Supporting Information - A Bayesian approach to extracting kinetic information from enzymatic reaction networks

Mathieu G. Baltussen[1] Jeroen van de Wiel[1] Cristina Lía Fernández Regueiro[1]
Miglė Jakštaitė[1] Wilhelm T.S. Huck[1,*]

[1] Institute for Molecules and Materials, Radboud University Nijmegen, Nijmegen, the Netherlands;

[*] Correspondence: Wilhelm T.S. Huck <w.huck@science.ru.nl>

Contents

Materials and Instrumentation

Materials

All chemicals and reagents were used as received from commercial suppliers without any further treatment unless stated otherwise.

Enzymes

The enzymes immobilized on PEBs were purchased from Merck Sigma-Aldrich. Specifically:

- Lyophilized Trypsin (Tr) from bovine pancreas (product no. T1426)
- Lyophilized Glucose Dehydrogenase (GDH) from Pseudomonas sp. (product no. 19359)
- Lyophilized Hexokinase, type F-300 (HK) from Saccharomyces cerevisiae (product no. H4502)
- Lyophilized Glucose-6-phosphate dehydrogenase (G6P-DH) from Leuconostoc menesteroides (product no. G8529)

Trypsin reactions was conducted in a 200 mM Tris buffer, pH 7.8, with 20 mM of CaCl2. GDH, HK and G6P-DH reactions were conducted in a 200 mM Tris buffer, pH 7.8, with 10 mM of MgCl2.

Substrates

The substrates Cbz-Arg-AMC (R-AMC, CAS: 3701-04-6, New catnr: 4002540.0050) and Suc-Ala-Ala-Ala-AMC (AAA-AMC, CAS 73617-90-0, New catnr: 4006305.0050) were purchased from BioConnect B.V. and were dissolved as 150 mM stock in anhydrous DMSO and kept frozen at -20°C.

The substrates employed for all metabolic enzymatic reactions were:

- Glucose (anhydrous) (Merck product no. 1083370250)
- glucose-6-phosphate hydrate (Sigma Aldrich product no. G7250)
- ATP disodium salt (Sigma Aldrich product no. A26209)
- ADP monosodium salt (Sigma Aldrich product no. A2754)
- NAD+ (Roche product no. 10127965001)
- NADH disodium salt (Roche product no. 10107735001)
- NADP+disodium salt (Roche product no. 10128031001)
- NADPH tetrasodium salt (Roche product no. 1010724001)

Instrumentation & Quantification protocols

Flow setup

A custom made CSTR (see Figure 1 for the design schematic, effective volume $100\mu L$) was charged with PEBs (in a ratio of 1:31 mg beads:injection volume). The openings of the reactor were sealed with Whatman Nuclepore Track-Etch polycarbonate membranes (5 μm poresize, cat. number 10417414) to prevent outflow of PEBs. To subject the CSTR to various flow conditions we used Cetoni Low-Pressure High-Precision Syringe Pumps neMESYS 290N and gastight Hamilton syringes, with all flowrates programmed using the Cetoni neMESYS software. For all methods of offline detection, the outflow of the CSTR was connected to a BioRad 2210 Fraction collector, collecting for 7.5 (absorbance) or 10 minutes (HPLC) per fraction. This experimental setup is shown in Figure 2a-b

Spectroscopy

Offline fluorescence Collected fractions were pipetted onto a microplate (Greiner Bio One, black, polystyrene, 96 flat bottom chimney wells) at 60 μL per well. Fluorescence measurements for enzyme activity determination were performed with a Tecan Spark 10M plate reader. The fluorescence intensity of wells containing 30-200 μL of the reaction mixture was monitored for 1-4 min (shaking 3s/orbital mode/amplitude 4mm) at 23 °C using top or bottom reading mode, at $\lambda_{ex}/\lambda_e m$ = 380nm/460nm for 7-amino-4-methylcoumarin (AMC) based substrates.

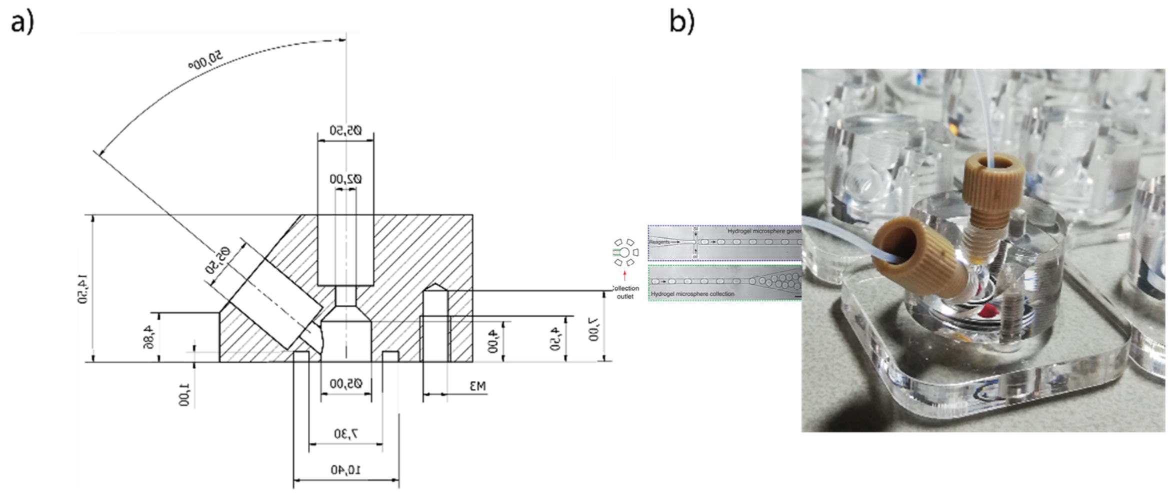

a)
b)
Hydrogel microsphere generation
Reagents
Hydrogel microsphere collection
Collection outlet

Figure 1: a) Schematic overview of the CSTR. b) Photograph of the CSTR

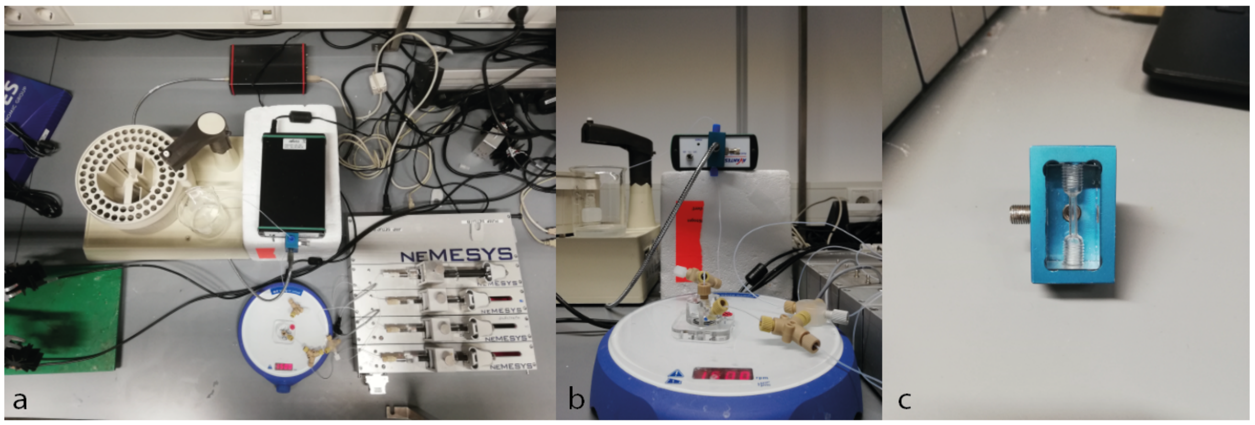


Figure 2: a) Overview of the experimental setup, with left the fraction collection, lower center the CSTR on a stirring plate, and lower right the neMESYS pump system. b) Close-up of the CSTR connected to input and output tubing. c) Close-up of the flow cuvette used for online absorbance detection, provided by LabM8

Offline absorbance Collected fractions were pipetted onto a microplate (Greiner Bio One, transparent, polystyrene, 96 flat bottom chimney wells) at 60 μL per well. The absorbance of each sample at 340 nm was measured in a TECAN SPARK M10 platereader. Using a calibration curve, the absorbance data was converted to NADH concentration.

