Published in final edited form as:
Comput Chem Eng. 2021 October ; 153: . doi:10.1016/j.compchemeng.2021.107408.

PharmaPy: An object-oriented tool for the development of hybrid pharmaceutical flowsheets

Daniel Casas-Orozco[a], **Daniel Laky**[a], **Vivian Wang**[b], **Mesfin Abdi**[b], **X. Feng**[b], **E. Wood**[b], **Carl Laird**[c], **Gintaras V. Reklaitis**[a], **Zoltan K. Nagy**[a,*]

[a]Davidson School of Chemical Engineering, Purdue University, West Lafayette, IN 47906, USA

[b]Office of Pharmaceutical Quality, Center for Drug Evaluation and Research, Food & Drug Administration, Silver Spring, MD, USA

[c]Sandia National Laboratories, Albuquerque, NM 87123, USA

Abstract

Process design and optimization continue to provide computational challenges as the chemical engineering and process optimization communities seek to address more complex and larger scale applications. Software tools for digital design and flowsheet simulation are readily available for traditional chemical processing applications such as in commodity chemicals and hydrocarbon processing; however, tools for pharmaceutical manufacturing are much less well developed. This paper introduces, PharmaPy, a Python-based modelling platform for pharmaceutical manufacturing systems design and optimization. The versatility of the platform is demonstrated in simulation and optimization of both continuous and batch processes. The structure and features of a Python-based modeling platform, PharmaPy are presented. Illustrative examples are shown to highlight key features of the platform and framework.

Keywords

Digital design; Flowsheet simulation; Pharmaceutical process simulator; Hybrid processes; Parameter estimation; Optimization; Python

1. Introduction

Pharmaceutical manufacturing has traditionally been dominated by the batch operating mode, which offers flexibility in the use of multi-purpose equipment to execute the various steps in the production of active pharmaceutical ingredients (APIs). Especially when

[*]Corresponding author. zknagy@purdue.edu (Z.K. Nagy).
Author statement
Daniel Casas-Orozco: Conceptualization, Software – Development and use, Investigation, Writing – original draft, Visualization; Daniel Laky: Software – Use and review, Investigation, Writing – original draft, Visualization; Gintaras V. Reklaitis: Conceptualization, Investigation, Writing – Review and Editing, Supervision; Zoltan K. Nagy: Conceptualization, Investigation, Writing – Review and Editing, Supervision, Funding acquisition, Project administration; Carl Laird: Investigation, Writing – Review and Editing; Vivian Wang: Writing – Review and Editing, Funding acquisition, Project administration; Mesfin Abdi: Writing – Review and Editing; X. Feng: Writing – Review and Editing; E. Wood: Writing – Review and Editing.

Declaration of Competing Interest
The authors declare the following financial interests/personal relationships which may be considered as potential competing interests.

considering low-volume, high-value products, producing APIs via a sequence of batches of material provides a flexible manufacturing strategy in which after each step, intermediate material may be tested while avoiding the material losses associated with the start-up of a continuous operation. However, even with these benefits, the transition to continuous operation is gaining momentum driven by expected benefits in controllability, safety, content uniformity and consistency of product quality, reduction in manual labor and other operating and capital costs (Içten et al., 2020; Nagy et al., 2020). Along this trend, several tools have been proposed in the last years, aiming to support the analysis of dynamic end-to-end continuous processes, focusing on solid processing manufacturing systems (Dosta, 2019; Kulikov et al., 2005; Skorych et al., 2017) or more general chemical processes involving fluid phases (Tu and Rinard, 2006).

Although end-to-end continuous manufacturing could achieve these benefits, limitations of existing manufacturing facility capabilities as well as the physics and chemistry-specific constraints may dictate that the best operating mode for a given API could be either end-to-end batch, end-to-end continuous, or some combination of the two, leading to a hybrid manufacturing configuration (Boukouvala et al., 2012; Patrascu and Barton, 2018). Ultimately, the choice of the optimal manufacturing line design should be determined by process economics and the requirements of meeting the critical quality attributes (CQAs) of an API and not based on prior beliefs about which mode is best. Such design must also be developed within the quality-by-design (QbD) or quality-by-control (QbC) framework and the associated design space construct (Su et al., 2019; Yu et al., 2014).

Implementing QbD and QbC analyses typically requires use of relevant digital process models to ensure accuracy in quantitatively establishing a robust operating region or design space. For example, relatively recent publications have explored design space identification and QbD, through deterministic (Ochoa et al., 2019) and stochastic flexibility analysis (Pulsipher and Zavala, 2019). Other publications have advanced the use Monte Carlo sampling alone (García-Muñoz et al., 2010) or in combination with feasibility analysis strategies (Laky et al., 2019). Given the computational burden associated with assuring numerical convergence and the possible lack of mechanistic models that can accurately describe relevant process disturbances, some researchers have also proposed data-driven methodologies (Bano et al., 2019; Boukouvala et al., 2010; Wang and Ierapetritou, 2017). All these approaches, ranging from fully mechanistic modeling to data-driven modeling, require that the models used accurately represent the process at hand over the design space. Overall, digital design and process modeling of such pharmaceutical systems present unique computational and modeling challenges due to the rich chemistry and physics and need to meet regulatory scrutiny. These challenges are further exacerbated in dealing with hybrid manufacturing modes, which inherently combine process dynamics and systemic mathematical and operational discontinuities (Esposito and Kumar, 2007; Puigjaner et al., 2002).

There are commercial tools, such as Aspen Plus (Aspen Technology Inc., 2020) and gPROMS (Process Systems Enterprise, 2020), which are well established for flowsheet design and simulation of traditional continuous chemical processes. These tools can accommodate the routine design and flowsheet optimization of a user-provided, specific flowsheet drawing on a library of unit operations models. However, the use of these tools to systematically compare structural permutations of unit operations within such a flowsheet as well as varying operational modes requires full specification of each flowsheet instance via the graphical user interface provided by the package. Also, although many advances have been made in modeling batch and continuous processes, there is still an ongoing and unfinished effort to seamlessly and accurately simulate hybrid process flowsheets. These two deficiencies represent a major gap between the currently available commercial simulation software and the modeling requirements to encourage end-to-end optimal, digital design of pharmaceutical manufacturing processes that include non-continuous elements.

