulary of metrology — Basic and general concepts and associated terms*. 2008. URL: https://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2008.pdf.

[11] Rink Hoekstra et al. "Robust misinterpretation of confidence intervals". In: *Psychonomic bulletin & review* 21.5 (2014), pp. 1157–1164.

[12] Sander Greenland et al. "Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations". In: *European journal of epidemiology* 31.4 (2016), pp. 337–350.

[13] Ward Edwards, Harold Lindman, and Leonard J Savage. "Bayesian statistical inference for psychological research." In: *Psychological review* 70.3 (1963), p. 193.

[14] Andreas Raue et al. "Data2Dynamics: a modeling environment tailored to parameter estimation in dynamical systems". In: *Bioinformatics* 31.21 (2015), pp. 3558–3560.

[15] Paul Stapor et al. "PESTO: parameter estimation toolbox". In: *Bioinformatics* 34.4 (2018), pp. 705–707.

[16] Ciaran M Welsh et al. "PyCoTools: a Python toolbox for COPASI". In: *Bioinformatics* 34.21 (May 2018), pp. 3702–3710. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty409. eprint: https://academic.oup.com/bioinformatics/article-pdf/34/21/3702/26146986/bty409.pdf. URL: https://doi.org/10.1093/bioinformatics/bty409.

[17] Stefan Hoops et al. "COPASI—a complex pathway simulator". In: *Bioinformatics* 22.24 (2006), pp. 3067–3074.

[18] Rens van de Schoot et al. "Bayesian statistics and modelling". In: *Nature Reviews Methods Primers* 1.1 (2021), pp. 1–26.

[19] Fabian Fröhlich, Carolin Loos, and Jan Hasenauer. "Scalable inference of ordinary differential equation models of biochemical processes". In: *Gene Regulatory Networks*. Springer, 2019, pp. 385–422.

[20] Frank Kensy et al. "Validation of a high-throughput fermentation system based on online monitoring of biomass and fluorescence in continuously shaken microtiter plates". In: *Microbial Cell Factories* 8.1 (2009), p. 31.

[21] Shukuo Kinoshita, Kiyoshi Nakayama, and Sadao Akita. "Taxonomical Study of Glutamic Acid Accumulating Bacteria, *Micrococcus glutamicus* nov. sp." In: *Journal of the Agricultural Chemical Society of Japan* 22.3 (1958), pp. 176–185.

[22] Michael Osthege and Laura Helleckes. *JuBiotech/robotools: v1.0.0*. Version v1.0.0. Apr. 2021. DOI: 10.5281/zenodo.4697606. URL: https://doi.org/10.5281/zenodo.4697606.

[23] John Salvatier et al. *pymc-devs/pymc3: PyMC3 3.11.2 (14 March 2021)*. Version v3.11.2. Mar. 2021. DOI: 10.5281/zenodo.4603971. URL: https://doi.org/10.5281/zenodo.4603971.

[24] Ravin Kumar et al. "ArviZ a unified library for exploratory analysis of Bayesian models in Python". In: *Journal of Open Source Software* 4.33 (2019), p. 1143.

[25] Francesco Biscani and Dario Izzo. "A parallel global multiobjective framework for optimization: pagmo". In: *Journal of Open Source Software* 5.53 (2020), p. 2338. DOI: 10.21105/joss.02338. URL: https://doi.org/10.21105/joss.02338.

[26] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[27] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[28] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[29] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[30] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[31] Michael Osthege and Laura Helleckes. *JuBiotech/calibr8: v6.0.0*. Version v5.0.1. Oct. 2020. DOI: 10.5281/zenodo.4127012. URL: https://github.com/JuBiotech/calibr8.

[32] Laura Helleckes and Michael Osthege. *JuBiotech/murefi: v5.0.0*. Version v5.0.0. Mar. 2020. DOI: 10.5281/zenodo.4652910. URL: https://github.com/JuBiotech/murefi.

[33] Paul G. Gottschalk and John R. Dunn. "The five-parameter logistic: A characterization and comparison with the four-parameter logistic". In: *Analytical Biochemistry* 343.1 (2005), pp. 54–65.

[34] Agnieszka Szparaga and Sławomir Kocira. "Generalized logistic functions in modelling emergence of Brassica napus L." In: *PLOS ONE* 13.8 (Aug. 2018), pp. 1–14. DOI: 10.1371/journal.pone.0201980. URL: https://doi.org/10.1371/journal.pone.0201980.