Online absorbance Absorbance in flow experiments was continuously measured at the reactor's output with a custom made flow cell kindly provided to us by LabM8 (shown in Figure 2c), connected to an AvaLight 355 nm LED lamp. Absorbance at 380 nm was detected using an AvaSpec-2048 with 1,005 ms integration time and averaging at 200 times per recorded datapoint. Using a calibration curve, the absorbance data was converted to NADH concentration.

High-performance liquid chromatography

High-performance Liquid Chromatography (HPLC) was performed using Shimadzu NexeraX3/Prominence system under a 0.9 mL/min flow at 45 °C with a Shimadzu WAX-1 column. For the Shimadzu system, an injection volume was subjected to a 25 min gradient program was used starting from 20 mM of potassium phosphate at pH 7.0

- 1 min – 20 mM at pH 7.0
- 16 min – 480 mM at pH 6.8
- 19 min – 480 mM at pH 6.8
- 22 min – 20 mM at pH 7.0
- 25 min – 20mM at pH 7.0

Retention time of ADP is at 8.8 minutes, ATP at 15.5 minutes.

G6P-DH assay

Collected fractions were pipetted onto a microplate (Greiner Bio One, transparent, polystyrene, 96 flat bottom chimney wells) at 60 μL per well. To these wells was added an 20 μL of 5 mM of NADP+ and 1 μL of 500 u/mL of G6P-DH. The absorbance of each sample at 340 nm was continuously measured in a TECAN SPARK M10 platereader. Using a calibration curve, the absorbance data was converted to NADPH concentration. We assumed this NADPH concentration equivalent to G6P concentration.

Production and characterisation of polyacrylamide-enzyme beads

Microfluidic devices fabrication for beads production

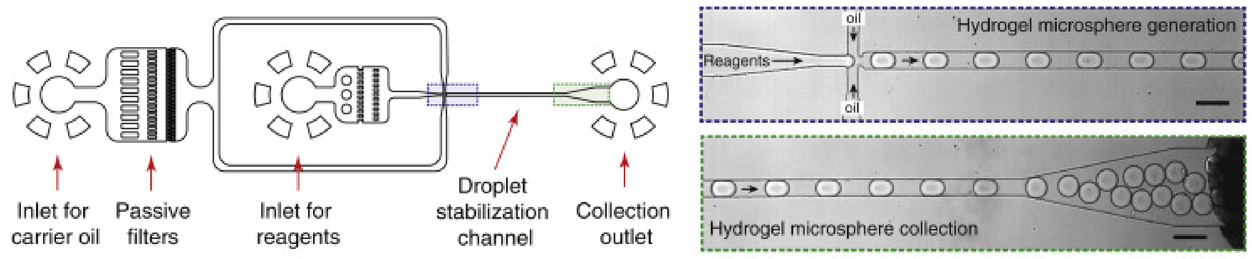

Inlet for carrier oil
Passive filters
Inlet for reagents
Droplet stabilization channel
Collection outlet
oil
Reagents
oil
Hydrogel microsphere generation
Hydrogel microsphere collection

Figure 3: Schematic of the microfluidic device for droplets production (*adaptated from*[1])

Microfluidic devices, designed according to the scheme shown in Figure 3[1], were produced in silicon wafers by photo and soft lithography, with different orifice sizes (20-80μm) at the T-junction, for droplet formation. After that, the wafers were used for the production of polydimethylsiloxane (PDMS) devices. The PDMS replicas were separated from the wafer, and inlets and outlets of 1mm inner diameter were punched. After the replica was bonded by oxygen plasma treatment to a glass slide, channels were coated with 2% 1H,1H,2H,2H-Perfluoro-octlytriethoxysilane to make them hydrophobic[2].

Method 1: Enzyme-first functionalization

This method was used for the synthesis of PEBs containing Trypsin.

Synthesis 6-acrylaminohexanoic acid succinate (AAH-Suc) linker

N-6-acryloyl amido hexanoic acid (30 mg, 0.172 mmol, 1 eq) and N-Hydroxy succinimide (20mg, 0.172 mmol, 1eq) were added in a large glass vial and brought under N2 atmosphere. Subsequently, they were dissolved in 1 mL of anhydrous DMF and cooled down to 0°C. Then, N, N'-dicyclohexylcarbodiimide (DCC, 38mg, 0.172mmol, 1eq) was added, and the vial was sealed and left to stir at 0°C for one hour. Subsequently, the reaction was brought to 4°C and stirred for 16h. The reaction mixture was assumed quantitative and immediately used for enzyme functionalisation without further analysis[3].

Enzyme functionalization

The desired enzyme (1 eq) was added to a falcon tube containing 0.1 M NaHCO3 (4mL), followed by addition of AAH-Suc (7.5 eq) in DMF. The mixture was left to stir at 21 °C for 1 h, after which the reaction mixture was dialysed and lyophilised.

General Enzyme Immobilization During Polymerization (EIDP) method

An emulsion of monodisperse water-in-oil droplets was produced by using the microfluidic device described above with a 20 μm-wide T-junction, following an adapted procedure from Rivello et al.[3] The reagents phase used for hydrogel formation was composed by 9.7% (w/v) acrylamide, 0.4% (w/v) bisacrylamide, 1.5% (w/v) 2,2'Azobis(2-methylpropionamidine) dihydrochloride (AAPH) and 0-8% (v/v) of a solution of functionalized enzyme (100 μM). The oil phase was composed of fluorinated fluid HFE-7500 (3M) and 1.5% (v/v) Pico-Surf™ 1. The flow rates used to produce monodisperse beads of 50μm average diameter were 600$\mu L/h$ (Qw) for the reagents phase and 900$\mu L/h$ for the oil phase (Qo). The emulsion of droplets created in the microfluidic device was collected in an Eppendorf with 100 uL mineral oil to avoid evaporation and breaking the emulsion and polymerised for 10 minutes under UV light. After polymerisation, beads were washed three times with 20% (v/v) 1H,1H,2H,2H perfluorooctanoic (PFO) in HFE-7500 oil to break the emulsion. The obtained beads were then washed three times with 1% (v/v) Span 80 in heptane, three times with 0.1% (v/v) Triton X-100 in miliQ and three times with Milli-Q. After every washing step, the beads were centrifugated 30 seconds at 5000 rcf, and the supernatant was removed by pipetting. The resulting beads were freeze-dried and re-dissolved in miliQ at a concentration of 0.0322mg/μL. Bead size was obtained after freeze-drying by imaging with a light microscope with a 40x objective lens. The average bead size was 50 μm.

Method 2:

This method was used for the synthesis of PEBs containing GDH, HK, and G6PDH.

Empty bead production method

The microfluidics device described above was used to produce gel beads with a 20 μm wide T-junction, following an adapted procedure from[2]. The gel solution phase consisted of 9.6% (w/v) acrylamide, 0.4% (w/v) N,N'-methylenebisacrylamide, 0.5% (w/v) acrylic acid and 1.5% (w/v) 2,2'Azobis(2-methylpropionamidine) dihydrochloride. The oil phase contained 1.5% (v/v) Pico-Surf™ 1 in fluorinated fluid HFE-7500 (3M). The flow rates for gel phase and oil phase were 600 $\mu L/h$ and 900 $\mu L/h$, respectively. The outflow emulsion was collected in the tube which was filled with 100 μL mineral oil. Afterwards beads were polymerised using UV lamp for 10 minutes at 70% gain. After polymerisation the lowest layer containing fluorocarbon phase was carefully removed with a P200 pipette. The remaining beads were washed 3 times with 20% (v/v) 1H,1H,2H,2H-Perfluoro-1-octanol in HFE-7500 (3M), then 3 times with 1% (v/v) Span 80 in hexane, 3 times with 0.1% (v/v) Triton X-100 in Milli-Q and finally 3 times with Milli-Q. Every washing step was finalised with mixing the tube using vortex, centrifuging at 5000 x g for 3 min and removing the layer which was not containing beads. Furthermore, beads were flash frozen using nitrogen and freeze dried overnight. After re-wetting beads, their size was determined using light microscope with 40x magnitude objective. The average size was 50 μm in diameter.

General Enzyme Immobilisation after Polymerisation (EIAP) Method

Empty acrylamide beads were re-dissolved in Milli-Q at a concentration of 0.0322 mg/μL. 1-(3-Dimethylaminopropyl)-3-ethylcarbodiimide hydrochloride (100 mM) and N-Hydroxysuccinimide (100 mM) were added to the reaction mixture. The total volume of the activation solution was 5-fold of the beads volume. The reaction mixture tube was put on the roller bank for 30 min. After this, beads were centrifuged at 5000 x g for 3 min. The supernatant was carefully removed using P200 pipette. Beads were washed 3 times by adding Milli-Q, mixing using vortex, centrifuging and removing the supernatant. Specific to each batch of PEBs, a certain amount of enzyme was added to the beads (see Table 1). The tube was put on the roller bank for 2 h coupling step. Sequentially, beads were washed 8 times by adding Milli-Q, centrifuging and removing the supernatant. Finally, beads were flash frozen using nitrogen and freeze dried overnight.