Given these gaps in commercial software packages, this work aims to address the need for flexible flowsheet design and reliable flowsheet simulation of discrete and/or hybrid manufacturing routes by introducing the open-source Python package PharmaPy for pharmaceutical manufacturing modeling. Our goal is not to replace any existing commercial packages: indeed, we encourage the use of a commercial packages, such as gPROMS FormulatedProducts. The goal is rather to efficiently compare the large number of possible discrete and continuous routes of a pharmaceutical manufacturing process without the need to concretely define each individual flowsheet alternative. PharmaPy can be utilized to systematically reduce this large set of process alternatives to a reasonably small number that can be configured and examined on an individual basis.

PharmaPy is an open-source framework implemented in Python incorporating two major thematic digital design goals. The first theme is a standardized modeling and simulation platform with the capability of parameter estimation. PharmaPy adopts a sequential-modular simulation approach with special consideration for the simulation of non-continuous process alternatives. The object-oriented nature of PharmaPy also promotes flexibility in flowsheet assembly, allowing the user a great deal of control when performing a direct, simulation-based comparison of different flowsheet configurations for a given pharmaceutical manufacturing process.

The second theme directly concerns process optimization. As a simulation platform, PharmaPy can accommodate conventional approaches to flowsheet optimization including direct grid-based searches to conduct case-driven optimization as well as simulation-based optimization using gradient free optimizers. This capability is a direct result of the generalized implementation of PharmaPy in the highly scriptable Python programming language. By keeping all functioning pieces within Python and easily accessible open-source libraries, scripting capabilities for custom, high-level analysis can be readily implemented, and collaboration facilitated. In fact, any Python tool that can use a callable function during numerical analysis may embed PharmaPy simulations within that tool. This feature will be showcased during flowsheet optimization of the case study in Section 3.3. There, a PharmaPy simulation is embedded first within a custom analysis Python code, and second within an open-source gradient-free optimization package to demonstrate the flexibility offered by choosing Python as the native language. These simulation-based optimization use cases need not be limited to traditional design optimization with or without model uncertainty. Also, PharmaPy can be embedded in a structure for optimization over multiple process configurations defined by equipment variations, effectively an enumeration strategy over the entire combinatorial space of design configurations.

This work walks through the software architecture of PharmaPy with a case study that demonstrates the direct capabilities of process modeling, simulation and parameter estimation. Once a flowsheet is realized, PharmaPy exploits the vast library of existing computational packages available to Python users for simulation and other forms of analysis. For instance, for the simulation of dynamic algebraic equation blocks, PharmaPy employs the numerical integration tool SUNDIALS (Hindmarsh et al., 2005). Flowsheets are simulated utilizing this and other tools in the sequential modular equation-solving strategy (Hillestad and Hertzberg, 1986). These simulation capabilities will be showcased with a case study in Section 3. With flowsheet simulation as a core component, the flexible scripting capabilities combined with the open-source nature of PharmaPy allow for deep numerical analysis using custom solution routines or externally available packages, for instance to conduct design space analysis or generate model and parameter uncertainty, among others. One demonstration of simulation-based optimization is shown to highlight the flexibility of PharmaPy within these analysis frameworks.

The rest of the article will be organized as follows. In Section 2, the general architecture of PharmaPy is introduced with more detailed information on specific modeling components and computational functions, and the overarching object-oriented software framework. In Section 3 the functionalities of PharmaPy will be illustrated with parameter estimation, hybrid flowsheet simulation and simulation-based optimization case studies. The paper concludes with a summary of lessons learned and comments on capabilities being pursued in ongoing work.

2. PharmaPy architecture

PharmaPy is a software tool developed in Python designed to address gaps in pharmaceutical flowsheet modeling and optimization. Python was chosen to create a widely available, collaborative, open-source package, which allows for flowsheet modeling. The open-source PharmaPy model library draws from several models for standard unit operations employed in drug substance manufacture, and in on-going work in modeling of drug product manufacture as well as providing the flexibility of accommodating customized user-created unit operation models. Flowsheet models are constructed by using several building blocks in an object-oriented software architecture, described in detail in Section 2.1 with a description of individual buildings blocks consisting of material phases (Section 2.2), unit operations (Section 2.3), and process kinetics (Section 2.4). These building blocks are linked through numerical methods for interpolation of input to and outputs from each unit operation in the sequence of dynamic unit operation models comprising a flowsheet (Section 2.5, 2.6). Finally, some of the embedded usage modes are described in Section 2.7, with the aforementioned intention of exploring these and the use of external packages for deeper analysis in future work.

2.1. Overall structure

PharmaPy is structured following an object-oriented architecture (Marquardt, 1992; Pantelides and Barton, 1993). In order to create a flowsheet, the user starts by instantiating a series of objects, as shown in Fig. 1:

- An object(s) representing material, using either a *Phase* for material holdup in equipment or a *Stream* for flowing material.
- A unit operation object.
- Optionally, a kinetic object that encapsulates parameter values for kinetic or transport mechanisms, and expressions for driving forces such as super-saturation in crystallization models.

As shown in Fig. 1, both material and kinetic instances are created separately and are added to the unit operation instance by *aggregation*. All these aggregated instances are independent of the unit operation (UO) instance and have their own attributes and methods. A UO instance can have i) no aggregated material objects: when the material comes solely from an upstream unit operation and does not need to be specified, such as when a batch reactor is followed by another batch operation ii) one material object: typically for a starting batch unit or as the initial holdup of a continuous operation iii) two or more material objects: in the case of multiphase operations such crystallization units, or for equipment that need to specify both initial holdups and entering flows (e.g. the first unit in a series of continuous UOs).

As shown in Fig. 1, several unit operation instances can be created and aggregated sequentially into a simulation instance. The simulation executive instance oversees the collection of all the unit operation instances in the first stage (square 1 in Fig. 1). In the second stage (square 2 in Fig. 1), a series of connection instances are created indicating how the material moves between UOs. Once all the flowsheet connectivity is specified by the aggregated connecting instances, a topological graph is created internally by the simulation executive, and then read by a sorting algorithm (Sridharan and Balakrishanan, 2019), in order to determine the best sequence to be used to solve each UO model contained in the flowsheet. Although for simple, strictly sequential processes the execution order is obvious, namely, the processing sequence, using the mentioned algorithm generalizes the use of PharmaPy for more complex flowsheets, e.g. with branches, and expands its capabilities for future software releases.

