Article

Workflow for Data Analysis in Experimental and Computational Systems Biology: Using Python as 'Glue'

Melinda Badenhorst[†], Christopher J. Barry[†], Christiaan J. Swanepoel[†], Charles Theo van Staden[†], Julian Wissing[†] and Johann M. Rohwer[*]

Laboratory for Molecular Systems Biology, Department of Biochemistry, Stellenbosch University, 7600 Stellenbosch, South Africa

* Correspondence: jr@sun.ac.za; Tel.: +27-21-808-5843

† These authors contributed equally to this work.

Abstract: Bottom-up systems biology entails the construction of kinetic models of cellular pathways by collecting kinetic information on the pathway components (e.g. enzymes) and collating this into a kinetic model, based for example on ordinary differential equations. This requires integration and data transfer between a variety of tools, ranging from data acquisition in kinetics experiments, to fitting and parameter estimation, to model construction, evaluation and validation. Here, we present a workflow that uses the Python programming language, specifically the modules from the SciPy stack, to facilitate this task. Starting from raw kinetics data, acquired either from spectrophotometric assays with microtitre plates or from NMR spectroscopy time courses, we demonstrate the fitting and construction of a kinetic model using scientific Python tools. The analysis takes place in a Jupyter notebook, which keeps all information related to a particular experiment together in one place and thus serves as an e-labbook, enhancing reproducibility and traceability. The Python programming language serves as an ideal foundation for this framework because it is powerful yet relatively easy to learn for the non-programmer, has a large library of scientific routines and active user community, is open-source and extensible, and many computational systems biology software tools are written in Python or have a Python API. Our workflow thus enables investigators to focus on the scientific problem at hand rather than worrying about data integration between disparate platforms.

Keywords: enzyme kinetics; Jupyter notebook; kinetic modelling; Matplotlib; NMR spectroscopy; optimisation; parametrisation; PySCeS; SciPy; validation

1. Introduction

With the inexorable advance of experimental techniques, the workload of researchers has begun shifting from data generation to data processing and analysis. Accordingly, it will become increasingly important for the systems biologist in the laboratory to utilise computational methods to improve data processing and visualisation of results. Computational systems biology presents the researcher with a powerful toolbox to integrate large kinetic datasets into models and eventually high resolution analyses of biological systems [1]. Here we describe a simple workflow for bridging the gap between experimental and computational systems biology using the Python programming language, which is well suited to this task, especially for the non-expert or novice programmer.

1.1. What is bottom-up systems biology?

The rationale for applying the systems approach to studying living cells is that the effects of dynamically interacting macromolecules can often only be understood in the context of complete systems (e.g. signalling networks or metabolic pathways); unintuitive and emergent properties would be missed if the macromolecules were studied in a reductionist and decontextualised manner without considering their interactions [2]. Systems biology is a broad research field and encompasses many different formalisms and approaches to investigate the dynamics of the system under study [2]. Two opposite approaches of biological model development have emerged, termed 'top-down' and 'bottom-up' [3,4]. The bottom-up approach involves assembling a collection of smaller systems into a more complex system. Bottom-up kinetic models are both mechanistic and dynamic and are capable of steady state and time-course simulations. In contrast to this, the top-down approach often involves constraint-based descriptive modelling where large datasets are used to infer relationships between parameters without necessarily understanding the underlying mechanisms.

Bottom-up systems biology principally involves the construction of kinetic models, their parametrisation and finally validation [4]. The system components are characterised in detail in terms of formulation of mathematical relationships that quantify the dependence of each component on species that it interacts with (in the case of enzymes, these would be enzyme-kinetic rate equations, see e.g. [5,6]). Kinetic parameters for the rate equations are obtained from literature or from experimental studies. Ultimately, these constituent descriptions are integrated into a combined kinetic model in order to describe the whole system from the bottom up [4]. Kinetic models of entire metabolic pathways using this approach can have significant predictive power. For example, a bottom-up kinetic model of epidermic growth factor (EGF) signalling constructed by Kholodenko *et al.* [7] was able to predict short-term cellular responses to EGF. More recently, Van Niekerk *et al.* [8] used a detailed kinetic model of glycolysis in the malaria parasite *Plasmodium falciparum* to identify the glucose transport reaction as a step with significant control on glycolytic flux in the parasite, a finding that was corroborated with experimental analyses.

1.2. Computational analyses and the need for integration

With the exponential increase in computing power, personal computers have become powerful enough to run even moderately large simulations within practical time-frames. This has led to an expansion of the number and variety of computational analyses, as evidenced by the growing number of models in the BioModels database [9]. The challenge has become to chain these analyses together to create higher-level analyses. One of the most routine analyses is model fitting for parameter estimation. Model parameters are estimated and the sum of squares of the difference between the model and experimental data are iteratively minimized. If the model is simple and requires no further analysis, this process can be achieved in mainstream spreadsheet applications. However, if the model requires an ordinary differential equation (ODE) solver, spreadsheet applications will likely be inadequate.

Kinetic models are constructed as a series of reactions that are linked in a stoichiometric network, with each reaction described by an appropriate rate equation (reviewed e.g. in [10]). These are then integrated into a series of ODEs describing the rates of change of the variable species (typically metabolites). Systems of ODEs can be integrated to track changes in species concentrations and reaction rates over time, or solved for steady state using appropriate solvers. Once a model has been constructed and sufficiently parametrised, the system can be simulated under a range of different conditions. These simulations can be used to fit the model parameters to a set of experimental data, or to discover non-intuitive system properties and compare different models of the same system. All of these analyses require close integration between the simulation software and experimental datasets.

1.3. Determining enzyme-kinetic parameters

The foundation of the bottom-up systems biology approach is provided by kinetic parameters, which need to be determined for each enzyme in the pathway investigated. Classically, these parameters are obtained with spectrophotometric assays to determine initial reaction rates. This low-cost technique is well established and measures the progress of a reaction by monitoring the change in a light-absorbing species over time; these assays are frequently miniaturised and the throughput increased by making use of microtitre plates. Enzyme-kinetic parameters for the substrates and products are determined by fitting a kinetic rate equation to datasets of initial rate *versus* concentration.

As a second alternative, if no convenient spectrophotometric assay is available, metabolites can also be measured with (high performance) liquid chromatography, either on its own e.g. using detection by UV-light absorbance, or in combination with mass spectrometry. In contrast to spectrophotometric measurements, this is a discontinuous assay, requiring that the reaction be quenched at different time points before the substrates and products are analysed in order to obtain a time-course.

A third method involves using NMR spectroscopy to follow the progress curve of a reaction or reactions by measuring the concentrations of substrates and products on-line in a non-invasive way. Various time courses with different initial conditions are then fitted to a kinetic model to obtain kinetic parameters for the enzymes [11].

1.4. Why use Python?

In this paper we demonstrate that the Python programming language (http://python.org) is well suited to performing the computational analyses required for experimental data processing, fitting of enzyme-kinetic parameters, construction of kinetic models, as well as model validation and further analysis. Python is an open-source, high-level interpreted programming language that is relatively easy to learn with many applications in data science because of the availability of scientific libraries [12]. Little to no knowledge of computer science is required to learn Python, the syntax is simple and imminently human-readable.

The aim of this work is therefore to introduce a workflow using Python that will aid the experimental researcher to successfully process raw enzyme kinetic data from spectrophotometric assays or NMR spectroscopy time courses, and use these to build a kinetic model. More specifically, the methods will elaborate on how to construct kinetic models using the principles of bottom-up systems biology, to fit experimental data to the model and do validation runs to further test the accuracy of the model. In this way, we will showcase Python and its associated software packages as a ‘glue’ that can assist the investigator with integration and simultaneous processing of numerous datasets.

2. Methods

This section provides a description of the Python modules that were used to assemble the workflow.