Overview of PEBs

Table 1: Overview of PEB batches

Enzyme	Batch number	Enzyme concentration added	Volume per mg of beads
GDH	1	0.54 kUnit/mL	100 μL/mg
	2	2 kUnit/mL	100 μL/mg
	3	1 kUnit/mL	100 μL/mg
HK	1	1 kUnit/mL	100 μL/mg
G6P-DH	1	1 kUnit/mL	100 μL/mg

Overview of experiments

Processed datafiles from these experiments can be found on the accompanying github repository in the `data` folder as csv-files. The files are directly used during analysis performed in the Jupyter notebooks.

In the tables below, the volume refers to the volume of PEBs suspension injected into the CSTR, where the PEBs suspension itself is created by suspending 1 mg of dry PEBs in 31 μL buffer, or other quantities in the same ratio (1:31). The batch refers to the batch of PEBs, which can contain different effective enzyme concentation. See Table 1 for an overview of all PEB batches.

Table 2: Overview of single-enzyme experiments

Code	flowrate ($\mu L/h$)	enzyme	volume (μL)	batch	inputs	observables	observation techniques
SNCA14	750	GDH	10	1	G, NAD	NADH	offline abs.
SNCA15	750	GDH	10	1	G, NAD	NADH	offline abs.
SNCA17	750	HK	10	1	G, ATP	G6P	G6P assay
SNCA18	750	HK	1	1	G, ATP	G6P	G6P assay
SNKS03	750	HK	1	1	G, ATP	G6P	G6P assay
SNKS04	750	HK	1	1	G, ATP	G6P,ATP	G6P assay, HPLC
SNKS08	750	G6PDH	10	1	G6P, NAD	NADH	offline abs.
SNKS11	750	GDH	0.5	2	G, NAD	NADH	online abs.
SNKS12	750	GDH	0.5	2	G, NAD	NADH	online abs.
SNKS18	750	GDH	2.0	3	G, NAD	NADH	offline abs.
SNKS20	750	G6PDH	2.0	1	G6P, NAD	NADH	online abs.

Table 3: Overview of multi-enzyme experiments

Code	flowrate ($\mu L/h$)	enzymes	volumes (μL)	batches	inputs	observables	observation techniques
SNKS06	750	GDH,HK	10.0, 1.0	1, 1	G, NAD, ATP	NADH, ADP	online abs., HPLC
SNNS002	750	GDH,HK	0.5, 1.0	2, 1	G, NAD, ATP	NADH	online abs.
SNNS003	750	GDH,HK	0.25, 1.5	2, 1	G, NAD, ATP	NADH	online abs.
SNNS004	750	GDH,HK	0.333, 1.33	2, 1	G, NAD, ATP	NADH	online abs.
SNNS005	750	GDH,HK	0.666, 0.666	2, 1	G, NAD, ATP	NADH	online abs.
SNNS006	750	GDH,HK	0.75, 0.5	2, 1	G, NAD, ATP	NADH	online abs.
SNNS007	750	GDH,HK	0.5, 1.0	2, 1	G, NAD, ATP	NADH	online abs.

Overview of computational methods

Python scripts and Jupyter notebooks were used to create the Bayesian models and perform inference and predictive sampling. All computational studies were performed with Jupyter notebooks. Datasets were loaded in from csv-files with Pandas, and if relevant, concatenated together into larger objects.

The Bayesian model, including the determination of prior probabilities and likelihood function, was created using PyMC3[4]. Generally, prior probabilities for Michaelis-Menten parameters were chosen as uniform distributions over a specified interval. These distributions were used as uninformative priors to ensure no subjective information would enter the model, while garanteeing correct sampling and estimation of parameters. Priors for the uncertainty estimations (denoted by `sigma` in the notebooks) were given an exponential distribution, which also acted as an uninformative distribution, while garanteeing correct sampling and convergence. In larger models, where multiple likelihoods were combined, hyperpriors were placed on the k_{cat} and σ priors to increase convergence of the sampling algorithm.

All sampling was peformed using the No-U-Turn Sampler (NUTS)[5], which is an adaptive step-size Hamiltonian Monte Carlo sampler. When (automatically, or via a custom operator) the gradients of the likelihoods with respect to the kinetic parameters were given, this sampling method is much more efficient then a classical Metropolis Monte Carlo sampler, showing faster convergence and requiring less samples for precise posterior estimations. Generally, sampling was performed using 4 or 8 independent chains on 4 or 8 cpu cores, all with 1000 tuning steps, and 1000 sampling steps, and a target step acceptance probability of 0.95. These values were found to yield good sampling results without becoming computatially inefficient.

For likelihood calculations of partially observable ERNs, a custom steady-state likelihood operator was written in Theano, according to description given in the manuscript. This operator is freely available from the public github repository at https://github.com/mgbaltussen/BayERN.

The samples obtained from the posterior distribution were further analysed using standard statistical tools in Python, the Numeric Python package and the Scientific Python package (NumPy and Scipy)[6]. To ensure the accessibility and reproducibility of these results the datasets and Jupyter notebooks, used for the analysis and creation of figures found in the publication, are made available as additional Supporting Information, and directly on github at https://github.com/huckgroup/Bayesian-enzymatic-networks_manuscript_2022. This repository also includes version information for all software dependencies.

PDF reproductions of example notebooks with explanation (`example_simple.ipynb` and `example_complex.ipynb`) are also included in the appendix of this supporting information.

Extended iterative combination of experiments

See Figure 4 for an extended version of the figure, showing posterior updating per iteration of new experiment.