2.2. Material level

`Phase` and `Stream` classes allow the specification of material handled by the simulation. For initial holdups for any operating mode, different Phase classes are made available (Liquid, Solid and Vapor). On the other hand, specification of flowing material is facilitated by `Stream` classes. From the software architecture perspective, `Phase` classes are specialized versions of the `ThermoPhysicalManager` (`TPM`) parent class, from which `Phase` classes inherit characteristic functions. In turn, `Stream` classes are constructed by inheritance with respect to their corresponding Phase classes, Fig. 2. The TPM parent class contains functions for calculating thermodynamic and physical properties, typically dependent on composition and/or temperature, such as heat capacity, vapor pressure, density, viscosity and others. As PharmaPy is intended to be used at conditions typically found in the pharmaceutical industry (moderate temperatures and pressures), the offered models for property determination are those for ideal gases and ideal solutions. For conditions where the ideal solution does not hold (e.g. liquid-liquid separation or vapor-liquid equilibria), UNIFAC and UNIQUAC models (Abrams and Prausnitz, 1975; Gmehling et al., 1993) are available, which can be selected by the user when instantiating a material object. For additional property models, custom functions can be embedded into the `TPM` class or child classes, allowing to handle more specific material properties.

As seen in Fig. 1, all phase instances receive a *.json* input text file, which is arranged as a set of dictionaries containing data for each pure component involved in the simulation. Within each dictionary lie both constant values, such as molecular weight and critical properties, as well as coefficients for computing physical properties from polynomial or first-principles thermodynamic models. In order to create a material instance, a specification of material amount (mass, volume, moles), composition (mass/mole fraction or volume-based concentration) and optionally, temperature and/or pressure must be provided. In the case of solid phases, specification of crystal size distribution (CSD) is allowed by entering a vector of particle sizes, and either its corresponding volume-percentage size distribution and total crystal mass, or number-based CSD (μm^{-1}).

Mixed phases (`Slurry` or `Cake` objects) are constructed using a hierarchical material structure. In a similar fashion as shown for multiple aggregation in Fig. 1, mixed phases can be created at execution time by instantiating a slurry instance, and then aggregating liquid and solid instances previously created, as represented by the dashed lines in Fig. 2.

2.3. Unit operation level

Each UO class in PharmaPy has a signature set of methods that describe its operation by calculating material and energy balances (which may or may not require using a kinetic instance) and solving them according to the specification given by the user during instantiation. This way, several modes of operation and model complexities are handled. When a collection of models is available for a given UO, the user specifies the type of model equations to be used. For instance, different mechanistic models can be used to describe crystallization, e.g. reduced order models such as the method of moments (Randolph and Larson, 1988) as well as more detailed population balance models (Rawlings et al., 1993), which can be further divided depending on the number of internal crystal dimensions retained in the formulation. On the other hand, energy balances are switched on and off depending on whether the UO is specified to be isothermal, adiabatic, or its temperature trajectory $T(t)$ given via a Python function. This specification will determine whether the system temperature T and/or heat transfer temperature fluid T_{ht} are to be listed as UO model states (must be solved by the model), and consequently, how material and energy balances are assembled. When executed, material and/or energy balance results are concatenated by a wrapper function and sent to a method internally built into the UO class that solves the assembled balance equations by calling an appropriate numerical solver. After this, a retrieving function is called, which extracts the solution from the numerical solver and provides useful names to the disaggregated result, to be accessed by the user after simulation, or to be internally used, e.g. for internal plotting methods or data post-processing.

As described in Fig. 1, if a UO model is completely specified, it can be run in standalone mode. However, to solve a flowsheet comprised of several UOs, aggregation into a simulation instance is necessary. Then, the simulation instance calls the solution method for each unit in the appropriate order and transfers data from unit to unit, as described in Section 2.1.

Regarding model solution, PharmaPy relies on robust numerical integrators for solving dynamic models. Unit operation models are systems of ordinary differential equations (ODE), or more generally differential algebraic equations (DAE):

$$\begin{aligned} \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} &= \mathbf{f}(t, \mathbf{y}, \mathbf{z}, \mathbf{u}, \boldsymbol{\theta}), \\ \mathbf{0} &= \mathbf{g}(t, \mathbf{y}, \mathbf{z}, \mathbf{u}, \boldsymbol{\theta}), \end{aligned} \tag{1}$$

where $\mathbf{y}$ is a vector of differential states, $\mathbf{z}$ is (if present) the vector of algebraic states, $\mathbf{u}$ is the vector of input variables coming from upstream unit operations, and $\boldsymbol{\theta}$ is a vector containing the parameters of the system.

PharmaPy uses the SUNDIALS suite (Andersson et al., 2015; Hindmarsh et al., 2005), which internally calls robust backward differentiation formulas (Biegler, 2010; Hairer and Wanner, 1991) to solve the often stiff models encountered in solving pharmaceutical dynamic flowsheets. Specifically, SUNDIALS uses the CVODEs and IDAs packages for ODE and DAE systems, respectively. These packages adaptively switch between stiff and non-stiff implicit integrators as needed, capable of handling the complex dynamics of process models more effectively than their explicit counterparts. Moreover, these packages have extended sensitivity capabilities, which are of great importance for the parameter estimation components built into PharmaPy.

Given the focus of this tool on pharmaceutical processes, partial differential equations (PDEs) arise in various integral unit operations. For instance, during crystallization, population balance modeling that includes higher resolution finite volume methods must be specified in terms of PDEs. Also, when considering certain unit operating modes, for instance a plug-flow reactor, PDEs are employed to capture process dynamics. For CVODES to be employed, these PDEs must be discretized as an ODE system. Thus, in the case of partial differential equations (PDEs), all the spatial coordinates are discretized by various schemes, resulting in enlarged systems of ODEs (i.e. the so-called method of lines (Barton, 1992; Oh and Moon, 1998)).