2.1. Python libraries for scientific computing

Python has many excellent and well-maintained libraries that facilitate high-level scientific computing analyses. The following libraries were used in this work (references, which link to further documentation, are included):

- numpy [13], a numerical processing library that supports multi-dimensional arrays;
- scipy [14], a scientific processing library providing advanced tools for data analysis, including regression, ODE solvers and integrators, linear algebra and statistical functions;
- pandas [15], a data and table manipulation library that offers similar functionality to spreadsheets such as Excel^{TM}; and
- matplotlib [16], a plotting library with tools to display data in a variety of ways.

These libraries, plus a host of others for data science, can be downloaded as a pre-packaged bundle from various distributions, such as the Anaconda Software Distribution [17], which is freely available for Windows, macOS and Linux. This makes installation of the pre-requisites a simple task.

2.2. Python-based computational biology software

2.2.1. PySCeS

The ease of use of Python lowers the barrier for users to learn programming and thus to develop their own libraries for specific needs. While it is possible in principle to perform simulations of models that can be cast as a system of ODEs directly using numpy and scipy [18], the repetitive nature of many routine modelling tasks prompted our group to develop the open-source Python Simulator for Cellular Systems, PySCeS [19].

PySCeS simplifies the construction and analysis of such metabolic or signalling models by providing a set of high-level functions. A PySCeS model is defined in a human-readable input file according to a defined format, termed the PySCeS Model Description Language. To be able to exchange models with other computational systems biology software, PySCeS can import and export the Systems Biology Markup Language (SBML [20]), the de facto standard in the field.

A number of high-level analyses are available within PySCeS, including a structural analysis module for determination of the nullspace and reduced stoichiometric matrix for models up to the genome scale, time-course simulation through numerical integration of ODEs, steady-state solvers, metabolic control analysis, stability analysis and continuation/bifurcation analysis to identify multistationarity. PySCeS makes use of matplotlib library (see above) to plot the outputs of simulations.

Importantly, many of the leading computational systems biology software programs (e.g. Copasi [21] or libRoadRunner [22]) expose a Python API, making it possible to easily interface with these programs from within Python and PySCeS if needed.

In the workflow presented in this paper, PySCeS was used in the fitting of time-course data to a kinetic model of a multi-enzyme system to obtain kinetic parameters (Section 3.4), as well as for validation of a complete pathway model (Section 3.5).

2.2.2. NMRPy

NMRPy [23] (https://github.com/jeicher/nmrpy) is a Python 3 module that provides a set of tools for processing and analysing NMR data. Its functionality is structured to simplify the analysis of arrayed NMR spectra as were acquired when following reaction time-courses or progress curves (Section 3.2). NMRPy provides an intuitive approach and a number of high-level functions to process such NMR datasets.

NMRPy can import experimental raw data from the major NMR instrument vendors, in this case a Varian NMR spectrometer was used. The processing cycle consisted of apodisation and Fourier transform of the free induction decays (FIDs), phase correction of spectra, identification and picking of peaks representing the metabolites of interest, and finally quantification of metabolites through fitting of Gaussian or Lorenzian functions and normalisation to an internal standard.

2.3. Jupyter notebook as software platform

The various strengths of the Python programming language are enhanced by the IPython architecture [24] (https://ipython.org), which provides a standalone interactive shell as well as a kernel for the interactive Jupyter notebook [25] (https://jupyter.org). The Jupyter notebook runs a server on a local machine which is accessed by a web browser and provides a persistent environment where code, annotations (using Markdown) and graphical outputs are intermixed and can be viewed together. Python code is contained in separately executable cells, which facilitates step-wise debugging.

The Jupyter notebook formed the core of the workflow described in this paper. Because it offers a single interface for annotation and description, code execution and storage of results, everything relating to a particular experiment or analysis could be stored in a single place, which allowed the use of these notebooks as e-labbooks.

3. Results

3.1. Workflow for enzyme kinetics for systems biology

The main workflow for bottom-up kinetic model construction in systems biology, as described in this paper, is summarised in Figure 1. Enzyme kinetic data were obtained in one of two ways: either, progress curves for a reaction or group of reactions were acquired with NMR spectroscopy,

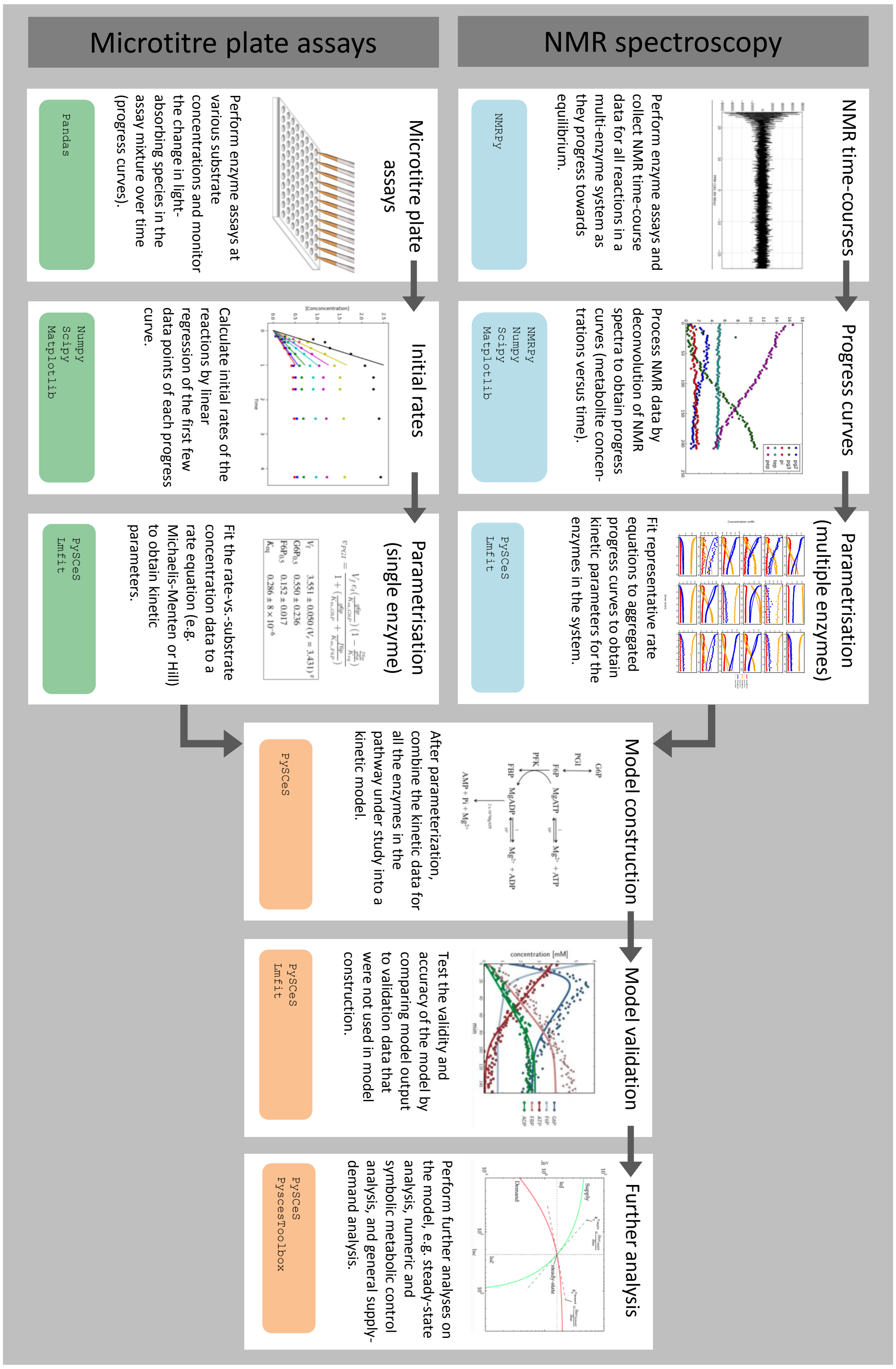
Microtitre plate assays
NMR spectroscopy
Microtitre plate assays
Perform enzyme assays at various substrate concentrations and monitor the change in light-absorbing species in the assay mixture over time (progress curves).
Pandas
NMR time-courses
Perform enzyme assays and collect NMR time-course data for all reactions in a multi-enzyme system as they progress towards equilibrium.
NMRPy
Initial rates
Calculate initial rates of the reactions by linear regression of the first few data points of each progress curve.
Numpy
Scipy
Matplotlib
Progress curves
Process NMR data by deconvolution of NMR spectra to obtain progress curves (metabolite concentrations versus time).
NMRPy
Numpy
Scipy
Matplotlib
Parametrisation (single enzyme)
Fit the rate-vs.-substrate concentration data to a rate equation (e.g. Michaelis-Menten or Hill) to obtain kinetic parameters.
PySCeS
Lmfit
Parametrisation (multiple enzymes)
Fit representative rate equations to aggregated progress curves to obtain kinetic parameters for the system.
PySCeS
Lmfit
Model construction
After parameterization, combine the kinetic data for all the enzymes in the pathway under study into a kinetic model.
PySCeS
Model validation
Test the validity and accuracy of the model by comparing model output to validation data that were not used in model construction.
PySCeS
Lmfit
Further analysis
Perform further analyses on the model, e.g. steady-state analysis, numeric and symbolic metabolic control analysis, and general supply-demand analysis.
PySCeS
PyscesToolbox