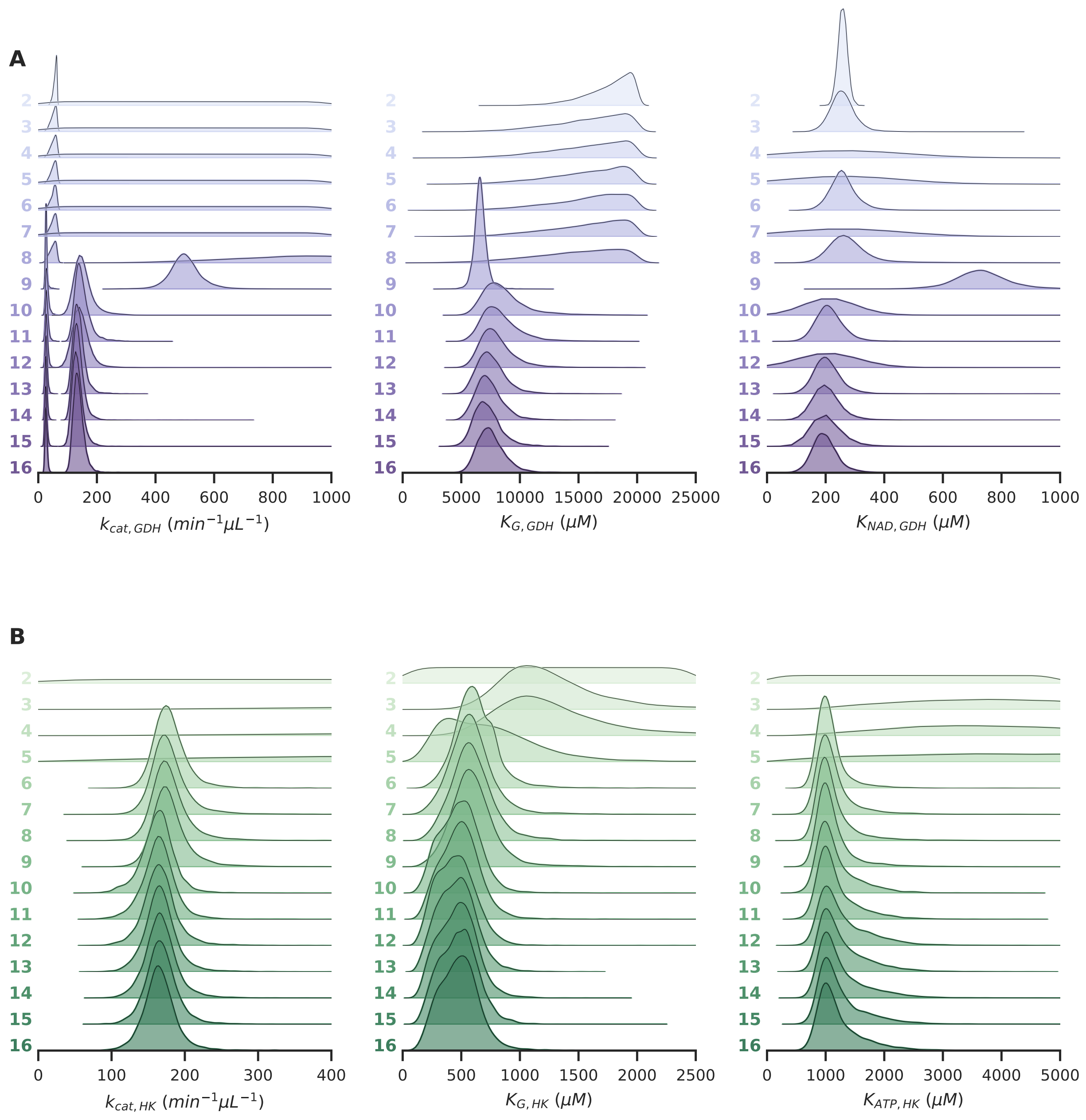


Figure 4: **Extended iterative posterior updating A,B** Posterior parameter estimates obtained from the model combining all three (GDH, HK, GDH+HK) observation likelihoods. For every parameter, the distributions are shown for 15 different datasets, with each following dataset containing an extra experiment, added in chronological order. Distributions are shifted and scaled to increase visibility. For the GDH k_{cat}, two estimates are obtained because PEBs with two different enzyme concentrations were used in different experiments.

Appendix

example_simple

February 9, 2022

1 Bayesian Inference of Enzyme kinetics - Simple example

This notebook will show a basic setup for performing Bayesian inference on enzyme data to estimate kinetic parameters. It will use the same data as shown in figure 1 of this publication: cleavage of the peptide R-AMC to AMC by the enzyme Trypsin, and inhibited by another peptide AAA-AMC.

[1]:
```
# Standard scientific imports
import numpy as np
import pandas as pd
import scipy.stats as stats
import scipy.optimize as optimize

# Bayesian inference imports
import pymc3 as pm
import arviz as az
import theano.tensor as tt

# Visualization imports
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns; sns.set_theme(style='ticks', context='notebook',␣
 ↪font_scale=0.8);

%reload_ext watermark
%watermark -a "Mathieu Baltussen" -d -t -u -v -iv
```

```
Author: Mathieu Baltussen

Last updated: 2022-01-11 11:24:12

Python implementation: CPython
Python version       : 3.9.5
IPython version      : 7.28.0

arviz     : 0.11.4
pandas    : 1.2.4
matplotlib: 3.4.2
seaborn   : 0.11.1
```

```
numpy      : 1.20.3
scipy      : 1.6.2
theano     : 1.1.2
pymc3      : 3.11.4
sys        : 3.9.5 | packaged by conda-forge | (default, Jun 19 2021, 00:32:32)
[GCC 9.3.0]
```

1.1 Data import

Steady-state data obtained from a CSTR flow reactor at different substrate (and inhibitor) input concentrations is obtained and stored in `csv` files. In these files, every row corresponds to a unique observation, and for every observation, the experimental parameters are written alongside the observations. So, for the Trypsin data (`CEKS33.csv`), the datafile has 3 columns: R-AMC input concentration (`R`), AAA-AMC input concentration (`AAA`), and observed AMC output concentration (`AMC`).

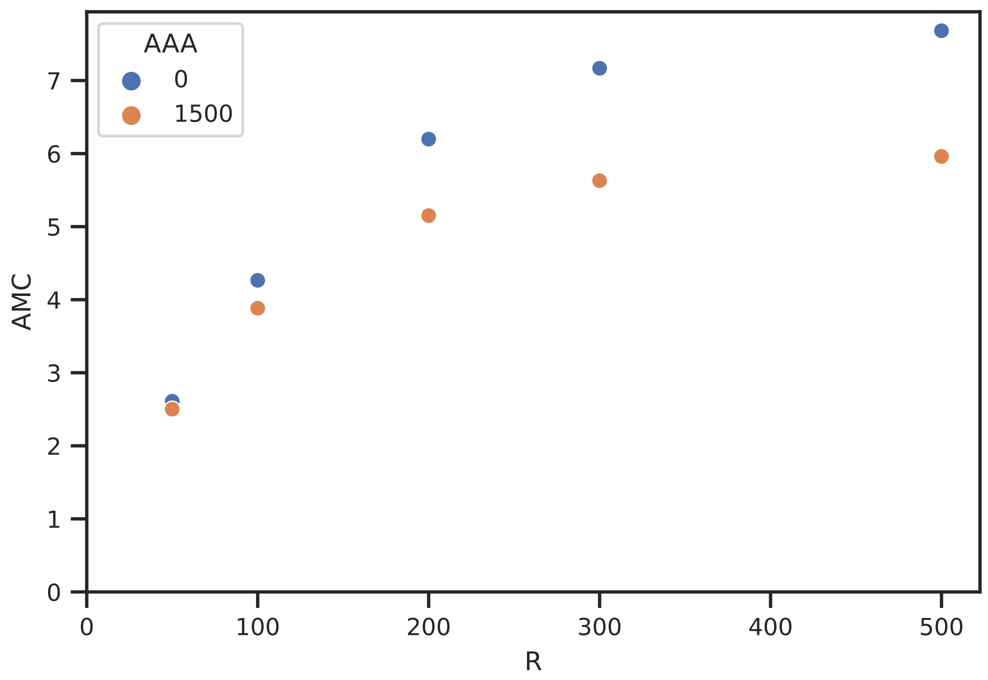
Two other experimental parameters relevant for the analysis are the flowrate k_f and the enzyme concentration E. While in this case, they are the same for the full experimental dataset, this doesn't need to be the case. Here, we add them as additional columns to the dataset after loading in the `csv` file.

In the below two cells, we load in the data and show the resulting Pandas Dataframe, and we plot the data using Seaborn.

[2]:
```
data = pd.read_csv("../data/CEKS33.csv")
kf = 0.125  # minute^-1
E = 0.012
data = data.assign(kf=kf, Tr=E)
data
```

[2]:
```
     R   AAA       AMC     kf     Tr
0   50     0  2.609888  0.125  0.012
1  100     0  4.265064  0.125  0.012
2  200     0  6.197783  0.125  0.012
3  300     0  7.167780  0.125  0.012
4  500     0  7.681850  0.125  0.012
5   50  1500  2.501800  0.125  0.012
6  100  1500  3.882780  0.125  0.012
7  200  1500  5.150944  0.125  0.012
8  300  1500  5.629337  0.125  0.012
9  500  1500  5.961178  0.125  0.012
```

[3]:
```
sns.scatterplot(data=data, x="R", y='AMC', hue='AAA', palette='deep')
plt.xlim(0)
plt.ylim(0)
plt.show()
```



1.2 Model definition

Next, we create a probabilistic model for the steady-state concentrations using PyMC3. For a single substrate enzyme this is relatively simple, and only requires rewriting of the ODE's associated to the system.

1.2.1 Rewriting the ODE's

In the case of R-AMC cleavage by Trypsin, inhibited by AAA-AMC, the system of ODE's is as follows:

$$\frac{dS}{dt} = \frac{-k_{cat}ES}{K_M + S*(1+I/K_I)} + k_f \cdot (S_{in} - S)$$
$$\frac{dP}{dt} = \frac{k_{cat}ES}{K_M + S*(1+I/K_I)} - k_f \cdot P$$

Here, S is the substrate concentration inside the CSTR, P the product concentration inside the CSTR, and I the inhibitor concentration. Flow in to and out of the reactor is accounted for by inclusion of the flow-terms $k_f \cdot (...)$.

As we only observe our system in steady-state ($dP/dt = dS/dt = 0$), this can be implicitly written as:

$$P_{ss} = \frac{k_{cat}ES_{ss}}{k_f * (K_M + S_{ss} * (1 + I/K_I))}$$

or rewritten expliticly using $S_{ss} = S_{in} - P_{ss}$ (which follows from mass conservation):

$$P_{ss} = 0.5(k_{cat}E/\alpha * k_f + K_M/\alpha + S_{in}) - 0.5\sqrt{(k_{cat}E/\alpha * k_f + K_M/\alpha) + S_{in})^2 - 4S_{in}k_{cat}E/\alpha * k_f}$$

where we have rewritten the inhibition factor as $\alpha = 1 + I/K_I$. However, the implicit version can be used during Bayesian inference, and is often easier to write down (and less susceptible to mistakes) than the explicit version.