By breaking the spatial coordinate into nodes, a simple upwind discretization for node n (Fig. 3) for a given coordinate x reads:

$$\left.\frac{\mathrm{d}\mathbf{y}(t)}{\mathrm{d}x}\right|_{n} \approx \frac{\mathbf{y}(t)\mid_{n+1} - \mathbf{y}(t)\mid_{n}}{\Delta x}. \tag{2}$$

For more numerically challenging models, such as population balance modeling for crystallization, a high-resolution discretization method is used (Gunawan et al., 2004; LeVeque, 2002). Under this formulation, the change of a given physical quantity is represented as the result of input and output fluxes $F_{n \pm 1/2}$ to a finite volume indexed as n, Fig. 3:

$$\frac{\mathrm{d}\mathbf{y}_n(t)}{\mathrm{d}x} \approx \frac{F_{n+1/2} - F_{n-1/2}}{\Delta x}. \tag{3}$$

For example, an upward flux has the following form under the high-resolution method $(f_n = y(t)\mid_n)$:

$$F_{n-1/2} = \left[f_{n-1} + \frac{1}{2}(f_n - f_{n-1})\varphi_{n-1}\right], \tag{4}$$

where φ_n is a limiter function that provides enhanced numerical stability, and is dependent on the slope of f_n with respect to the studied spatial coordinate (LeVeque, 2002) . It can be seen that if $\varphi_n = 0$, the upwind discretization scheme is recovered in Eq.(2).

Sparse jacobians are expected after reformulating a system of PDEs as a system of ODEs. For this reason, the generalized minimal residual GMRES (Saad and Schultz, 1986) linear method is used, which is proven to be very efficient for large, sparse systems.

2.4. Kinetics level

Different classes are available in PharmaPy for the construction of kinetic instances intended for specific physical or chemical phenomena. The modularity offered by this approach makes it possible to add a specific behavior to a unit operation (reaction kinetics to a batch reactor), and allows to extend the functionality of PharmaPy for more complex phenomena such as reactive crystallization. As with Phase and UO instances, kinetic objects are instantiated from the available kinetic classes by entering rate parameters and any other constitutive equations of interest, depending on the modeled phenomena. The ubiquity of reaction and crystallization operations in pharmaceutical processes motivated the construction of classes to support these particular mechanisms.

The reaction kinetics class supports mechanisms where reaction rate r_i for reaction i is conceptualized as the product of a temperature-dependent term (k_i) and a concentration-dependent term f_i, each one with its corresponding subset of parameters $\boldsymbol{\theta}_k$ and $\boldsymbol{\theta}_f$:

$$r_i = k_i(T, \boldsymbol{\theta}_k) \cdot f_i(\boldsymbol{C}, \boldsymbol{\theta}_f), \tag{5}$$

$$\boldsymbol{\theta} = \{\boldsymbol{\theta}_k, \boldsymbol{\theta}_f\}\,. \tag{6}$$

Temperature-dependent constant k_i is described by the Arrhenius equation:

$$k_i = k_{i0} \cdot \exp\left(-\frac{E_{a,i}}{RT}\right), \tag{7}$$

whereas the concentration-dependent term, in the case of elementary reactions, is given by:

$$f_i = \prod_j C_j^{\alpha_j} \quad \text{for} \quad \nu_{i,j} < 0, \tag{8}$$

with reaction orders α_j for reactant species in reaction i (those with negative stoichiometric coefficient $\nu_{i,j}$).

For reversible reactions:

$$r_i = k_i\left(\prod_{j \mid \nu_{i,j} < 0} C_j - \frac{1}{K_C}\prod_{j \mid \nu_{i,j} > 0} C_j\right). \tag{9}$$

For reaction rate equations different from elementary kinetics, PharmaPy allows the specification of a user-defined python function for the concentration-dependent term f_i, which extends the reaction simulation capabilities to include different reaction mechanisms such as catalytic reaction rate equations (Casas-Orozco et al., 2018).

In order to facilitate parameter estimation, Eq. (7) can be reformulated as a centered version with respect to a reference temperature T_{ref} (Bilardello et al., 1993; Watts, 1994), Eq.(10).

$$k_i = \exp\left[\varphi_{1i} + \exp\left(\varphi_{2i}\right)\left(\frac{1}{T_{ref}} - \frac{1}{T}\right)\right], \tag{10}$$

where the new defined parameters $\varphi_{1,i}$ and $\varphi_{2,i}$ are given by:

$$\varphi_{1i} = \ln(k_{i0}) - \frac{E_{a,i}}{RT_{ref}}, \tag{11}$$

$$\varphi_{2i} = \ln\left(\frac{E_{a,i}}{R}\right). \tag{12}$$

The reformulations shown in Eqs. (11) and (12) constrain the original kinetic parameters to be positive, and also avoid ill-conditioning of the jacobian with respect to the parameters k_0 and E_a. Under this new formulation, the parameters sets are:

$$\begin{aligned}\boldsymbol{\theta}_k &= \{\varphi_{1,1}, \varphi_{1,2}, \ldots, \varphi_{1,n_{rxns}}, \varphi_{2,1}, \varphi_{2,2}, \ldots, \varphi_{2,n_{rxns}}\},\\ \boldsymbol{\theta}_f &= \{\alpha_1, \alpha_2, \ldots\}.\end{aligned}$$

Crystallization kinetics are also supported by PharmaPy in the form of power law expressions for nucleation, growth and dissolution, which depend on absolute or relative supersaturation, Eqs (13) and (14), respectively (Szilagyi et al., 2020).

$$S = C - C_{sat} \tag{13}$$

$$S = \frac{C - C_{sat}}{C_{sat}} \tag{14}$$

where C_{sat} represents the temperature-dependent saturation concentration of the crystallizing compound as given by solubility measurements.