Figure 1. The basic workflow for integrating enzyme kinetics for systems biology with computational modelling using Python. For a detailed description see main text.

which were then parametrised by fitting to a system of ODEs with the appropriate enzyme kinetic rate equations; or alternatively, initial-rate kinetics were performed on a single enzyme, typically with a spectrophotometric assay using microtitre plates, and fitted to a rate equation. In this paper, one example of each approach is discussed in detail (Sections 3.2–3.4); in general, it needs to be repeated until all of the enzymes in the pathway under study have been characterised.

In the next step, all the kinetic rate equations and parameters were assembled into a model of the complete pathway, which was then validated by comparing its output to experimental data that were not used for model construction (Section 3.5). The workflow subsequently allowed a number of additional computational analyses to be easily performed on a properly constructed and validated model (Figure 1).

Each of the above steps is described in greater detail in the following sections, emphasising the role of the Python language in 'gluing' the various analyses together. The Python modules that were used for each step are listed at the bottom of each block in Figure 1.

3.2. Enzyme kinetics from NMR spectroscopy

To obtain enzyme-kinetic parameters with NMR spectroscopy, a cell lysate was incubated with substrates, products, cofactors and any allosteric modifiers. A series of NMR spectra was collected over time; the spectra were processed and peaks quantified by deconvolution to yield a series of progress curves, which were then fitted to a kinetic equation or set of equations for the reactions followed. The method [11] can also be applied to purified enzymes.

The use of lysates (in contrast to purified enzyme preparations, which only contain the enzyme of interest) required that reaction boundaries be delimited by omitting essential cofactors as appropriate. For example, the dataset in Figure 2 was acquired by incubating a *Saccharomyces cerevisiae* lysate with phosphoenolpyruvate, leading to the enolase (ENO) and phosphoglycerate mutase (PGM) reactions proceeding in the reverse direction. Subsequent reactions on either side did not proceed because the necessary cofactors (ADP for pyruvate kinase, ATP for phosphoglycerate kinase) were missing. It was important to limit the size of the system of reactions in this way, as fitting too many reactions (and their associated kinetic parameters) at once may lead to unidentifiable parameters [26].

Our custom open-source NMR processing Python module, NMRPy [23], facilitates the bulk-processing and quantification of arrayed NMR spectra that are typically produced by such experiments. Figure 2 provides the raw NMR spectra and quantification of a representative experiment on the PGM-ENO reaction couple, where the reaction was initiated by incubating the lysate with phosphoenolpyruvate. Because this was a ^{31}P-NMR experiment, natural substrates could be used, but when performing ^{13}C-NMR spectroscopy, ^{13}C-labelled substrates have to be used because of the low natural abundance of this NMR-active isotope. The Supplementary Material contains an example Jupyter notebook with code and annotations to read and process the NMR data for Figure 2.

To fit the kinetic parameters for both enzymes, a number of such experiments had to be performed with different starting concentrations of substrates and/or products. The fitting procedure is described in detail in Section 3.4 below.

3.3. Enzyme kinetics from spectrophotometric assays

Enzyme-kinetic parameters were determined from spectrophotometric assays, which were performed on microtitre plates to increase throughput. Initial rates were obtained for different substrate concentrations by linear regression on the initial parts of the reaction progress curves. Where possible, such assays were coupled to reactions producing or consuming NAD(P)H, which has a convenient light absorbance peak at a wavelength of 340 nm and can thus be detected directly with visible-light spectrophotometry. Non-linear regression of the rate-*versus*-concentration data, using appropriate rate equations, yielded the kinetic constants for the enzyme.

Microtitre plate readers typically produce tabulated time *versus* absorbance data, which can be in several formats (CSV, Excel, plain text, etc.). Python has useful modules for dealing with each of

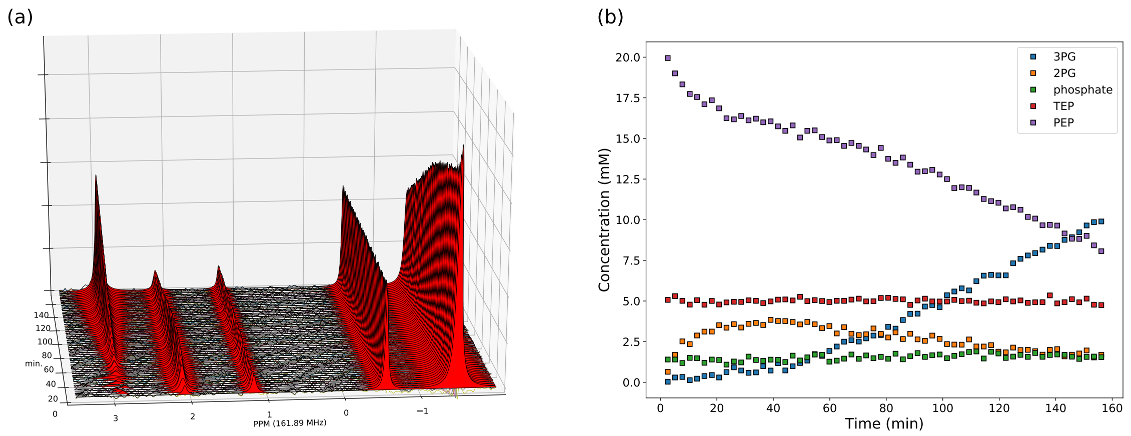

Figure 2. (a) Array of ^{31}P-NMR spectra from an incubation of *S. cerevisiae* lysate with phosphoenolpyruvate. Spectra were acquired 2.6 min apart (repetition time) and processed with NMRPy (apodisation, Fourier transform, phase correction and integration by deconvolution). The peak identities are, from left to right: 3-phosphoglycerate (3PG), 2-phosphoglycerate (2PG), phosphate, triethyl phosphate (TEP, internal standard), and phosphoenolpyruvate (PEP). Original spectra are shown as black lines and the deconvoluted peak areas are shown with filled red colour. (b) Quantification of the spectra after processing with NMRPy. The output from the analysis was concentration-*versus*-time data. NMRPy can read raw data from the major NMR instrument vendors and has built-in functions to display both the arrayed spectra and quantified data. Data and annotated code (Jupyter notebook) are provided in the Supplementary Material.

these; the data analysis library pandas [15] (https://pandas.pydata.org) is specifically suited to this task. Jupyter notebooks in conjunction with matplotlib for visualisation and plotting provided a powerful single interface for data analyses, including loading and preprocessing data, performing the actual data analysis, visualisation and saving the results.