1.2.2 Stating the likelihood and priors

We know S_{in} and observe corresponding P_{ss} (including noise), so we can infere the parameters k_{cat}, K_M and K_I from this data. We can do this by writing down the likelihood function $P(y|\phi)$, giving the probability of observing the data, given specific values of the kinetic parameters, generally indicated as:

$$\mathcal{L}(y, \phi)$$

where y denotes the observed data and ϕ the parameters to be inferred. In the case of observations of steady-state, we assume that our observations P_{obs} are normally-distributed with some noise σ around the expected value $P_{ss}(S_{in}, k_f, E; K_M, k_{cat}, K_I):

$$\mathcal{L}(y, \phi) = P(P_{obs}|S_{in}, k_f, E; K_M, k_{cat}, K_I) = \mathcal{N}(P_{ss}(S_{in}, k_f, E; K_M, k_{cat}, K_I), \sigma)$$

where the experimental parameters S_{in}, k_f and E are known beforehand, and the kinetic parameters K_M and k_{cat} are unknown.

A priori we only know that both kinetic parameters should be positive-definite. We include this prior knowledge by assuming that these kinetic parameters have a uniformally distributed probability of having a value between 0 and some high maximum boundary (setting this boundary too low will cause estimation problems, because the correct values are then never sampled). This is an example of an uninformative distribution, which ensures that the sampler can converge, without adding subjective information to the model. These priors are indicated as $P(K_M)$, $P(k_{cat})$ and $P(K_I)$. If more knowledge is available about the values of the kinetic parameters, than this knowledge can be used in the analysis by changing the appropriate priors.

Similar to the kinetic parameters, the experimental noise (or more generally, the observational uncertainty) is unknown. Therefore, we also use a prior for this value. This prior generally has an exponential distribution (a particular case of the gamma distribution), which ensures good convergence while remaining uninformative.

Both the *likelihood* and the *priors* are modelled using PyMC3, which also takes care of assembling these probabilities together using Bayes' Theorem, and sampling the probabilistic model to obtain parameter estimates using a Hamiltonian Monte Carlo algorithm. Applying Bayes' Theorem to the priors and likelihood finally results in the expression:

$$P(k_{cat}, K_M, K_I, \sigma|P_{obs}; S_{in}, k_f, E) \propto P(k_{cat}) \cdot P(K_M) \cdot P(K_I) \cdot P(\sigma) \cdot P(P_{obs}|S_{in}, k_f, E; K_M, k_{cat}, K_I)$$

This is shown in the cell below.

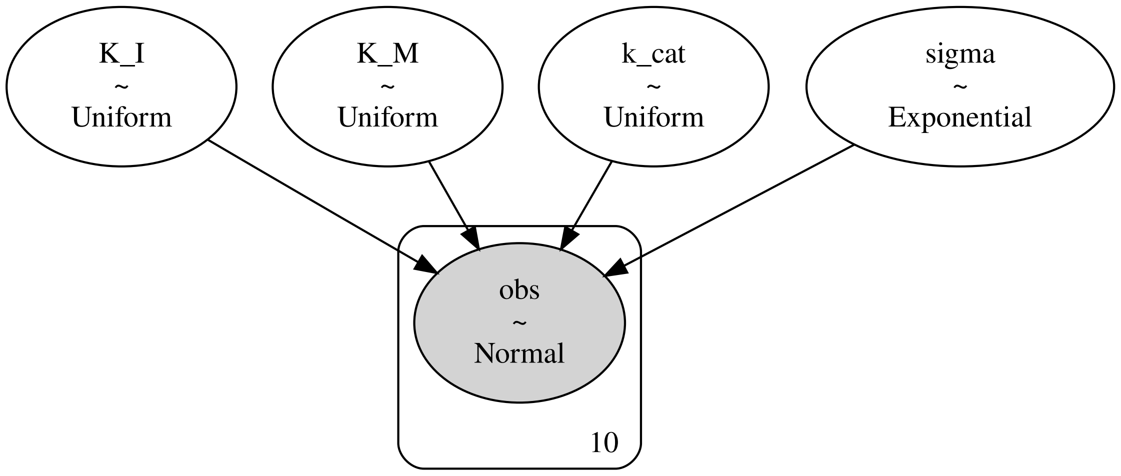
[4]:
```
with pm.Model() as model:
    # Stating the priors
    k_cat = pm.Uniform("k_cat", 0, 500)
    K_M = pm.Uniform("K_M", 0, 500)
    K_I = pm.Uniform("K_I", 1000, 10000)
    sigma = pm.Exponential("sigma", 10)

    # We extract all data from the dataframe here so the likelihood is easier␣
 ↪to write down
    S_in = data["R"].values
    I_in = data["AAA"].values
    P_obs = data["AMC"].values
    S_obs = S_in - P_obs # Substrate concentration inside reactor determined␣
 ↪via stoichiometric conservation at steady-state
    E = data["Tr"].values
    kf = data["kf"].values

    # Inference of probabilistic model at steady-state conditions
    P = pm.Normal(
        "obs",
        mu=k_cat * E * S_obs / (kf * (K_M + S_obs*(1+I_in/K_I))),
        sigma=sigma,
        observed=P_obs
    )
```

1.3 Visualizing the probabilistic model

After composing the model, we can visualize it using graphviz (a built-in function in PyMC3). This graph shows the relationship between the variables which are to be inferred, and the observations that have been made. In the case of this simple model, these relationships are rather straightforward, but this is a useful tool to inspect more complex models based on multiple datasets.

[5]:
```
pm.model_to_graphviz(model)
```

[5]:



1.4 Sampling of the model

After creating the probabilistic model, we can use Monte-Carlo sampling to obtain the posterior probability distribution $P(k_{cat}, K_M, K_I, \sigma | P_{obs}; S_{in}, k_f, E)$. This distribution is obtained as an InferenceData object, which stores meta-information about the priors, data, and sampling, alongside the sampling traces from which the distribution is build.

[6]:
```
with model:
    idata = pm.sample(
        1000,
        tune=1000,
        cores=8,
        step=pm.NUTS(target_accept=0.95),
        return_inferencedata=True,
    )
```

```
Multiprocess sampling (8 chains in 8 jobs)
NUTS: [sigma, K_I, K_M, k_cat]
```

```
<IPython.core.display.HTML object>
```

```
Sampling 8 chains for 1_000 tune and 1_000 draw iterations (8_000 + 8_000 draws
total) took 6 seconds.
```

1.5 Analyzing the posterior

The posterior is obtained as a trace of samples drawn from the posterior distribution, where samples are simultaneously drawn from all inferrable parameters.

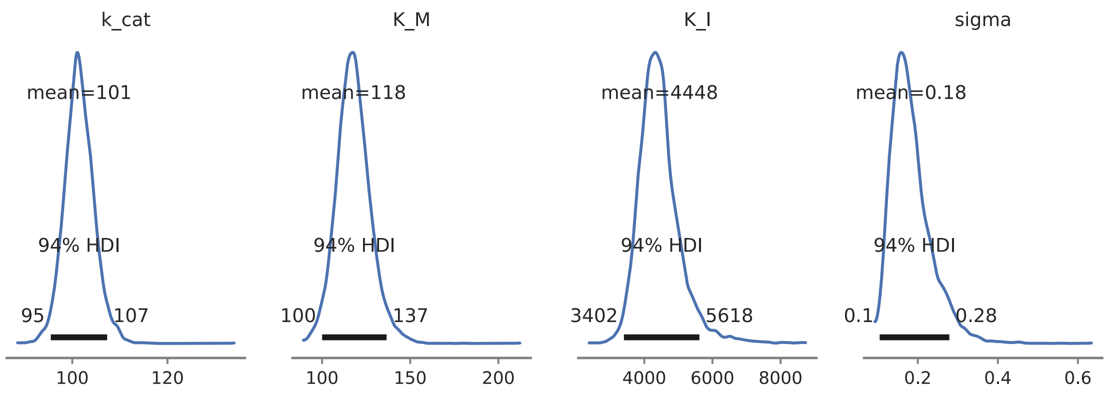
This posterior can then be used for a variety of analysis tasks, such as plotting the full probability distributions of single parameters, or the correlated probability distribution of two parameters. By calculating the mean or specific quantiles of the parameters, specific statistics can be obtained, such as the mean parameter value, the 95% probability (or confidence) interval, or the most likely parameter value (the mode of the distribution).