[35] Wikipedia contributors. *Generalised logistic function — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Generalised_logistic_function&oldid=945474789. [Online; accessed 2-April-2020]. 2020.

[36] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[37] Brandon T. Willard et al. *pymc-devs/aesara:* version rel-2.0.7. Apr. 2021. DOI: 10.5281/zenodo.4635498. URL: https://doi.org/10.5281/zenodo.4635498.

[38] Atılım Günes Baydin et al. "Automatic Differentiation in Machine Learning: A Survey". In: *J. Mach. Learn. Res.* 18.1 (Jan. 2017), pp. 5595–5637. ISSN: 1532-4435.

[39] Adrian Seyboldt et al. *aseyboldt/sunode v0.1.2*. Version v0.1.2. Sept. 2020. DOI: 10.5281/zenodo.4058330. URL: https://doi.org/10.5281/zenodo.4058330.

[40] Alan C. Hindmarsh et al. "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers". In: *ACM Trans. Math. Softw.* 31.3 (Sept. 2005), pp. 363–396. ISSN: 0098-3500. DOI: 10.1145/1089014.1089020. URL: https://doi.org/10.1145/1089014.1089020.

[41] The HDF Group. *Hierarchical Data Format, version 5*. http://www.hdfgroup.org/HDF5/. 1997.

[42] Andrew Collette. *Python and HDF5*. O'Reilly, 2013.

[43] Nicholas Metropolis et al. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092. DOI: 10.1063/1.1699114. eprint: https://doi.org/10.1063/1.1699114. URL: https://doi.org/10.1063/1.1699114.

[44] Cajo J. F. ter Braak and Jasper A. Vrugt. "Differential Evolution Markov Chain with snooker updater and fewer chains". In: *Statistics and Computing* 18.4 (Dec. 2008), pp. 435–446. ISSN: 1573-1375. DOI: 10.1007/s11222-008-9104-9. URL: https://doi.org/10.1007/s11222-008-9104-9.

[45] Matthew D. Hoffman and Andrew Gelman. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: *Journal of Machine Learning Research* 15.47 (2014), pp. 1593–1623. URL: http://jmlr.org/papers/v15/hoffman14a.html.

[46] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2 (Apr. 2016), e55. DOI: 10.7717/peerj-cs.55. URL: https://doi.org/10.7717/peerj-cs.55.

[47] Eli Bingham et al. "Pyro: Deep Universal Probabilistic Programming". In: *Journal of Machine Learning Research* (2018).

[48] Bob Carpenter et al. "Stan: A Probabilistic Programming Language". In: *Journal of Statistical Software, Articles* 76.1 (2017), pp. 1–32. ISSN: 1548-7660. DOI: 10.18637/jss.v076.i01. URL: https://www.jstatsoft.org/v076/i01.

[49] Joshua V. Dillon et al. *TensorFlow Distributions*. 2017. arXiv: 1711.10604 [cs.LG].

[50] Cajo J. F. Ter Braak. "A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces". In: *Statistics and Computing* 16.3 (Sept. 2006), pp. 239–249. ISSN: 1573-1375. DOI: 10.1007/s11222-006-8769-1. URL: https://doi.org/10.1007/s11222-006-8769-1.

[51] Iain Murray, Ryan Adams, and David MacKay. "Elliptical slice sampling". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 541–548. URL: http://proceedings.mlr.press/v9/murray10a.html.

[52] John W. A. Findlay and Robert F. Dillard. "Appropriate calibration curve fitting in ligand binding assays". In: *The AAPS journal* 9.2 (2007), E260–E267.

[53] Mitra Azadeh et al. "Calibration curves in quantitative ligand binding assays: recommendations and best practices for preparation, design, and editing of calibration curves". In: *The AAPS journal* 20.1 (2018), p. 22.

[54] Johannes Hemmerich et al. "Microbioreactor Systems for Accelerated Bioprocess Development". In: *Biotechnology Journal* 13.4 (2018), p. 1700141. DOI: https://doi.org/10.1002/biot.201700141. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/biot.201700141. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/biot.201700141.

[55] Lothar Eggeling and Michael Bott. *Handbook of Corynebacterium glutamicum*. CRC press, 2005.

[56] Jacques Monod. "The Growth of Bacterial Cultures". In: *Annual Review of Microbiology* 3.1 (1949), pp. 371–394. DOI: 10.1146/annurev.mi.03.100149.002103. eprint: https://doi.org/10.1146/annurev.mi.03.100149.002103. URL: https://doi.org/10.1146/annurev.mi.03.100149.002103.