The Supplementary Material contains an annotated example Jupyter notebook to illustrate the processing of kinetic data acquired with a microtitre plate reader. The following steps were involved:

Importing data New dataframes were created in pandas from a variety of input formats, including Excel and CSV. Several preprocessing and customisation methods were used (e.g. for the conversion of date/time fields into a format that can be used by Python), as near-perfect tabulated data are rarely produced by the associated software and the formats differ between instrument vendors.

Linear regression The absorbance-*versus*-time data were subject to linear regression over a suitable time range to calculate initial rates. The attached Jupyter notebook provides two tools (using interactive matplotlib graphs, and using ipywidgets), which were used to efficiently apply this analysis to a large number of datasets.

Preprocessing data The pandas library provided functions to easily normalise the data, either to a single entry, a single row, or an entire dataframe. Further, Python functions were written to automate repetitive processing tasks in a consistent way. Examples of such normalisations included subtraction of blank readings, the conversion of absorbance values to concentrations, or the subtraction of the initial time reading from subsequent time data.

Fitting data For fitting of initial rate data to an enzyme-kinetic rate equation (e.g. the Michaelis-Menten equation), the Python package lmfit [27] provided a high-level interface to various non-linear optimization and curve fitting routines with access to both global and local optimisation algorithms. This is further discussed in Section 3.4 below.

Data Visualization A leading visualisation and 2D-plotting library for Python is matplotlib, which was used in this analysis because of its powerful and flexible design, its interoperability with numpy and scipy, and its excellent integration into Jupyter notebooks.

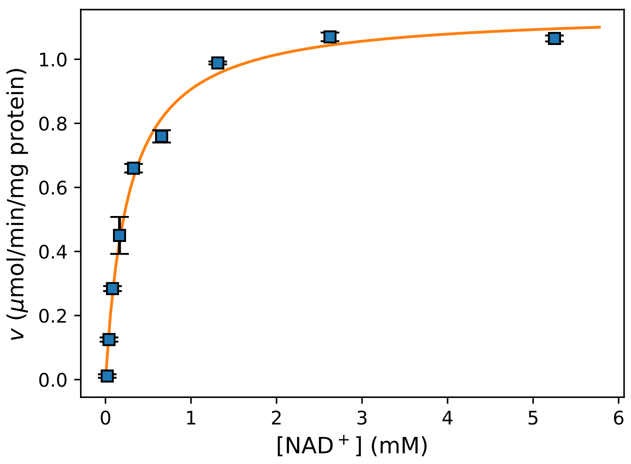

Figure 3. Kinetic characterisation of glucose-6-phosphate dehydrogenase in lysates of *Zymomonas mobilis* by initial rate kinetics. Lysates were incubated with a fixed concentration of glucose-6-phosphate and varying concentrations of NAD$^+$. The graph shows initial rate data for varying NAD$^+$ concentrations (points, mean ± SE of triplicate determinations) and the kinetic equation fit (line). Further details and code are available in the supplementary Jupyter notebook.

3.4. Fitting experimental data to obtain kinetic parameters

In the case of initial rate assays on a single enzyme the rate-*versus*-concentration data for the varying substrate were fitted to an appropriate rate equation by non-linear regression with the lmfit Python module. Figure 3 shows a fit for the enzyme glucose-6-phosphate dehydrogenase as a function of varying NAD$^+$ concentrations. Further details and fitting code are provided in the Jupyter notebook in the Supplementary Material.

Kinetic parameters were obtained from NMR progress curves in a similar way with a few modifications. An ODE model was created for the system of reactions studied in the NMR assay, using generic rate equations. The kinetic parameters were obtained by fitting the experimental data (concentration time courses) to model simulations for the same time period. While the system of ODEs could in principle be coded directly by hand [18], our simulation software PySCeS [19] simplified this task by allowing reaction stoichiometry and rate equations to be intuitively specified in an input file, and by being able to specify directly and easily the time points for which model simulations needed to be output. The fitting was again accomplished with the lmfit module.

Figure 4 shows a representative example of four progress curves for the PGM-ENO couple at different starting concentrations of substrates and products, where the arrayed NMR spectra have already been processed to calculate concentration time-courses (see Section 3.2). Note that some of the reactions ran in reverse and in one case more than one metabolite was present at the start of the assay. The lines represent the model output with the parameters fitted to *all* of the datasets, not only those shown here.

The Supplementary Material contains a Jupyter notebook with the annotated fitting code that provided the fitted parameters and associated error estimates. The workflow allowed for easy-to-follow data processing, from the original NMR FID data to the final fitted parameter values.

3.5. Assembly and validation of a larger kinetic model of a pathway

Once all the enzymes of a pathway under study have been characterised as described in Sections 3.2–3.4, the next step was to combine this information into a kinetic model. The process of bottom-up model construction has been reviewed [10] and will not be repeated in detail here, other than to emphasise the importance of a set of consistent enzyme data, especially in regard to the enzyme

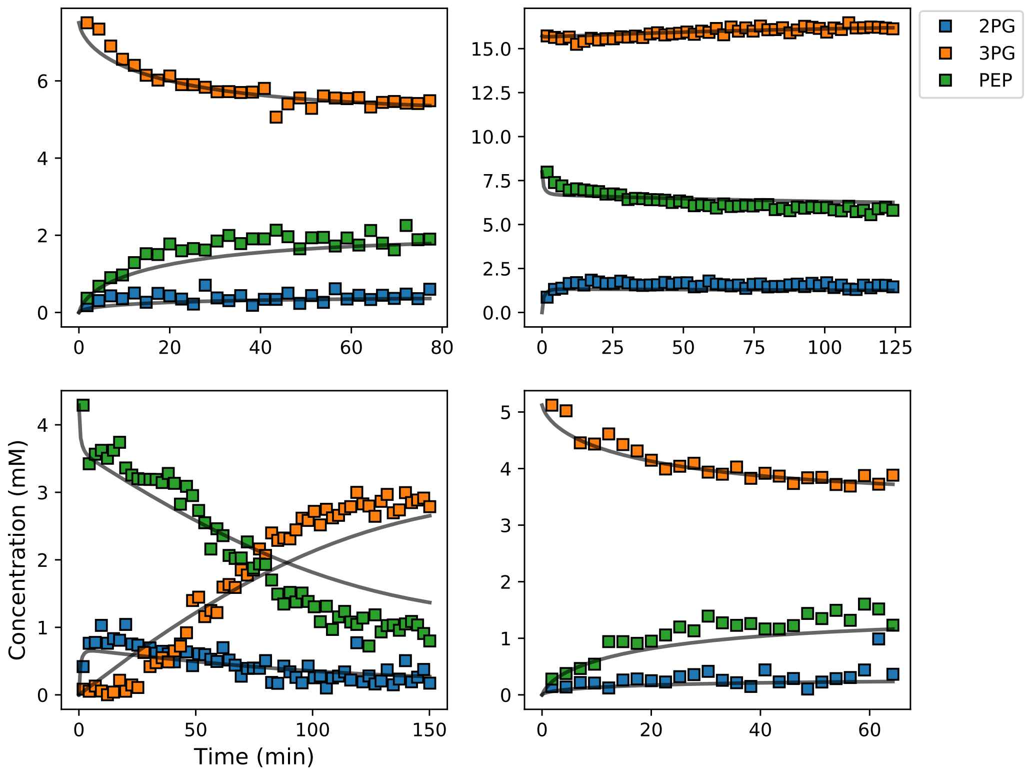

Figure 4. Example of experimental and simulated data for the PGM-ENO couple studied by NMR in *S. cerevisiae* lysates incubated with different starting concentrations of substrates and products. Square symbols represent experimental data and solid lines represent the simulated model data after parameter optimization to all data sets simultaneously. Abbreviations: 2PG, 2-phosphoglycerate; 3PG, 3-phosphoglycerate; PEP, phosphoenolpyruvate.

activities and kinetics, which should all have been determined under the same *in vivo*-like conditions (e.g. [28,29]).

The role of the next step, model validation, was to test the accuracy of the predictions of the model. By investigating how well the model reproduced *independent* experimental data that were *not used* in the model construction process itself (i.e. for fitting the model parameters), this allowed us to assess the quality of the model.