The Arviz package can be used for quick visualizations of various statistical properties.

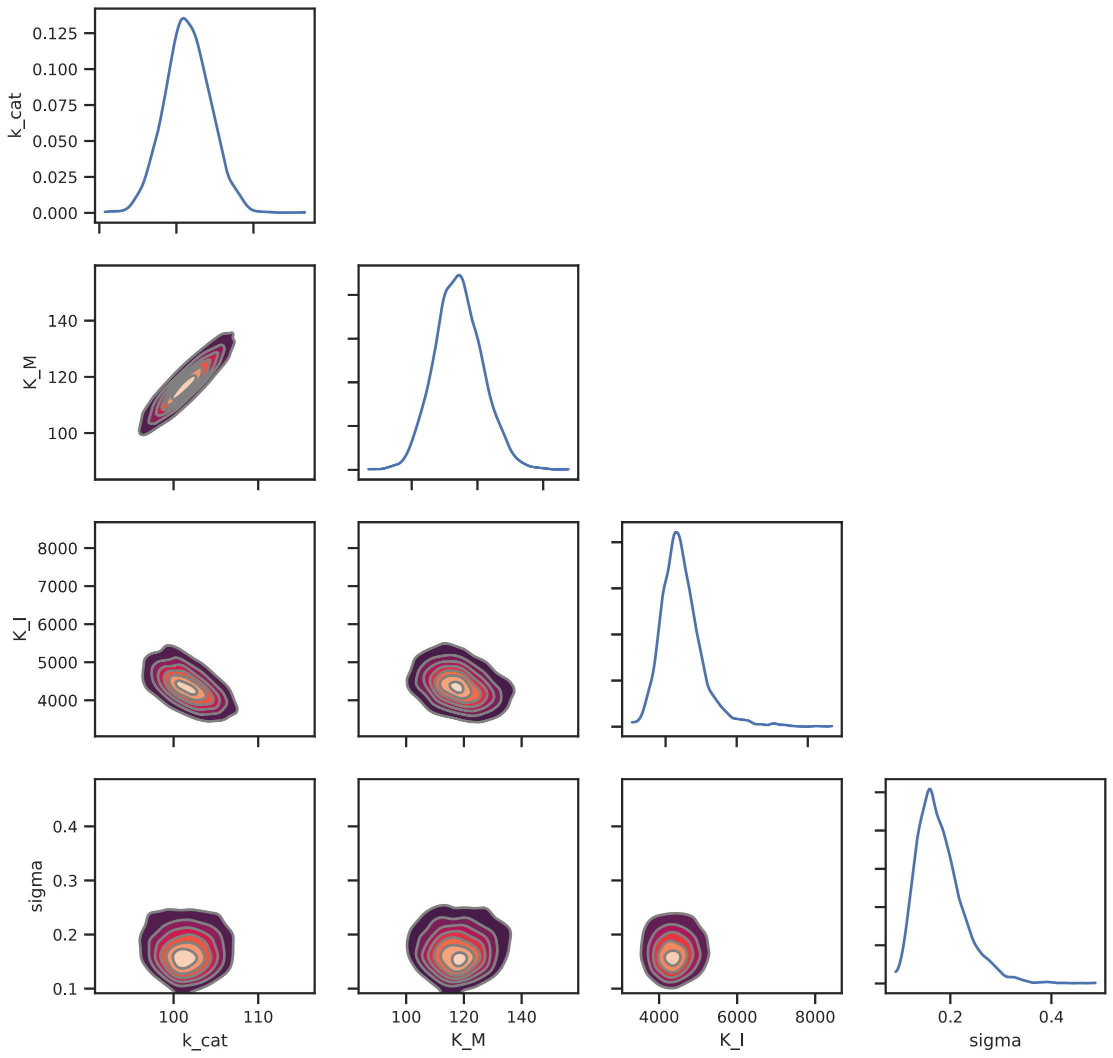
[11]:
```
idata
```

[11]:
```
Inference data with groups:
        > posterior
        > log_likelihood
        > sample_stats
        > observed_data
```

[7]:
```
with model:
    az.plot_posterior(idata, figsize=(10,3))
```



[19]:
```
with model:
    az.plot_pair(idata, marginals=True, kind='kde', divergences=True,␣
 ↪figsize=(10,10))
```



1.6 Analyzing the experimental uncertainty

In addition to analysing the posterior distributions, we can also use the resulting InferenceData to evaluate the experimental uncertainty in our data. We do this by sampling from the posterior predictive distribution

$$P(P_{ss}|k_{cat}, K_M, K_I, \sigma; P_{obs}; S_{in}, k_f, E)$$

which uses the parameter estimates obtaind by the posterior distribution to predict the probability of observing specific product concentrations at specific experimental conditions, taking into account all of the observations that have already been done. Using the previous experimental conditions for this, we can compare the experimental uncertainty σ present in a dataset, relative to the actual observed values.

This is shown in the plot below, where we plot the observations (the dots), alongside the posterior predictive distributions at the same experimental conditions of those observations. Thus, these distributions show us the probability of observing specific values. For experiments with little uncertainty, the posterior predictive should be roughly Normal-distributed with little variance, and with the actual observations corresponding to unbiased samples from those distributions. A large variance, or a bias in observations compared to posterior predictive distributions, is an indication of an experimental dataset with large uncertainty.

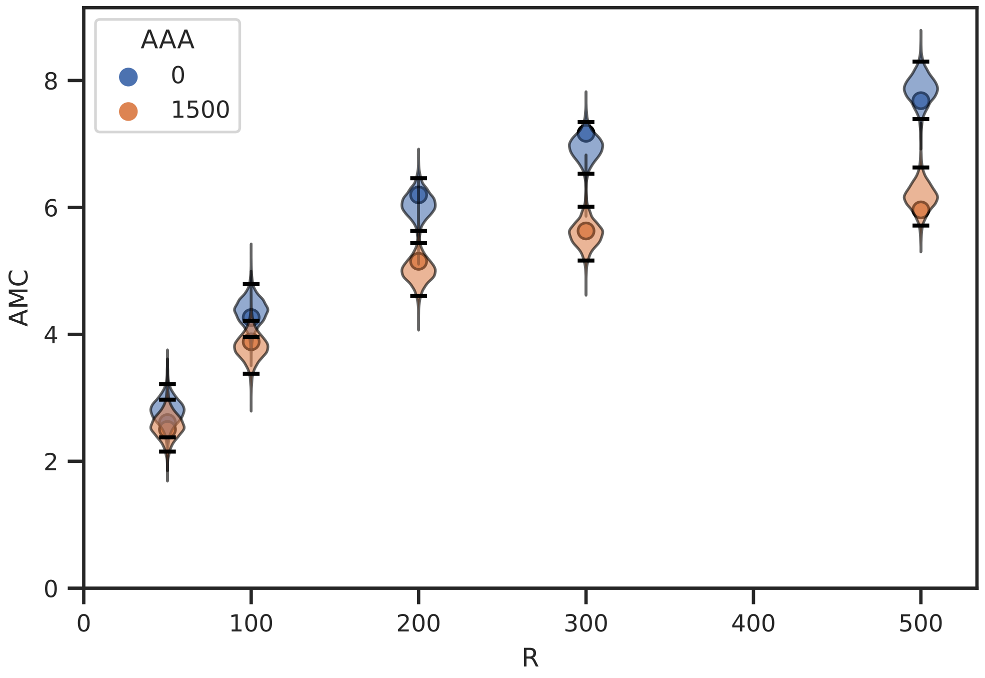
For the single experiment shown in this example, this analysis is not so relevant. But when combining many different datasets together, the uncertainty estimates act as a natural, objective weighting of the observations, and reveals which experiments (relative to all other experiments) have a large uncertainty associated with them.