[57] Heinrich Senn et al. "The growth of Escherichia coli in glucose-limited chemostat cultures: a re-examination of the kinetics". In: *Biochimica et Biophysica Acta (BBA) - General Subjects* 1201.3 (1994), pp. 424–436. ISSN: 0304-4165. DOI: https://doi.org/10.1016/0304-4165(94)90072-8. URL: http://www.sciencedirect.com/science/article/pii/0304416594900728.

[58] Johannes Hemmerich et al. "Less Sacrifice, More Insight: Repeated Low-Volume Sampling of Microbioreactor Cultivations Enables Accelerated Deep Phenotyping of Microbial Strain Libraries". In: *Biotechnology Journal* 14.9 (2019), p. 1800428. DOI: 10.1002/biot.201800428. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/biot.201800428.

[59] Simon Unthan et al. "Beyond growth rate 0.6: What drives Corynebacterium glutamicum to higher growth rates in defined medium". In: *Biotechnology and bioengineering* 111.2 (2014), pp. 359–371.

[60] Michael Betancourt. *Hierarchical Modeling*. 2020. URL: https://betanalpha.github.io/assets/case_studies/hierarchical_modeling.html.

[61] *calibr8 Documentation*. URL: https://calibr8.readthedocs.io.

[62] *murefi Documentation*. URL: https://murefi.readthedocs.io.

[63] *PyCoTools Documentation*. URL: https://pycotools3.readthedocs.io.

[64] *d2d Examples*. URL: https://github.com/Data2Dynamics/d2d/tree/master/arFramework3/Examples.

[65] Paul-Christian Bürkner. "brms: An R package for Bayesian multilevel models using Stan". In: *Journal of statistical software* 80.1 (2017), pp. 1–28.

[66] SAS Institute. *JMP*. URL: https://www.jmp.com.

A Appendix

A.1 Reparametrization of asymmetric logistic function

For simplicity of the reparameterization, the asymptote parameters L_L and L_U of the original *Richard's Curve* (11) can be omitted and substitutions $b = -B$ and $1/v = -e^{-c}$ were made such that all parameters may be real numbers (17). In (18) the symmetry is already parametrized exactly as in the final result (12): As it increases, the inflection point I_y moves towards the upper limit. At $c = 0$ it lies centered between the limits.

$$\begin{aligned} f(x) &= \frac{1}{(1+e^{-b(a-x)})^{e^{-c}}} \\ a, b, c &\in \mathcal{R} \end{aligned} \tag{17}$$

In the following steps a and b are reparametrized in terms of the x-coordinate of the inflection point I_x and the slope S at the inflection point respectively. I_x was obtained by solving the second derivative of the a, b, c parametrization (17) for a (18):

$$\begin{aligned} & f''(I_x) = 0 \\ \Leftrightarrow \quad & I_x = a - \frac{c}{b} \\ \Leftrightarrow \quad & a = I_x + \frac{c}{b} \end{aligned} \tag{18}$$

The slope parameter was obtained by substituting x in the first derivative of the a, b, c parametrization (17) with the analytical solution for I_x from (18).

$$\begin{aligned} & S = f''(I_x) \\ \Leftrightarrow \quad & S = b(e^c+1)^{-1-e^{-c}} \end{aligned} \tag{19}$$

Substituting a in (17) with $a(I_x, b, c)$ from (18) yields a parametrization in terms of I_x, b, c (20):

$$\begin{aligned} f(x) &= (e^{b(I_x - x + \frac{c}{b})+1})^{-e^{-c}} \\ I_x, b, c &\in \mathcal{R} \end{aligned} \tag{20}$$

For a parametrization in terms of both I_x and S, their equations from (18) and (19) must be solved for a and b:

$$\begin{aligned} a &= \frac{I_x e^c}{e^c+1} + \frac{I_x}{e^c+1} + \frac{c(e^c+1)^{-1-e^{-c}}}{S} \\ b &= S(e^c+1)^{(e^c+1)e^{-c}} \end{aligned} \tag{21}$$

A parametrization in terms of I_x, S, c is then obtained by substitution of a an b in (17):

$$\begin{aligned} f(x) &= (e^{(e^c+1)^{(e^c+1)e^{-c}} \cdot (I_x S - Sx + c(e^c+1)^{-(e^c+1)e^c})} + 1)^{-e^{-c}} \\ I_x, S, c &\in \mathcal{R} \end{aligned} \tag{22}$$