Nucleation rate is expressed as the sum of primary and secondary nucleation rates:

$$B = B_p + B_s, \tag{15}$$

$$B_p = k_p \exp(-E_p / RT) S^p, \tag{16}$$

$$B_s = k_s \exp(-E_s / RT)\, S^{S_1} (k_v \mu_3)^{S_2}, \tag{17}$$

$$B = \max(B, 0), \tag{18}$$

where the crystal size distribution third moment μ_3 is calculated from the numerical solution of particle size (Section 3.2), and k_v represents the crystal volumetric shape factor. Primary nucleation parameter set is $\boldsymbol{\theta}_p = \{k_p, E_p, p\}$, whereas for secondary nucleation $\boldsymbol{\theta}_s = \{k_s, E_s, s_1, s_2\}$.

Growth and dissolution rates are expressed similarly:

$$G = \max(k_g \exp(-E_g / RT) S^g, 0) \tag{19}$$

$$D = \min(k_d \exp(-E_d / RT) S \mid S \mid^{d-1}, 0) \tag{20}$$

with parameter sets for growth and dissolution given by $\boldsymbol{\theta}_g = \{k_g, E_g, g\}$ and $\boldsymbol{\theta}_d = \{k_d, E_d, d\}$, respectively.

The operators *max* and *min* in Eqs. (18), (19) and (20) are used to capture the change from crystallization to dissolution regimes, depending on the value of supersaturation.

2.5. Material transfer policies

Different material transfer policies are established in order to support a variety of operation modes, viz. end-to-end batch, end-to-end continuous or hybrid. In general, the material exiting a batch or semi-batch unit is assumed to be transferred entirely with or without time delay to the next connected unit.

Two operation modes are possible with respect to input material regarding semi-batch UOs, depending whether or not material instance was added to the UO at instantiation. If the unit is empty, the incoming material from the upstream UO is assumed to be transferred completely to fill its initial holdup (Fig. 4a), and an entering stream must have been added during instantiation to reflect the continuous feed of material. However, if the unit has an initial holdup, then the material coming from the upstream UO is assumed to enter the unit as a continuous flow, and consequently, a flowrate value must be provided when specifying its corresponding connection (Fig. 4b).

With continuous UOs, different transfer scenarios are possible. For end-to-end continuous flowsheets, all connection objects contain Stream instances, thus the material flow is continuous from unit to unit. On the other hand, with hybrid flowsheets, one scenario can occur when a continuous operation is preceded by one or more discontinuous UOs (Fig. 5a). In this case, material is assumed to flow from either the last discontinuous unit, or, if the discontinuous unit is to be emptied to be used for another task, from a Mixer, a special UO present in the PharmaPy library designed to hold material from different sources. Mixers are considered adiabatic and serve to support batch operation and continuous operation, i.e. one or more batch of materials are added to the Mixer or may be fed from continuously flowing inputs, respectively.

The second scenario for hybrid flowsheets arises when a batch/semi-batch unit is preceded by a continuous unit, as depicted in Fig. 5b. In order to decouple the two sections of the flowsheet, the material coming from the continuous unit must be continuously received in a special type of container known as a `DynamicCollector`, also present in the PharmaPy library and added by the user when creating a process flowsheet. For instance, when material is received from a mixed-suspension-mixed-product-removal (MSMPR) continuous crystallizer, the `DynamicCollector` acts as an adiabatic semi-batch crystallizer, which permits the accurate representation of the dynamics of the material holdup as a function of time.

2.6. Connection strategy

In order to connect two UOs, two main tasks must be executed. The first task is to determine the one-to-one relations between upstream and downstream state names describing the same physical attribute. For instance, the composition might be given in molar concentration units for a given upstream UO (e.g. a reactor) but in molar fractions for the downstream UO (e.g. a vaporizer). Thus, these names must be reconciled with each other for data transfer purposes. The second task is related to the sequential-modular architecture of PharmaPy, which numerically integrates each unit operation model one by one, thus requiring an intermediate calculation to match their corresponding time grids.

2.6.1. Name-matching algorithm

The first task in the connection strategy is accomplished by creating a bipartite graph (Bogusch and Marquardt, 1997; Braunschweig and Gani, 2002), such as the one depicted in Fig. 6. This is constructed as a Python dictionary by exploring the list of output and input names of each of the related unit operations, and recognizing four categories: i) composition names: e.g. mass/molar fraction, mass/molar concentration, ii) amount names: viz. *mass, moles* or *vol*, iii) distribution names: number-based or volume-based particle distribution, and iv) flow names: e.g. mass/volumetric flow. Names contained in the output/input sets are checked against the aforementioned categories, and paired when they appear on the same category in both UOs, Fig. 6.

After this matching is established, the second step is unit conversion of the values of the upstream UO, if needed. For this purpose, a subset of the states solved by the ODE/DAE integrator must be extracted, since in general, not all the UO states are output states (e.g. liquid volume in a well stirred continuous tank is not an output state, whereas temperature is). Then, the selected states are passed to the corresponding unit conversion method of the receiving material object (Phase or Stream). Unit conversion is performed making extensive use of vectorized operations to enhance numerical performance. In the case of volumetric conversions (molar or mass concentration), ideal solution is assumed (Rawlings and Ekerdt, 2002):

$$\sum_j C_j \upsilon_j = 1 \tag{21}$$

with C_j: molar concentration of component j (mol L^{-1}) and υ_j: molar volume of component j (L mol^{-1})

2.6.2. Interpolation algorithm

The second task of the connectivity component of PharmaPy is the interpolation algorithm, which makes part of the classes in the *Stream* module, . The need of such a calculation is shown in Fig. 7, where a given output u from the upstream UO, solved at a series of time points, is to be used by the connected downstream UO. In general, sampling times of two connected UOs differ, since numerical integration takes place independently for each UO, resulting in different internal time steps and evaluation times. Differences in time grids for process models solved independently arise from the different timescales of the phenomena being modeled, which can be slow (requiring longer time steps) for one UO and fast for another (shorter time steps) within the same integration period. This sampling time mismatch is alleviated by using interpolation of the time series returned when solving the upstream UO.

Two interpolation schemes are used. For the instantaneous numerical integration of the downstream UO (a single time value is passed to the interpolation function), three sampling points of the upstream UO timeseries are selected, such that the time value selected by the integrator for the downstream UO is contained (Fig. 7). Then, a Newton interpolation formula (Press et al., 2001) is used to build a local interpolation model that evaluates the states at the time value requested by the numerical integrator.