By way of example, Figure 5 shows the output from a kinetic model of *Escherichia coli* glycolysis plotted together with independent metabolite time courses determined *in situ* using *E. coli* cells permeabilised with detergent. This time-course experiment was acquired and processed using similar techniques as outlined Section 3.2, and is from a whole-pathway study starting with the addition of glucose-6-phosphate and co-factors [23]. The Supplementary Material contains a Jupyter notebook with code, model description and data to recreate Figure 5.

While there were some discrepancies between the data and the model fit, the general agreement was remarkable considering that these are independent validation data. At this point, further analyses could be done, e.g. the χ^2 (discrepancy between model and data) could be calculated, or another validation dataset could be plotted and compared to the current one. If different models are available, they can be compared in terms of how well they fit the data [23].

These data could also be used to further fit and refine the model. In this case they are no longer independent validation data, and the model would have to be validated against additional independent experimental data if these are available.

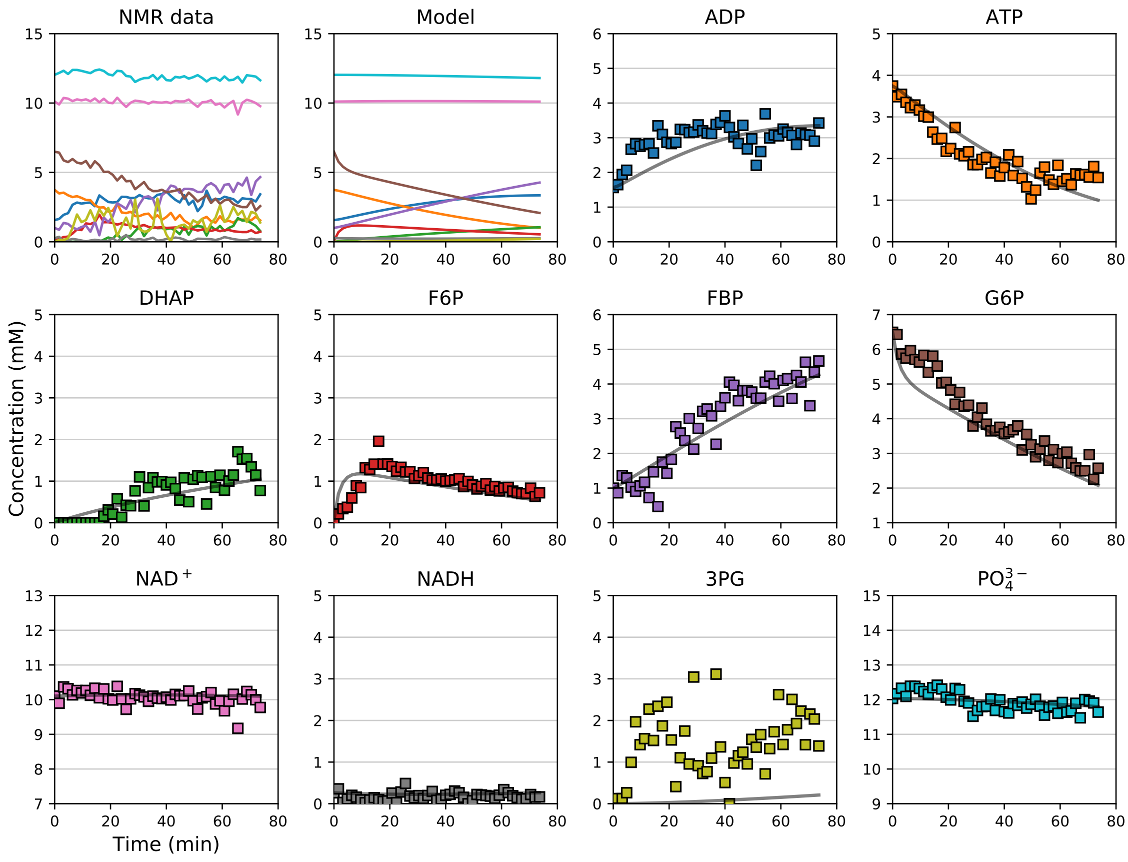

Figure 5. Validation of a kinetic model by comparison to independent experimental data. The data points are quantified NMR time-courses from an *in situ* experiment with permeabilised *Escherichia coli* cells, starting out with 7 mM G6P, 4 mM ATP, 2 mM ADP, 12 mM phosphate, and 10 mM NAD$^+$. The lines are simulation output from a kinetic model of *E. coli* glycolysis, assembled from kinetic measurements on the individual enzymes as outlined in Sections 3.2–3.4. Adapted from [23]. Non-standard abbreviations: DHAP, dihydroxy-acetone phosphate; F6P, fructose-6-phosphate; FBP, fructose-1,6-bisphosphate; G6P, glucose-6-phosphate; 3PG, 3-phosphoglycerate.

3.6. Further model analysis: MCA, GSDA and PyscesToolbox

Once a kinetic model for a pathway has been constructed and properly validated, it can be subject to a variety of analyses to gain further insight into its regulatory function. A fundamental example is metabolic control analysis (MCA) [30,31], which aims to quantify the contribution of each of the steps in a pathway to the control of flux or metabolite concentrations, and thus to identify key control points. PySCeS has built-in functions to perform MCA directly. Other analyses, based on MCA, include supply-demand analysis (SDA) [32,33] and its generalised variant GSDA [34], symbolic MCA (SymCA) [35,36], as well as a framework, ThermoKin, that dissects the contributions of thermodynamic and kinetic aspects to enzyme regulation [37].

The above-mentioned analysis frameworks have been incorporated into a Python module, PySCeSToolbox [38], which uses the Jupyter notebook and IPython kernel to analyse the models with PySCeS and visualise the output in various interactive ways. Figure 6 summarises the overall architecture and workflow of PySCeSToolbox. At the centre of the analysis is a PySCeS model object which can be used to instantiate one of three analysis objects:

RateChar This module performs GSDA by fixing each variable metabolite in turn (thus making it a system parameter) and varying it below and above its steady-state value. This allows one to identify regulatory metabolites as well as routes of regulation in the network. This approach

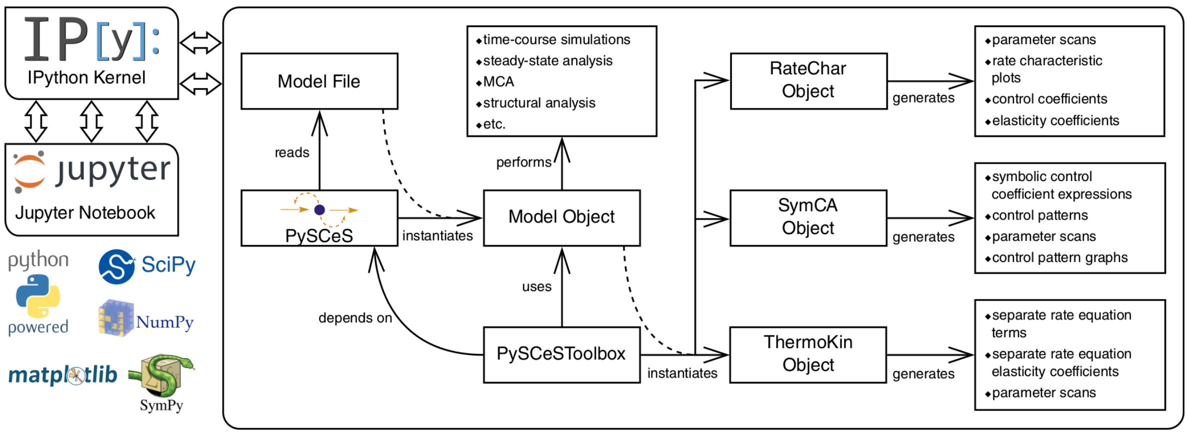

Figure 6. PySCeSToolbox architecture and workflow. PySCeS instantiates a model object from file, which is then used by PySCeSToolbox to instantiate an analysis tool object. The recommended usage involves running these processes within an IPython kernel with which the user interacts via the Jupyter notebook. The bottom-left corner shows some of the main technologies used by PySCeSToolbox. Refer to [38] for details. Reproduced with permission from Christensen *et al.*, Bioinformatics; published by Oxford University Press, 2018.

was computationally applied [39] to the analysis of published models of pyruvate metabolism in *Lactococcus lactis* [40] and aspartate-derived amino acid synthesis in *Arabidopsis thaliana* [41].