By common subexpression elimination (22) simplifies to (23).

$$\begin{aligned} f(x) &= (e^{x_2 \cdot (I_x S - Sx + \frac{c}{x_2})} + 1)^{x_1} \\ x_0 &= e^c + 1 \\ x_1 &= e^{-c} \\ x_2 &= x_0^{x_0 \cdot x_1} \\ I_x, S, c &\in \mathcal{R} \end{aligned} \tag{23}$$

The final generalized parametrization (12) in terms of L_L, L_U, I_x, S, c was obtained by scaling slope parameter and function value with $L_U - L_L$ and shifting by L_U. The corresponding Python implementation is shown in Code 6. For a step by step derivation of (12), as well as its inverse using `sympy` we refer to the "Background Asymmetric Logsitc" Jupyter notebook in the `calibr8` repository [31].

Code 6: Implementation of reparameterized asymmetric logistic function

```
def asymmetric_logistic(x, theta):
    """5-parameter asymmetric logistic model.

    Parameters
    ----------
    x : array-like
        independent variable
    theta : array-like
        parameters of the logistic model
            L_L: lower asymptote
            L_U: upper asymptote
            I_x: x-value at inflection point
            S: slope at the inflection point
            c: symmetry parameter (0 is symmetric)

    Returns
    -------
    y : array-like
        dependent variable
    """
    L_L, L_U, I_x, S, c = theta[:5]
    # common subexpressions
    s0 = numpy.exp(c) + 1
    s1 = numpy.exp(-c)
    s2 = s0 ** (s0 * s1)
    # re-scale the inflection point slope with the interval
    s3 = S / (L_U - L_L)

    x = numpy.array(x)
    y = (numpy.exp(s2 * (s3 * (I_x - x) + c / s2)) + 1) ** -s1
    return L_L + (L_U-L_L) * y
```

A.2 Implementation, planning and saving of calibrations

Code 7: Convenience class `BaseAsymmetricLogisticT`

```python
class BaseAsymmetricLogisticT(BaseModelT):
    def __init__(
        self, *,
        independent_key:str, dependent_key:str,
        scale_degree:int=0,
        theta_names: Optional[Tuple[str]]=None,
    ):
        """ Template for a model with asymmetric logistic trend (mu)
        and polynomial scale (as a function of mu).

        Parameters
        ----------
        independent_key : str
            name of the independent variable
        dependent_key : str
            name of the dependent variable
        scale_degree : optional, int
            degree of the polynomial model describing the scale as a function of mu
        theta_names : optional, tuple of str
            may be used to set the names of the model parameters
        """
        self.scale_degree = scale_degree
        if theta_names is None:
            theta_names = tuple('L_L,L_U,I_x,S,c'.split(',')) + tuple(
                f'scale_{d}'
                for d in range(scale_degree + 1)
            ) + ('df',)
        super().__init__(independent_key, dependent_key, theta_names=theta_names)
```

Code 8: Human-readable pipetting instructions for the serial dilution of biomass for the calibration experiment

```
Serial dilution plan (0.00102 to 1.00) from at least 12232.0 µL stock and 54368.0 µL diluent:
  Prepare column 1 with [2000. 1727. 1491. 1287. 1111.  959.] µL from stock and fill up to 2000 µL
  Prepare column 2 with [538. 465. 401. 346. 299. 258.] µL from stock and fill up to 1300 µL
  Prepare column 3 with [223. 192. 166. 143. 124. 107.] µL from stock and fill up to 1300 µL
  Prepare column 4 with [92. 80. 69. 59. 51. 44.] µL from stock and fill up to 1300 µL
  Prepare column 5 with [39. 39. 39. 39. 39. 39.] µL from column 1 and fill up to 1300 µL (1 serial dilutions)
  Prepare column 6 with [39. 39. 39. 39. 39. 39.] µL from column 2 and fill up to 1300 µL (1 serial dilutions)
  Prepare column 7 with [39. 39. 39. 39. 39. 39.] µL from column 3 and fill up to 1300 µL (1 serial dilutions)
  Prepare column 8 with [39. 39. 39. 39. 39. 39.] µL from column 4 and fill up to 1300 µL (1 serial dilutions)
```