On the other hand, when the inputs to the downstream UO are to be evaluated at multiple time values, e.g. for calculating heat duty from the energy balance solution, Newton interpolation formulas are no longer used, given their natural oscillations at high polynomial orders. Instead, the cubic spline capabilities of Scipy (Virtanen et al., 2020) are used to efficiently evaluate inputs at several time values simultaneously.

2.7. Usage modes

In order to provide different functionalities, from the analysis of flowsheets to the generation of digital twins, several usage modes are available in PharmaPy. These currently include parameter estimation, flowsheet simulation and flowsheet optimization.

Parameter estimation is facilitated by the `ParameterEstimation` class in PharmaPy, which offers both unconstrained optimization via Levenberg-Marquardt algorithm (Marquardt, 1963; Nielsen, 1999) and constrained optimization through IPOPT (Wächter and Biegler, 2006). The `ParameterEstimation` class receives a Python function that encodes the model of interest, and datapoints of the measured independent variable and any number of dependent variables. In general, the provided function should depend on the parameters, and on a given independent variable (time, spatial position). Gradient of the objective function with respect to the parameters is calculated either by finite differences or by a user-defined function.

In order to communicate UO models and the parameter estimation platform, each UO class in PharmaPy has a method that wraps its corresponding solution method to meet the required model structure by the `ParameterEstimation` class. This wrapper also performs any additional calculations that might be needed before passing the state variable and derivative information. Additionally, the parameter seed values can be automatically retrieved by PharmaPy from a kinetic instance attached to a given UO, or can be independently provided by the user to the `ParameterEstimation` instance.

Given that many applications of parameter estimation involve the use of dynamic models, time-dependent sensitivities $\mathbf{S} \equiv \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\boldsymbol{\theta}}$ and $\mathbf{R} \equiv \frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\boldsymbol{\theta}}$, when information to calculate them is available, are used to solve for the jacobian of the objective function with respect to the parameters (Bard, 1974; Bates and Watts, 1988), Eq. (22)

$$\begin{aligned}\frac{\mathrm{d}\mathbf{S}}{\mathrm{d}t} &= \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{y}}\mathbf{S} + \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{z}}\mathbf{R} + \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\boldsymbol{\theta}},\\ \mathbf{0} &= \frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{y}}\mathbf{S} + \frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{z}}\mathbf{R} + \frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\boldsymbol{\theta}},\end{aligned} \tag{22}$$

with $\mathrm{d}\mathbf{f}/\mathrm{d}\mathbf{y}\,(\mathrm{d}\mathbf{g}/\mathrm{d}\mathbf{y})$ and $\mathrm{d}\mathbf{f}/\mathrm{d}\boldsymbol{\theta}\,(\mathrm{d}\mathbf{g}/\mathrm{d}\boldsymbol{\theta})$ the jacobian matrices of the right-hand side of Eqs. (1) with respect to the states and with respect to the parameters, respectively. In PharmaPy, these jacobians are computed from their analytical forms for elementary reaction kinetic mechanisms, under the reparametrization shown in Section 3.1. For the method of moments in crystallization models (Section 3), a series of analytical derivatives are also provided (Nagy and Braatz, 2007). For other kinetic mechanisms, automatic differentiation via Autograd (Maclaurin et al., 2020) is used to obtain reliable derivative information.

PharmaPy also is a flexible instrument while performing flowsheet optimization via parameter sweeps and other simulation-based strategies. For sample-based optimization techniques, the Pythonic nature of PharmaPy allows for analysis to be parallelized through packages such as MPI for Python (Dalcín et al., 2005), significantly improving computational flexibility for users limited only by the availability of HPC resources. Given the high amount of user control, implementing complex and unique uncertainty schemes on both operational and model parameters, users can readily tailor existing or novel solution techniques around flowsheet simulation and optimization. More direct optimization approaches are available by utilizing gradient-free optimization packages, such as Nevergrad (Rapin and Teytaud, 2018), inside user-created solution algorithms.

There are also plans for other usage modes that have yet to be implemented. Discussion of these plans and other future implementations in PharmaPy is available in Section 4.

3. Case studies

In this section, the three main functionalities of PharmaPy, namely, parameter estimation, process simulation and process optimization, are shown through a case study involving an API manufacturing process. For illustrative purposes, the reaction system shown in Eq. (24) is used, where reactants A and B participate in two reactions yielding C and D, with C the API of interest and D an undesired side-product.

$$\begin{array}{c} A + B \rightarrow C \ (1), \\ B + C \rightarrow D \ (2), \end{array} \tag{23}$$

All models were run on an Intel Core i7 Windows 10 machine, with 3.0 GHz and 16 GB of RAM memory.

3.1. Parameter estimation

Parameter estimation capabilities in PharmaPy are illustrated by fitting a set of batch, isothermal reaction datasets to the reaction mechanism in Eq. (1.24). As explained in Section 2.7, analytical jacobians for elementary reaction models are implemented in PharmaPy. For this purpose, material balances for a batch reaction system can be conveniently represented in matrix form as:

$$\frac{\mathrm{d}\mathbf{C}}{\mathrm{d}t} = \mathbf{St}^{T} \cdot \mathbf{r}(\mathbf{C}, T, \boldsymbol{\theta}) \tag{24}$$

Where $\mathbf{r} = [r_1, \cdots, r_{n_{rxn}}]$ is a vector containing reaction rates r_i as given in Eq. (5), and **St** matrix a $n_{rxn} \times n_{comp}$ matrix containing stoichiometric coefficients $\nu_{i,j}$.