SymCA This module performs symbolic metabolic control analysis by generating algebraic expressions for the control coefficients in terms of the elasticity coefficients, using the SymPy Python module for symbolic algebra [42]. These expressions are then used to evaluate and visualise so-called control patterns in the network and quantify their relative contribution to the overall value of the control coefficient. A control coefficient can thus be dissected into its most important components.

ThermoKin This module calculates, for each reversible reaction in the model, the contribution of thermodynamics and kinetics to the enzyme regulation at a particular steady state using the formalism described in [37]. This contribution may vary as conditions change (e.g. as a result of changes in some model parameters), as the reaction operates closer to or further away from equilibrium.

SymCA and ThermoKin were applied [43] to the above-mentioned model of pyruvate metabolism [40].

The main point of this section is to illustrate that fine-grained model analysis can be performed within the same computational framework as the model construction and validation, using Python and Jupyter notebooks; there is no need to change to a new system. The paper describing PySCeSToolbox [38] has example notebooks as supplementary information, illustrating each of the three module functionalities; these will not be repeated here. The detailed model analyses in [39,43] are also accompanied by Jupyter notebooks, allowing readers to reproduce the findings.

4. Discussion

In this paper we have provided examples of applying the powerful capabilities of the Python programming language to systems biology in terms of both computation and data visualisation, making extensive use of the IPython environment and Jupyter notebooks as an interactive platform. One compelling aspect of this is the ability to pass information from one Python software to another to create a versatile computational pipeline. For example, experimental data (e.g. in CSV format) can be directly loaded into the shell, restructured into a numpy array or pandas data frame, analysed by any number of scipy tools, passed into a computational systems biology software such as PySCeS, and finally visualised using the matplotlib plotting library. While all of these functions are available in standalone software packages that are used by researchers, the ability to perform all these functions within a single environment using scripts that can be automated is incredibly powerful and gives researchers with programming skills a significant advantage.

Python is relatively easy to learn compared to other programming languages. The language was designed with a very human-readable format and does not contain the syntactical minutiae of lower level programming languages. This, combined with the excellent freely available resources for learning, makes Python accessible to life-sciences researchers who often have limited computer science training, lowering the barrier of entry and broadening the availability of the analysis platform. In addition to being able to run on different operating systems, Python can integrate with other programming languages and execute Fortran or C code at near-native speeds using the modules f2py [44], which is part of numpy, and cython [45] (https://cython.org). This means that increased readability and interpreted code do not have to come at the expense of computational power and speed, as Fortran and C code can be readily wrapped to run natively in Python by using "interfaces to low-level high-performance software in a high-level programming environment" [44]. For example, for our PySCeS software [19] we have wrapped two additional Fortran solvers that are not part of the standard scipy distribution. Furthermore, the interpreted nature of Python allows scripts to easily be transferred between collaborators without recompiling, meaning that script can be executed on machines with different architectures to produce identical results. This greatly facilitates collaborations between groups and simplifies collaborations within groups.

There are many purpose built proprietary software packages for performing the types of analysis described in this paper. These range from general-purpose mathematical analysis and simulation packages such as Mathematica (http://www.wolfram.com/mathematica) or MATLAB (https://www.mathworks.com/products/matlab.html) to dedicated analysis software provided by instrument vendors. While these programs are usually excellent at performing the tasks for which they were designed, the fact that they are closed-source limits their extensibility.

In contrast, the Python programming language and the libraries described in this paper are open-source. The Open Source Initiative (OSI, https://opensource.org/) is an organisation that promotes awareness and adoption of open-source software and protects open-source communities of practice. Open-source software development is a system of collaboration where loosely affiliated contributors work towards a common software goal or innovation, making the source codes of these projects available to the public. Python and the OSI are inextricably linked; the founder of Python, Guido van Rossum, has served on the OSI board of Directors and championed the movement by organising 'sprint' events, often following Python conventions such as PyCon (http://www.pycon.org/), where programmers work on a variety of open- source projects. This approach extends to the scientific Python community, which organises annual SciPy and EuroSciPy conferences (https://conference.scipy.org/). Python has a healthy, active and supportive community, which facilitates its adoption. This creates a feed-forward mechanism where researchers can work and develop new tools in Python because these can be easily integrated into existing software pipelines.

The workflow described in this paper latches on to the above feed-forward mechanism by integrating various tools. As such, the list is by no means exhaustive but rather a collection of examples that we use in day-to-day analyses. We do not claim that Python is the best, nor is the aim of this paper to provide a systematic comparison of programming languages or tools; rather, it is an illustration of an adaptable and expandable workflow that has proven useful in our hands. In addition, while our examples in the Supplementary Material are presented as Jupyter notebooks, this is not a strict requirement and the analysis could have been performed with a series of Python scripts. The interactive nature of Jupyter, as well as its capabilities for annotation, structuring and visualisation, just provided additional functionality.

To further substantiate the case for Python in systems biology, we note that, while we have focussed in this paper on those programs and libraries that are most frequently used in our group, researchers have a wide choice of software, many of which are either written in Python or expose a Python API (summarised in the SBML software matrix, see http://sbml.org/SBML_Software_Guide/

SBML_Software_Matrix). Each of these programs is dedicated to particular analysis tasks, and they will not all be covered in detail here. To mention only a few in addition to the libraries listed in Section 2.2.1, modelbase [46] has a focus on kinetic modelling similar to PySCeS, while COBRAPy [47] and CBMPy (http://cbmpy.sourceforge.net/) have a focus on constraint-based modelling of large stoichiometric networks. ScrumPy [48] can do both, but has a focus on constraint-based modelling. Importantly, by working in a Python environment, the user has the flexibility to interact with any of these programs as required and to easily create new workflows.

In the field of systems biology, new software and technical developments have led to an exponential growth in the rate of experimental data generation, particularly via high throughput methods [49], as well as the number and complexity of computational tools. Concomitantly, the importance of sharing data and resources is increasingly being recognised. Computational systems biology models are curated and stored in databases such as JWS Online [50] and BioModels [9]. Moreover, standards such as SBML [20] facilitate exchange of models across simulation tools. The need to share, exchange and reuse data led to the development of the FAIRDOM project [51], which aims to develop frameworks and guidelines to make data more Findable, Accessible, Interoperable and Reusable. This project has produced the FAIRDOMHub which uses the SEEK [52] open-source web platform with tools for collating and annotating datasets, models, simulations and research outcomes. SEEK has a JavaScript Object Notation (JSON) API for uploading and downloading files, and Python supports JSON natively, facilitating integration into Python workflows.

5. Conclusions

We have demonstrated how the Python programming language can act as a glue to interface between different analysis tools required for the construction, validation and analysis of kinetic models in bottom-up systems biology. Our workflow enables investigators to focus on the scientific problem instead of issues of data integration between platforms. The Jupyter notebook is an ideal e-labbook and allows the user to keep everything related to a particular analysis in one place, including raw data, graphical output and descriptive annotations.

Supplementary Materials: The following are available at
http://www.mdpi.com//xx/1/5/s1, Document S1: Instructions for Running Supplementary Notebooks;
http://www.mdpi.com//xx/1/5/s2, Archive S2: ZIP archive with supplementary notebooks and associated data files.