Code 9: JSON file containing stored model properties.

```json
{
    "calibr8_version": "6.0.0",
    "model_type": "models.LogisticGlucoseCalibrationModelV1",
    "theta_names": [
        "L_L", "L_U", "I_x", "S", "c", "scale_0", "scale_1", "df"
    ],
    "theta_bounds": [
        [-Infinity, 0.3],
        [2, 5],
        [-50, 50],
        [0, 20],
        [-3, 3],
        [0, 0.1],
        [0, 0.06],
        [1, 20]
    ],
    "theta_guess": [0.1, 2.8, 1.2, 10, 1, 0.08, 0.01, 3],
    "theta_fitted": [
        -8.812, 2.765, 8.246, 0.0839,
```

```
            2.69, 0.000374, 0.0154, 3.007
        ],
        "theta_timestamp": "2021-02-15T14:27:11Z",
        "independent_key": "glc",
        "dependent_key": "A365",
        "cal_independent": [
            50.0,
            27.94736842105263,
            ...
            0.05030330952033825
        ],
        "cal_dependent": [
            2.6449,
            2.2389,
            ...
            0.1166
        ]
    }
```

A.3 Process model parametrization, parameter estimates and MCMC results

The parametrization of the batch cultivation process model was given by the tabular notation of a `murefi.ParameterMapping` Table 2. Maximum likelihood estimates, posterior sample means, standard deviation, HDI interval and $\hat{R}$ statistic are shown in Appendix A.3. Two-dimensional kernel densities of all posterior samples are shown in Figure 17.

Table 2: Parameter mapping for fitting of Monod kinetics
Repetitive rows were left out for clarity. The full length table has 28 rows.

replicate	S_0	X_0	μ_{max}	K_S	Y_{XS}
A02	`S0`	`X0_A02`	`mu_max`	0.02	`Y_XS`
...	`S0`	`X0_A0.`	`mu_max`	0.02	`Y_XS`
A08	`S0`	`X0_A04`	`mu_max`	0.02	`Y_XS`
...	`S0`	`X0_...`	`mu_max`	0.02	`Y_XS`
D08	`S0`	`X0_D08`	`mu_max`	0.02	`Y_XS`

Table 3: Parameter estimates from MLE and MCMC

	MLE	mean	sd	hdi_5%	hdi_95%	r_hat
S0	16.92	15.23	0.103	15.06	15.40	1.0
mu_max	0.425	0.425	0.001	0.424	0.426	1.0
Y_XS	0.673	0.747	0.005	0.739	0.756	1.0
X0_mu	-	0.269	0.010	0.252	0.285	1.0
X0_A02	0.231	0.266	0.053	0.179	0.348	1.0
X0_A03	0.335	0.341	0.022	0.304	0.377	1.0
X0_A04	0.301	0.299	0.006	0.289	0.310	1.0
X0_A05	0.267	0.267	0.002	0.263	0.271	1.0
X0_A06	0.250	0.250	0.002	0.247	0.252	1.0
X0_A07	0.252	0.252	0.002	0.250	0.255	1.0
X0_A08	0.241	0.241	0.002	0.238	0.244	1.0
X0_B02	0.297	0.277	0.047	0.200	0.354	1.0
X0_B03	0.356	0.345	0.015	0.321	0.369	1.0
X0_B04	0.291	0.292	0.005	0.285	0.300	1.0
X0_B05	0.256	0.255	0.002	0.252	0.258	1.0
X0_B06	0.252	0.252	0.002	0.249	0.255	1.0
X0_B07	0.256	0.256	0.002	0.253	0.259	1.0
X0_B08	0.240	0.240	0.002	0.237	0.242	1.0
X0_C02	0.416	0.333	0.041	0.266	0.400	1.0
X0_C03	0.314	0.318	0.012	0.299	0.337	1.0
X0_C04	0.273	0.273	0.004	0.267	0.279	1.0
X0_C05	0.251	0.250	0.002	0.248	0.253	1.0
X0_C06	0.257	0.257	0.002	0.255	0.260	1.0
X0_C07	0.240	0.240	0.002	0.237	0.243	1.0
X0_C08	0.241	0.241	0.002	0.238	0.243	1.0
X0_D02	0.380	0.338	0.030	0.289	0.388	1.0
X0_D03	0.315	0.317	0.009	0.303	0.332	1.0
X0_D04	0.275	0.275	0.003	0.270	0.280	1.0
X0_D05	0.256	0.256	0.002	0.253	0.258	1.0
X0_D06	0.248	0.248	0.002	0.246	0.251	1.0
X0_D07	0.241	0.242	0.002	0.239	0.244	1.0
X0_D08	0.250	0.249	0.002	0.247	0.252	1.0