The representation shown in Eq. (24) permits the required jacobians for parameter estimation to be expressed in a straightforward computation. The jacobian of the right-hand side of Eq. (24) with respect to the states, for the isothermal case, reads:

$$\mathbf{J}_{\mathrm{C}} = \mathbf{St}^{T} \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\mathbf{C}} \tag{25}$$

with the elements in matrix $\mathrm{d}\mathbf{r} / \mathrm{d}\mathbf{C}$ given by:

$$\frac{\partial r_i}{\partial C_j} = k_i \alpha_j \frac{\prod_k C^{\alpha_k}}{C_j}, \qquad \forall\ i \in \{1, \ldots, n_{rxn}\},\ j \in \{1, \ldots, n_{species}\} \tag{26}$$

Similarly, the jacobian with respect to parameters is given by:

$$\mathbf{J}_{\theta} = \mathbf{St}^{T} \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\boldsymbol{\theta}} \tag{27}$$

The derivatives necessary to build matrix $\mathrm{d}\mathbf{r} / \mathrm{d}\boldsymbol{\theta}$ are:

$$\frac{\partial r_i}{\partial \varphi_{1,i}} = r_i, \qquad \forall\, i \in \{1, \ldots, n_{rxn}\} \tag{28}$$

$$\frac{\partial r_i}{\partial \varphi_{2,i}} = r_i \left(\frac{1}{T_{ref}} - \frac{1}{T} \right) \exp\left(\varphi_{2,i}\right), \qquad \forall\, i \in \{1, \ldots, n_{rxn}\} \tag{29}$$

$$\frac{\partial r_i}{\partial \alpha_k} = \ln\left(C_k\right) r_i \qquad \forall\, i \in \{1, \ldots, n_{rxn}\},\; k \in \{1, \ldots, n_{ord,i}\} \tag{30}$$

where $n_{ord,i}$ represents the number of non-zero reaction orders for reaction i.

Simulated reaction data at three different temperatures (288 K, 298 K and 308 K) and with initial concentrations $C_A = C_B = 1 \text{ mol L}^{-1}$ were generated for the reaction system shown in Eq. (23), using the batch reactor model, Eq. (24), and the nominal parameter values shown in Table 1. Normal noise $N \sim (0, 0.01)$ was added to the resulting concentration timeseries in order to convert it to the reference dataset used by the parameter estimation platform.

A flowchart of the generation of PharmaPy objects for parameter estimation is shown in Fig. 8. As shown in the Fig., UO object (batch reactor) receives material and kinetics instances by aggregation. Then, the UO object can be in turn aggregated to a simulation object, in order to unlock further capabilities including parameter estimation. Initial reaction conditions are specified in a `LiquidPhase` object and consist of the amount, concentration and temperature of the initial liquid loaded to the reactor. The `LiquidPhase` object also holds a reference to a file containing relevant material thermophysical properties for pure components. On the other hand, a `RxnKinetics` object is then instantiated and the nominal kinetic parameters shown in Table 1 are passed in along with a stoichiometric matrix representing Eqs. (23). A batch reactor object is also instantiated, specifying which species contained in the pure component database participate in the reaction (have entries in the stoichiometric matrix). The phase and kinetic data are then embedded in the batch reactor object, as explained above (Fig. 8). A further aggregation of the specified reactor model to a simulation instance completes the requirements to build the PharmaPy model.

Aggregating the unit operation model to the simulation instance permits two important functions: (1) to execute the reactor model in standalone form (for data generation purposes), and (2) to facilitate the parameter estimation problem. For the latter, the simulation object offers an interface that allows specification of experimental data (in this case the one generated with the model), initial parameter seeds, solver algorithm options (for UO model and optimizer), and, if more than one experimental dataset is provided, the conditions at which the unit operation model should be specified for each unique data set. For the present case, this information is provided as phase modifiers, which set the initial state of the phase or each of the provided datasets at the three different temperatures.

A sampling interval of 60 seconds over the course of 1-hour experiment was assumed for the three synthetic datasets. The corresponding conversions of B for the three tested temperatures were 73.4%, 82.5% and 89.4%, respectively. A reference temperature $T_{ref} = 303\,\mathrm{K}$ was used for the exponential term of the rate equation.

Results of the parameter estimation are shown in Fig. 9, where all the three datasets were used to determine k_i and $E_{a,i}$ values simultaneously. As observed, a rather poor model prediction is obtained using the initial estimate parameters, with the reaction going to complete conversion before 15 min for each of the three datasets. This is mainly caused by the relatively low initial activation energies used, Table 1. The Levenberg-Marquardt algorithm is successful in fitting the data after 10 iterations, with a total run time of 0.64 seconds wall clock time.

In order to test the robustness of the parameter estimation algorithm, $E_a = [2.5 \times 10^4, 2.5 \times 10^4]$ values were used as initial estimates. Unlike the original estimates in Table 1, the total reaction conversion for this parameter estimate is much lower than the data (a slower reaction speed). The effect of changing the initial parameter values on the initial parametric sensitivity of the reaction model is shown in Fig. 10, using the plotting capabilities provided by the PharmaPy reactor model. A significant drop in parametric sensitivity is observed after increasing the initial activation energy estimates. This change is verified by examining the parameter condition number cn (Eq. (32)) at the beginning of the optimization run, which exhibits an order of magnitude increase from 1568 for the first parameter estimate to 15 325 for the second parameter estimate

$$cn = \frac{\varsigma_1}{\varsigma_n}, \tag{31}$$

with ς_1 and ς_n the first and last singular values of the Hessian approximation of the Levenberg Marquardt algorithm $\mathbf{A} = \mathbf{J}^T\mathbf{J}$.

Consequently, more iterations and computation time are required to estimate the parameters (14 iterations and 2.2s instead of 10 iterations and 0.64s). Furthermore, evaluating an initial estimate with high enough activation energy, $E_a = [3 \times 10^4, 3 \times 10^4]$, significantly increases the initial condition number $cn = 7 \times 10^5$, which renders the parameter estimation infeasible.

The conditioning analysis available in PharmaPy is a valuable tool for a user to inspect the seed values used in parameter estimation, and to analyze the conditioning of the system that leads to a stable parameter estimation run. This ultimately will increase the chances of a successful run and will also impact the amount of user and computational time spent performing parameter estimation.