[53]:
```
with model:
    post_pred = pm.sample_posterior_predictive(idata, var_names=["obs"])
```

```
<IPython.core.display.HTML object>
```

[54]:
```
scat = sns.scatterplot(data=data, x="R", y='AMC', hue='AAA', palette='deep',␣
 ↪edgecolor='black', linewidth=1)

mask_1 = data.AAA == 0
part1_N = len(data[mask_1])
violin_parts = plt.violinplot(post_pred["obs"][:,mask_1],␣
 ↪positions=data[mask_1].R,
showextrema=False, widths=20, quantiles=[[0.025, 0.975]]*part1_N, points=100)
for pc in violin_parts['bodies']:
    pc.set_facecolor('C0')
    pc.set_alpha(0.6)
    pc.set_edgecolor('black')
violin_parts['cquantiles'].set_colors("black")

mask_2 = data.AAA != 0
part2_N = len(data[mask_2])
```

```
violin_parts = plt.violinplot(post_pred["obs"][:,mask_2],␣
 ↪positions=data[mask_2].R,
showextrema=False, widths=20, quantiles=[[0.025, 0.975]]*part2_N, points=100)
for pc in violin_parts['bodies']:
    pc.set_facecolor('C1')
    pc.set_alpha(0.6)
    pc.set_edgecolor('black')
violin_parts['cquantiles'].set_colors("black")

plt.xlim(0)
plt.ylim(0)
plt.show()
```



example_complex

February 9, 2022

1 Bayesian Inference of Enzyme Kinetics - Complex example

This notebook will show how measurements on enzymatic systems with partial observability can be used to perform Bayesian Inference in order to estimate kinetic parameters. This requires the usage of a custom Theano operator that implements the numeric calculation of steady-state concentrations directly from the system of ODE's, alongside with gradient information in order to use Hamiltonian Monte Carlo sampling.

This notebook uses the same dataset of R-AMC cleavage to AMC by the enzyme Trypsin, and inhibited by AAA-AMC, but implement the steady-state condition via a Theano operator instead of explicitly.

[1]:
```
# Standard scientific imports
import numpy as np
import pandas as pd
import scipy.stats as stats
import scipy.optimize as optimize
import sympy as sp

# Bayesian inference imports
import pymc3 as pm
import arviz as az
import theano.tensor as tt

# Visualization imports
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns; sns.set_theme(style='ticks', context='notebook',␣
 ↪font_scale=0.8);

# Package with the SteadyStateOperator
from bayern import ops
# function to speed up numerical computations
from numba import njit

%reload_ext watermark
%watermark -a "Mathieu Baltussen" -d -t -u -v -iv
```

```
Author: Mathieu Baltussen
```

```
Last updated: 2022-01-11 11:54:23

Python implementation: CPython
Python version       : 3.9.5
IPython version      : 7.28.0

pandas    : 1.2.4
numpy     : 1.20.3
pymc3     : 3.11.4
seaborn   : 0.11.1
arviz     : 0.11.4
bayern    : 0.1.0
theano    : 1.1.2
sympy     : 1.8
scipy     : 1.6.2
matplotlib: 3.4.2
sys       : 3.9.5 | packaged by conda-forge | (default, Jun 19 2021, 00:32:32)
[GCC 9.3.0]
```

1.1 Data import

[2]:
```
data = pd.read_csv("../data/CEKS33.csv")
kf = 0.125  # minute^-1
E = 0.012
data = data.assign(kf=kf, Tr=E)
data
```

[2]:
```
     R   AAA       AMC     kf     Tr
0   50     0  2.609888  0.125  0.012
1  100     0  4.265064  0.125  0.012
2  200     0  6.197783  0.125  0.012
3  300     0  7.167780  0.125  0.012
4  500     0  7.681850  0.125  0.012
5   50  1500  2.501800  0.125  0.012
6  100  1500  3.882780  0.125  0.012
7  200  1500  5.150944  0.125  0.012
8  300  1500  5.629337  0.125  0.012
9  500  1500  5.961178  0.125  0.012
```

1.2 Model definition

Similar to the model found in `example_simple.ipynb`, the system is described by a set of ODE's.

$$\mathbf{f}(\mathbf{x}, \phi, \theta) : \begin{cases} \frac{dS}{dt} &= \frac{-k_{cat}ES}{K_M + S*(1+I/K_I)} + k_f \cdot (S_{in} - S) \\ \frac{dP}{dt} &= \frac{k_{cat}ES}{K_M + S*(1+I/K_I)} - k_f \cdot P \end{cases}$$

with state variables $\mathbf{x} = [S, P]$, kinetic parameters $\phi = [k_{cat}, K_M, K_I]$, and control parameters $\theta = [S_{in}, I, k_f, E]$.

These equations are implemented symbolically using the SymPy package. SymPy is then used to determine the symbolic expressions for the Jacobians (J_i) with respect to the state variables $\mathbf{x}$, the kinetic parameters ϕ, and the control parameters θ. These equations, together with the rate equation $\mathbf{f}$ are converted to their numeric counterpart, and wrapped inside a `numba-njit` wrapper, which speeds up their computation by applying optimization algorithms. Using these functions, the gradient of the steady-state conditions $\frac{dg}{d\phi}$ is determined using the Implicit Function Theorem:

$$\frac{dg}{d\phi}(\phi, \theta) = -J_x^{-1} \cdot J_\phi$$

which can be directly calculated from the numeric Jacobians.

Finally, an algorithm is determined to efficiently calculate the steady-state conditions directly from the set of rate equations $\mathbf{f}$. This can be done either via direct integration over a long time, or by finding the root of the function $\mathbf{f}$. However, while the latter method is faster, it may be less reliable in more complex systems.

[3]:
```
R, AMC = sym_x = sp.symbols("R, AMC") # State variables
k_cat, K_M, K_I = sym_phi = sp.symbols("k_cat, K_M, K_I") # Kinetic parameters
R_in, AAA_in, kf, Tr = sym_theta = sp.symbols("R_in, AAA_in, kf, Tr ")  #␣
 ↪Control parameters

# Rate equation
sym_rate_equations = [
    -k_cat * Tr * R / (K_M + R * (1+AAA_in/K_I)) + kf*(R_in - R),
    k_cat * Tr * R / (K_M + R * (1+AAA_in/K_I)) - kf*AMC
]

# The Jacobians
sym_jac_x = sp.Matrix(sym_rate_equations).jacobian(sym_x)
sym_jac_phi = sp.Matrix(sym_rate_equations).jacobian(sym_phi)
sym_jac_theta = sp.Matrix(sym_rate_equations).jacobian(sym_theta)

# Numeric equations
num_rate_equations = njit(sp.lambdify([sym_x, sym_phi, sym_theta],␣
 ↪sym_rate_equations, "numpy"))
num_jac_x = njit(sp.lambdify([sym_x, sym_phi, sym_theta], sym_jac_x, "numpy"))
num_jac_phi = njit(sp.lambdify([sym_x, sym_phi, sym_theta], sym_jac_phi,␣
 ↪"numpy"))
num_jac_theta = njit(sp.lambdify([sym_x, sym_phi, sym_theta], sym_jac_theta,␣
 ↪"numpy"))

# Numeric gradients (theta is currently unused, but still has to be given to␣
 ↪the operator)
num_grad_phi = njit(lambda x,phi,theta: np.dot(-np.linalg.
 ↪inv(num_jac_x(x,phi,theta)),num_jac_phi(x,phi,theta)))
```

```
num_grad_theta = njit(lambda x,phi,theta: np.dot(-np.linalg.
 ↪inv(num_jac_x(x,phi,theta)),num_jac_theta(x,phi,theta)))

# Function to numerically determine the steady-state concentrations
def find_root(fun, jac, phi, theta):
    return optimize.root(fun=fun, x0=[theta[0],0.0], jac=jac, args=(phi,␣
 ↪theta)).x
```

With the numerical expressions for the steady-state concentrations, jacobians, and gradients, we can again construct a probabilistic model using PyMC3. However, now instead of explicitly writing down the steady-state condition, we create an operator (the SteadyStateOp) from the various numerical expressions combined with a matrix containing the control parameters for every observations. This `theta_set` matrix is obtained from the Pandas dataframe holding all data.