Author Contributions: conceptualization, J.M.R.; methodology, C.J.S, C.T.v.S., J.W., J.M.R.; software, C.J.S., C.T.V.S., J.W., J.M.R.; validation, M.B., C.J.B.; formal analysis, C.J.S, C.T.v.S., J.W.; investigation, C.J.S, C.T.v.S., J.W.; resources, J.M.R.; data curation, M.B., C.J.B.; writing–original draft preparation, all authors; writing–review and editing, J.M.R., C.J.B.; visualization, M.B., C.J.S., J.W., J.M.R.; supervision, J.M.R.; project administration, J.M.R.; funding acquisition, J.M.R.

Funding: This research was funded by the National Research Foundation (South Africa), grant numbers 93466, 93670, and 114748, as well as by Stellenbosch University (student scholarships to CJS and MB). The APC was funded in part from the Open Access Publication Fund of Stellenbosch University.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
ENO	Enolase
FID	Free Induction Decay
GSDA	Generalised Supply-Demand Analysis
JSON	JavaScript Object Notation
MCA	Metabolic Control Analysis
NMR	Nuclear Magnetic Resonance
ODE	Ordinary Differential Equation
OSI	Open Source Initiative
PGM	Phosphoglycerate Mutase
SBML	Systems Biology Markup Language
SDA	Supply-Demand Analysis

References

1. Kitano, H. International alliances for quantitative modeling in systems biology. *Mol. Syst. Biol.* **2005**, *1*, 2005.0007. doi:10.1038/msb4100011.
2. Westerhoff, H.V.; Alberghina, L. Systems Biology: Did we know it all along?; Springer-Verlag: Berlin, 2005; pp. 3–9. doi:10.1007/b137744.
3. Snoep, J.L.; Bruggeman, F.; Olivier, B.G.; Westerhoff, H.V. Towards building the silicon cell: a modular approach. *Biosystems* **2006**, *83*, 207–216. doi:10.1016/j.biosystems.2005.07.006.
4. Bruggeman, F.J.; Westerhoff, H.V. The nature of systems biology. *Trends Microbiol.* **2007**, *15*, 45–50. doi:10.1016/j.tim.2006.11.003.
5. Rohwer, J.M.; Hanekom, A.J.; Crous, C.; Snoep, J.L.; Hofmeyr, J.H.S. Evaluation of a simplified generic bi-substrate rate equation for computational systems biology. *IEE Proc. Syst. Biol.* **2006**, *153*, 338–341.
6. Rohwer, J.M.; Hanekom, A.J.; Hofmeyr, J.H.S. A universal rate equation for systems biology. Experimental Standard Conditions of Enzyme Characterizations. Proceedings of the 2nd International Beilstein Workshop; Hicks, M.G.; Kettner, C., Eds.; Beilstein-Institut zur Förderung der Chemischen Wissenschaften: Frankfurt, 2007; pp. 175–187.
7. Kholodenko, B.N.; Demin, O.V.; Moehren, G.; Hoek, J.B. Quantification of short term signaling by the epidermal growth factor receptor. *J. Biol. Chem.* **1999**, *274*, 30169–30181. doi:10.1074/jbc.274.42.30169.
8. van Niekerk, D.D.; Penkler, G.P.; du Toit, F.; Snoep, J.L. Targeting glycolysis in the malaria parasite *Plasmodium falciparum*. *FEBS J.* **2016**, *283*, 634–646. doi:10.1111/febs.13615.
9. le Novère, N.; Bornstein, B.; Broicher, A.; Courtot, M.; Donizelli, M.; Dharuri, H.; Li, L.; Sauro, H.; Schilstra, M.; Shapiro, B.; Snoep, J.L.; Hucka, M. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res.* **2006**, *34*, D689–D691. doi:10.1093/nar/gkj092.
10. Rohwer, J.M. Kinetic modelling of plant metabolic pathways. *J. Exp. Bot.* **2012**, *63*, 2275–2292. doi:10.1093/jxb/ers080.
11. Eicher, J.J.; Snoep, J.L.; Rohwer, J.M. Determining enzyme kinetics for systems biology with Nuclear Magnetic Resonance spectroscopy. *Metabolites* **2012**, *2*, 818–843. doi:10.3390/metabo2040818.
12. Oliphant, T.E. Python for scientific computing. *Comput. Sci. Eng.* **2007**, *9*, 10–20. doi:10.1109/MCSE.2007.58.
13. van der Walt, S.; Colbert, S.C.; Varoquaux, G. The NumPy array: a structure for efficient numerical computation. *Comput. Sci. Eng.* **2011**, *13*, 22–30. doi:10.1109/MCSE.2011.37.
14. Jones, E.; Oliphant, T.; Peterson, P.; others. SciPy: Open source scientific tools for Python, 2001–.
15. McKinney, W. Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference; van der Walt, S.; Millman, J., Eds., 2010, pp. 51–56.
16. Hunter, J.D. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering* **2007**, *9*, 90–95. doi:10.1109/MCSE.2007.55.
17. Anaconda Software Distribution. Computer Software, 2017. Vers. 2-2.4.0, https://www.anaconda.com.
18. Olivier, B.G.; Rohwer, J.M.; Hofmeyr, J.H.S. Modelling cellular processes with Python and SciPy. *Mol. Biol. Rep.* **2002**, *29*, 249–254.
19. Olivier, B.G.; Rohwer, J.M.; Hofmeyr, J.H.S. Modelling cellular systems with PySCeS. *Bioinformatics* **2005**, *21*, 560–561.