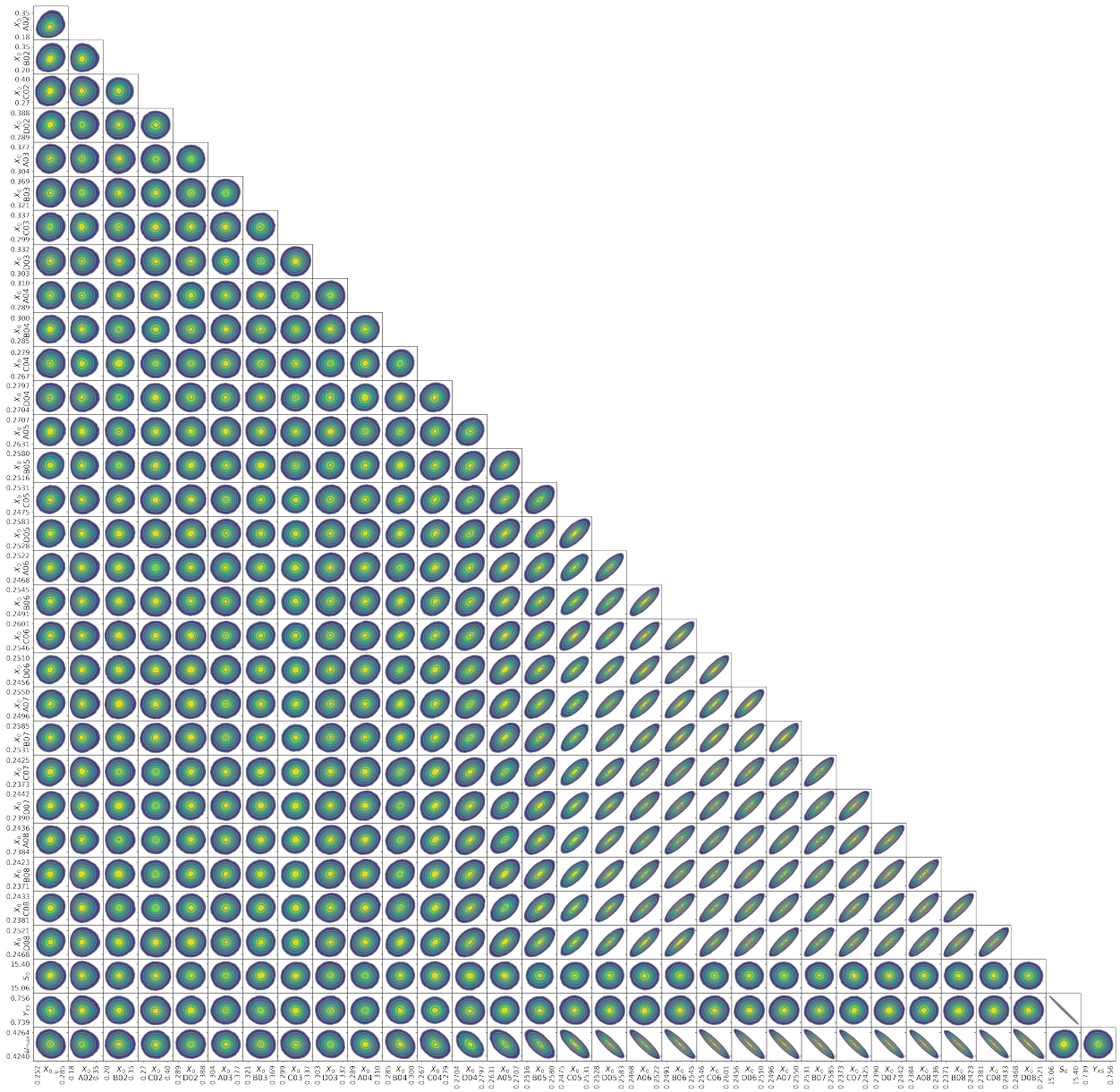

Figure 17: Pair plot of marginal posterior distribution
Axis labels mark the 90 % HDI and subplot axis limits are set at the 98 % HDI. Units are h^{-1} for $\mu_{\max}$, $\frac{g_{\text{glucose}}}{g_{\text{biomass}}}$ for Y_{XS} and $\frac{g}{L}$ for S_0 and X_0.