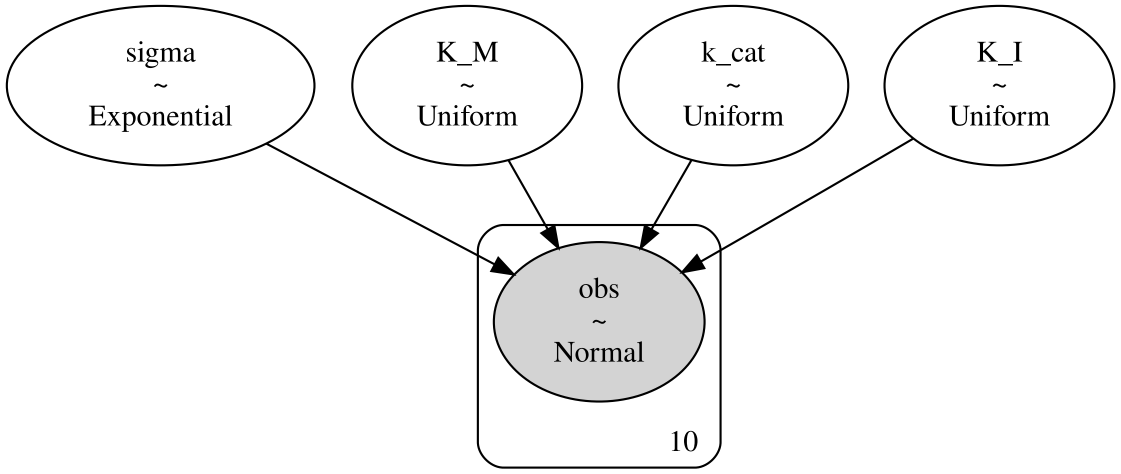
The SteadyStateOp takes a stacked tensor of the kinetic parameters that need to be inferred, and returns the predicted steady-state concentrations for every observation associated to the set of theta-values. As we only observe the product concentration, we only use the second component returned by the operator to feed into the final likelihood calculation.

This probabilistic model can be sampled in the same way as an explicitly defined model, and as expected returns the same results. The posterior obtained from the sampling can be analysed exactly in the same way as well. However, due to explicit numerical calculation of steady-state concentrations and gradients, the process is normally much slower then an explicitly defined model. Thus, when partial observability is not an issue, and when the steady-state condition can be written down explicitly, using the operator is not encouraged.

[4]:
```
with pm.Model() as model:
    # Stating the priors
    k_cat = pm.Uniform("k_cat", 0, 500)
    K_M = pm.Uniform("K_M", 0, 500)
    K_I = pm.Uniform("K_I", 1000, 10000)
    phi_set = tt.stack([k_cat, K_M, K_I])

    sigma = pm.Exponential("sigma", 10)

    # We extract all data from the dataframe here so the likelihood is easier␣
 ↪to write down
    P_obs = data["AMC"].values

    theta_set = data[["R", "AAA", "kf", "Tr"]].values
    SteadyStateOp = ops.SteadyStateDatasetOp(num_rate_equations, num_jac_x,␣
 ↪num_grad_phi, num_grad_theta, find_root, theta_set=theta_set)

    # Inference of probabilistic model at steady-state conditions
    P = pm.Normal(
        "obs",
        mu=SteadyStateOp(phi_set)[:,1],
        sigma=sigma,
        observed=P_obs
    )
```

[25]:
```
pm.model_to_graphviz(model)
```

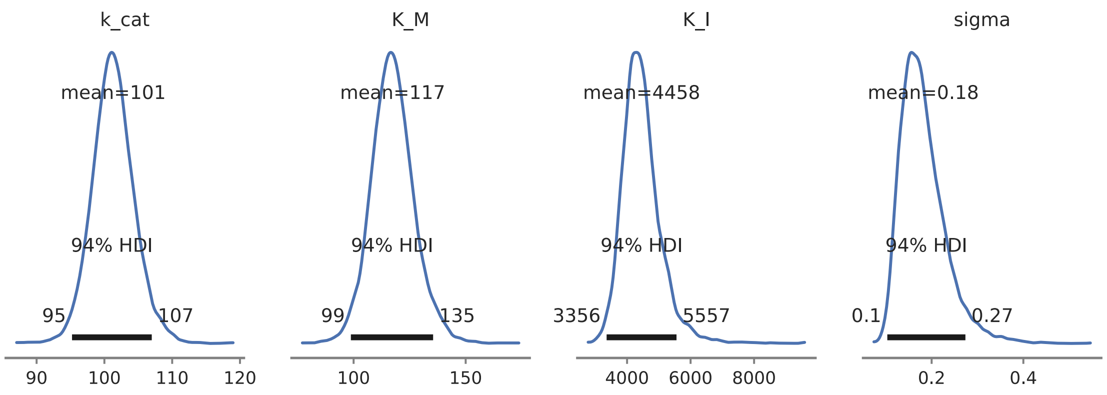
[25]:



[23]:
```
with model:
    idata = pm.sample(
        1000,
        tune=1000,
        cores=8,
        step=pm.NUTS(target_accept=0.95),
        return_inferencedata=True,
    )
```

```
Multiprocess sampling (8 chains in 8 jobs)
NUTS: [sigma, K_I, K_M, k_cat]

<IPython.core.display.HTML object>

Sampling 8 chains for 1_000 tune and 1_000 draw iterations (8_000 + 8_000 draws
total) took 35 seconds.
```

[24]:
```
with model:
    az.plot_posterior(idata, figsize=(10,3))
```


k_cat
mean=101
94% HDI
95
107
90
100
110
120
K_M
mean=117
94% HDI
99
135
100
150
K_I
mean=4458
94% HDI
3356
5557
4000
6000
8000
sigma
mean=0.18
94% HDI
0.1
0.27
0.2
0.4

References

(1) Klein, A. M.; Mazutis, L.; Akartuna, I.; Tallapragada, N.; Veres, A.; Li, V.; Peshkin, L.; Weitz, D. A.; Kirschner, M. W. Droplet Barcoding for Single-Cell Transcriptomics Applied to Embryonic Stem Cells. *Cell* **2015**, *161* (5), 1187–1201. https://doi.org/10.1016/j.cell.2015.04.044.

(2) Matula, K.; Rivello, F.; Huck, W. T. S. Single-Cell Analysis Using Droplet Microfluidics. *Advanced Biosystems* **2020**, *4* (1), 1900188. https://doi.org/10.1002/adbi.201900188.

(3) Semenov, S. N.; Markvoort, A. J.; Gevers, W. B. L.; Piruska, A.; Greef, T. F. A. de; Huck, W. T. S. Ultrasensitivity by Molecular Titration in Spatially Propagating Enzymatic Reactions. *Biophysical Journal* **2013**, *105* (4), 1057–1066. https://doi.org/10.1016/j.bpj.2013.07.002.

(4) Salvatier, J.; Wiecki, T. V.; Fonnesbeck, C. Probabilistic Programming in Python Using PyMC3. *PeerJ Computer Science* **2016**, *2*, e55. https://doi.org/10.7717/peerj-cs.55.

(5) Hoffman, M. D.; Gelman, A. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *arXiv:1111.4246 [cs, stat]* **2011**.

(6) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; Walt, S. J. van der; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; Mulbregt, P. van; Vijaykumar, A.; Bardelli, A. P.; Rothberg, A.; Hilboll, A.; Kloeckner, A.; Scopatz, A.; Lee, A.; Rokem, A.; Woods, C. N.; Fulton, C.; Masson, C.; Häggström, C.; Fitzgerald, C.; Nicholson, D. A.; Hagen, D. R.; Pasechnik, D. V.; Olivetti, E.; Martin, E.; Wieser, E.; Silva, F.; Lenders, F.; Wilhelm, F.; Young, G.; Price, G. A.; Ingold, G.-L.; Allen, G. E.; Lee, G. R.; Audren, H.; Probst, I.; Dietrich, J. P.; Silterra, J.; Webber, J. T.; Slavič, J.; Nothman, J.; Buchner, J.; Kulick, J.; Schönberger, J. L.; Miranda Cardoso, J. V. de; Reimer, J.; Harrington, J.; Rodríguez, J. L. C.; Nunez-Iglesias, J.; Kuczynski, J.; Tritz, K.; Thoma, M.; Newville, M.; Kümmerer, M.; Bolingbroke, M.; Tartre, M.; Pak, M.; Smith, N. J.; Nowaczyk, N.; Shebanov, N.; Pavlyk, O.; Brodtkorb, P. A.; Lee, P.; McGibbon, R. T.; Feldbauer, R.; Lewis, S.; Tygier, S.; Sievert, S.; Vigna, S.; Peterson, S.; More, S.; Pudlik, T.; Oshima, T.; Pingel, T. J.; Robitaille, T. P.; Spura, T.; Jones, T. R.; Cera, T.; Leslie, T.; Zito, T.; Krauss, T.; Upadhyay, U.; Halchenko, Y. O.; Vázquez-Baeza, Y. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. **2020**, *17* (3), 261–272. https://doi.org/10.1038/s41592-019-0686-2.