20. Hucka, M.; Finney, A.; Sauro, H.M.; Bolouri, H.; Doyle, J.C.; Kitano, H.; Arkin, A.P.; Bornstein, B.J.; Bray, D.; Cornish-Bowden, A.; Cuellar, A.A.; Dronov, S.; Gilles, E.D.; Ginkel, M.; Gor, V.; Goryanin, I.I.; Hedley, W.J.; Hodgman, T.C.; Hofmeyr, J.H.; Hunter, P.J.; Juty, N.S.; Kasberger, J.L.; Kremling, A.; Kummer, U.; Novère, N.L.; Loew, L.M.; Lucio, D.; Mendes, P.; Minch, E.; Mjolsness, E.D.; Nakayama, Y.; Nelson, M.R.; Nielsen, P.F.; Sakurada, T.; Schaff, J.C.; Shapiro, B.E.; Shimizu, T.S.; Spence, H.D.; Stelling, J.; Takahashi, K.; Tomita, M.; Wagner, J.; Wang, J. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* **2003**, *19*, 524–531.
21. Hoops, S.; Sahle, S.; Gauges, R.; Lee, C.; Pahle, J.; Simus, N.; Singhal, M.; Xu, L.; Mendes, P.; Kummer, U. COPASI–a COmplex PAthway SImulator. *Bioinformatics* **2006**, *22*, 3067–3074. doi:10.1093/bioinformatics/btl485.
22. Somogyi, E.T.; Bouteiller, J.M.; Glazier, J.A.; König, M.; Medley, J.K.; Swat, M.H.; Sauro, H.M. libRoadRunner: a high performance SBML simulation and analysis library. *Bioinformatics* **2015**, *31*, 3315–3321. doi:10.1093/bioinformatics/btv363.
23. Eicher, J.J. Understanding Glycolysis in *Escherichia coli* : a Systems Approach using Nuclear Magnetic Resonance Spectroscopy. PhD thesis, Stellenbosch University, 2013.
24. Pérez, F.; Granger, B.E. IPython: a system for interactive scientific computing. *Comput. Sci. Eng.* **2007**, *9*, 21–29. doi:10.1109/MCSE.2007.53.
25. Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Granger, B.E.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.B.; Grout, J.; Corlay, S.; Ivanov, P.; Avila, D.; Abdalla, S.; Willing, C.; others. Jupyter Notebooks – a publishing format for reproducible computational workflows; IOS Press: Amsterdam, 2016; pp. 87–90. doi:10.3233/978-1-61499-649-1-87.
26. Ashyraliyev, M.; Fomekong-Nanfack, Y.; Kaandorp, J.A.; Blom, J.G. Systems biology: parameter estimation for biochemical models. *FEBS J.* **2009**, *276*, 886–902.
27. Newville, M.; Stensitzki, T.; Allen, D.B.; Ingargiola, A. LMFIT: Non-linear least-square minimization and curve-fitting for Python. Zenodo, 2014. doi:10.5281/zenodo.11813.
28. van Eunen, K.; Bouwman, J.; Daran-Lapujade, P.; Postmus, J.; Canelas, A.B.; Mensonides, F.I.C.; Orij, R.; Tuzun, I.; van den Brink, J.; Smits, G.J.; van Gulik, W.M.; Brul, S.; Heijnen, J.J.; de Winde, J.H.; de Mattos, M.J.T.; Kettner, C.; Nielsen, J.; Westerhoff, H.V.; Bakker, B.M. Measuring enzyme activities under standardized *in vivo*-like conditions for systems biology. *FEBS J.* **2010**, *277*, 749–760. doi:10.1111/j.1742-4658.2009.07524.x.
29. García-Contreras, R.; Vos, P.; Westerhoff, H.V.; Boogerd, F.C. Why *in vivo* may not equal *in vitro* – new effectors revealed by measurement of enzymatic activities under the same *in vivo*-like assay conditions. *FEBS J.* **2012**, *279*, 4145–4159. doi:10.1111/febs.12007.
30. Kacser, H.; Burns, J.A. The control of flux. *Symp. Soc. Exp. Biol.* **1973**, *27*, 65–104.
31. Heinrich, R.; Rapoport, T.A. A linear steady-state treatment of enzymatic chains. General properties, control and effector strength. *Eur. J. Biochem.* **1974**, *42*, 89–95.
32. Hofmeyr, J.H.S.; Cornish-Bowden, A. Regulating the cellular economy of supply and demand. *FEBS Lett.* **2000**, *476*, 47–51.
33. Hofmeyr, J.H.S.; Rohwer, J.M. Supply-demand analysis: a framework for exploring the regulatory design of metabolism. *Methods Enzymol.* **2011**, *500*, 533–554. doi:10.1016/B978-0-12-385118-5.00025-6.
34. Rohwer, J.M.; Hofmeyr, J.H.S. Identifying and characterising regulatory metabolites with generalised supply-demand analysis. *J. Theor. Biol.* **2008**, *252*, 546–554. doi:10.1016/j.jtbi.2007.10.032.
35. Reder, C. Metabolic control theory: A structural approach. *J. Theor. Biol.* **1988**, *135*, 175–201.
36. Hofmeyr, J.H.S. Metabolic control analysis in a nutshell. Proceedings of the 2nd International Conference on Systems Biology; Yi, T.M.; Hucka, M.; Morohashi, M.; Kitano, H., Eds.; Omnipress: Madison, WI, USA, 2001; pp. 291–300.
37. Rohwer, J.M.; Hofmeyr, J.H.S. Kinetic and thermodynamic aspects of enzyme control and regulation. *J. Phys. Chem. B* **2010**, *114*, 16280–16289. doi:10.1021/jp108412s.
38. Christensen, C.D.; Hofmeyr, J.H.S.; Rohwer, J.M. PySCeSToolbox: a collection of metabolic pathway analysis tools. *Bioinformatics* **2018**, *34*, 124–125. doi:10.1093/bioinformatics/btx567.
39. Christensen, C.D.; Hofmeyr, J.H.S.; Rohwer, J.M. Tracing regulatory routes in metabolism using generalised supply-demand analysis. *BMC Syst. Biol.* **2015**, *9*, 89. doi:10.1186/s12918-015-0236-1.

40. Hoefnagel, M.H.N.; Starrenburg, M.J.C.; Martens, D.E.; Hugenholtz, J.; Kleerebezem, M.; Swam, I.I.V.; Bongers, R.; Westerhoff, H.V.; Snoep, J.L. Metabolic engineering of lactic acid bacteria, the combined approach: kinetic modelling, metabolic control and experimental analysis. *Microbiology* **2002**, *148*, 1003–1013.
41. Curien, G.; Bastien, O.; Robert-Genthon, M.; Cornish-Bowden, A.; Cárdenas, M.L.; Dumas, R. Understanding the regulation of aspartate metabolism using a model based on measured kinetic parameters. *Mol. Syst. Biol.* **2009**, *5*, 271. doi:10.1038/msb.2009.29.
42. Meurer, A.; Smith, C.P.; Paprocki, M.; Čertík, O.; Kirpichev, S.B.; Rocklin, M.; Kumar, A.; Ivanov, S.; Moore, J.K.; Singh, S.; Rathnayake, T.; Vig, S.; Granger, B.E.; Muller, R.P.; Bonazzi, F.; Gupta, H.; Vats, S.; Johansson, F.; Pedregosa, F.; Curry, M.J.; Terrel, A.R.; Roučka, v.; Saboo, A.; Fernando, I.; Kulal, S.; Cimrman, R.; Scopatz, A. SymPy: symbolic computing in Python. *PeerJ Comput. Sci.* **2017**, *3*, e103. doi:10.7717/peerj-cs.103.
43. Christensen, C.D.; Hofmeyr, J.H.S.; Rohwer, J.M. Delving deeper: Relating the behaviour of a metabolic system to the properties of its components using symbolic metabolic control analysis. *PLoS One* **2018**, *13*, e0207983. doi:10.1371/journal.pone.0207983.
44. Peterson, P. F2PY: a tool for connecting Fortran and Python programs. *Int. J. Comput. Sci. Eng.* **2009**, *4*, 296. doi:10.1504/ijcse.2009.029165.
45. Dalcin, L.; Bradshaw, R.; Smith, K.; Citro, C.; Behnel, S.; Seljebotn, D. Cython: the best of both worlds. *Comput. Sci. Eng.* **2011**, *13*, 31–39. doi:10.1109/MCSE.2010.118.
46. Ebenhöh, O.; van Aalst, M.; Saadat, N.P.; Nies, T.; Matuszyńska, A. Building mathematical models of biological systems with modelbase. *Journal of Open Research Software* **2018**, *6*. doi:10.5334/jors.236.
47. Ebrahim, A.; Lerman, J.A.; Palsson, B.O.; Hyduke, D.R. COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Syst. Biol.* **2013**, *7*, 74. doi:10.1186/1752-0509-7-74.
48. Poolman, M.G. ScrumPy: metabolic modelling with Python. *Syst. Biol. (Stevenage)* **2006**, *153*, 375–378.
49. Cook, C.E.; Bergman, M.T.; Finn, R.D.; Cochrane, G.; Birney, E.; Apweiler, R. The European Bioinformatics Institute in 2016: Data growth and integration. *Nucleic Acids Res.* **2016**, *44*, D20–D26. doi:10.1093/nar/gkv1352.
50. Olivier, B.G.; Snoep, J.L. Web-based kinetic modelling using JWS Online. *Bioinformatics* **2004**, *20*, 2143–2144. doi:10.1093/bioinformatics/bth200.
51. Wolstencroft, K.; Krebs, O.; Snoep, J.L.; Stanford, N.J.; Bacall, F.; Golebiewski, M.; Kuzyakiv, R.; Nguyen, Q.; Owen, S.; Soiland-Reyes, S.; Straszewski, J.; van Niekerk, D.D.; Williams, A.R.; Malmström, L.; Rinn, B.; Müller, W.; Goble, C. FAIRDOMHub: a repository and collaboration environment for sharing systems biology research. *Nucleic acids research* **2017**, *45*, D404–D407. doi:10.1093/nar/gkw1032.
52. Wolstencroft, K.; Owen, S.; Krebs, O.; Nguyen, Q.; Stanford, N.J.; Golebiewski, M.; Weidemann, A.; Bittkowski, M.; An, L.; Shockley, D.; Snoep, J.L.; Mueller, W.; Goble, C. SEEK: a systems biology data and model management platform. *BMC Syst. Biol.* **2015**, *9*, 33. doi:10.1186/s12918-015-0174-y.