Parameter estimation analysis is completed by determining the asymptotic confidence intervals of the regressed parameters, using the statistical platform offered by the simulation instance. Confidence intervals are calculated as:

$$\hat{\boldsymbol{\theta}} - t_{\alpha/2,dof}\hat{\boldsymbol{\sigma}}_\theta < \boldsymbol{\theta} < \hat{\boldsymbol{\theta}} + t_{\alpha/2,dof}\hat{\boldsymbol{\sigma}}_\theta, \tag{32}$$

where $dof = n_{data} - n_{params}$ and $t_{\alpha/2,dof}$ is the critical value of a Student t-distribution for the given confidence interval α, and the parameter standard deviation given by

$$\hat{\boldsymbol{\sigma}}_\theta = \sqrt{\text{diag}(\text{cov}_\theta)}\,. \tag{33}$$

Parameter covariance matrix is computed in terms of the residual **res** and the Hessian approximation **A**:

$$\text{cov}_\theta = \frac{\mathbf{res}^T \cdot \mathbf{res}}{dof}\mathbf{A}^{-1}\,. \tag{34}$$

Confidence intervals are shown in Table 2. The values obtained for the transformed parameters $\varphi_{1,i}$ and $\varphi_{2,i}$ are symmetric, but are asymmetric when translated to the corresponding physical model parameters, as observed in Table 2.

3.2. Flowsheet analysis

Simulation capabilities of PharmaPy are demonstrated with the analysis of a hybrid batch/continuous process for the synthesis, crystal formation, and solid separation of an API. The proposed process involves continuous and discontinuous UOs, comprising single phase UOs (reactor and liquid collector) and multiphase UOs (crystallizer and filter). In this process, an API is continuously synthesized in a plug flow reactor (R01) and subsequently crystallized in an unseeded batch crystallizer (CR01) using a linear cooling profile. The material from CR01 is then filtered batch-wise in a filtration unit (F01). To decouple R01 and CR01, an intermediate holding tank (HOLD01) gathers material output from the PFR to be later distributed to the batch crystallization unit, as shown in Fig. 11.

The models used for the process flow diagram in Fig. 11 are described next. First, plug flow reactors (PFRs) are modelled under the assumptions of no radial gradients, negligible axial dispersion and constant heat transfer temperature T_{ht}, below in Eqs. (35):

$$
\begin{gathered}
\frac{\partial C_j}{\partial t} = -\dot{F}_{in}\frac{\partial C_j}{\partial V} + \sum_i \nu_{i,j} \cdot r_j \qquad \forall\ j \in \{1, \ldots, n_{comp}\} \\
\sum_{j=1}^{n_{comp}} C_j C_{p,j}\frac{\partial T}{\partial t} = -\dot{F}_{in}\sum_{j=1}^{n_{comp}} C_j C_{p,j}\frac{\partial T}{\partial V} + \sum_{i=1}^{n_{rxn}} \Delta H_{rxn,i}(T) r_i - Ua(T - T_{ht}),
\end{gathered} \tag{35}
$$

where V is the volume coordinate (m^{-3}), $\dot{F}_{in}$ is the inlet flowrate (m^3 s^{-1}), U is the heat transfer coefficient (W m^{-2} K^{-1}), a is the heat transfer area (m^2 m^{-3}), $\Delta H_{rxn,i}$ is the heat of reaction for reaction i (J mol^{-1}) and $C_{p,j}$ is the heat capacity of component j (J kg^{-1} K^{-1}), given by:

$$
C_{p,j}(T) = A_j + B_j T + C_j T^2 + \ldots \tag{36}
$$

Upon upwind discretization of the V dimension, the reaction model, Eqs. (35), is converted into a set of ODEs, Eqs. (35):

$$
\begin{gathered}
\frac{\mathrm{d}C_j^n}{\mathrm{d}t} = -\dot{F}_{in}\frac{C_j^n - C_j^{n-1}}{\Delta V} + \sum_i \nu_{i,j} \cdot r_i\left(C_1^n, \ldots, C_{n_{comp}}^n, T^n\right), \\
\forall\ j \in \{1, \ldots, n_{comp}\},\ n \in \{1, \ldots, n_{discr}\} \\
C_{p,vol}\frac{\mathrm{d}T^n}{\mathrm{d}t} = -\dot{F}_{in}C_{p,vol}\frac{T^n - T^{n-1}}{\Delta V} + \sum_{i=1}^{n_{rxn}} [\Delta H_{rxn,i}(T^n) \cdot r_i] \\
- Ua(T^n - T_{ht}), \qquad \forall\ n \in \{1, \ldots, n_{discr}\}
\end{gathered} \tag{37}
$$

with the volumetric heat capacity $C_{p,vol}$:

$$
C_{p,vol} = \sum_{j=1}^{n_{comp}} \left[C_j^n \cdot C_{p,j}(T^n)\right]. \tag{38}
$$

For CR01, a one-dimensional population balance with size-independent growth is used, Eq. (39).

$$
\begin{gathered}
\frac{\partial f(t,x)}{\partial t} + G\frac{\partial f(t,x)}{\partial x} = B \cdot V \cdot \delta(L - L_0),\ S > 0 \\
\frac{\partial f(t,x)}{\partial t} - D\frac{\partial f(t,x)}{\partial x} = 0,\ S < 0,
\end{gathered} \tag{39}
$$

where f represents the crystal size distribution (CSD) of the solids present in suspension (μm^{-1}) and x represents the crystal size coordinate (μm). As explained in Section 2.3, a high-resolution scheme was adopted to discretize the space coordinate x in Eq. (39) . Thus, the change in crystal size distribution (CSD) can be written as the following ODE system ($n \in \{1, \cdots, n_{discr}\}$):

$$\frac{\mathrm{d}f_n}{\mathrm{d}t} = \frac{F_{n+1/2} - F_{n-1/2}}{\Delta x},$$

(40)

with the fluxes F given by:

$$\begin{aligned} F_{1-1/2} &= B \cdot V, \\ F_{n-1/2} &= G\left[f_{n-1} + \frac{1}{2}(f_n - f_{n-1})\varphi_{n-1}\right] \\ &\quad + D\left[f_n + \frac{1}{2}(f_{n-1} - f_n)\varphi_n\right], \qquad \forall\, n \in \{1, \ldots, n_{discr}\} \\ F_{n_{discr}+1/2} &= 0\,. \end{aligned}$$

(41)

Van-Leer flux limiters φ_n are used in the present study

$$\begin{aligned} \varphi_n &= \frac{|\theta_n| + \theta_n}{1 + |\theta_n|}, \\ \theta_n &= \frac{f_n - f_{n-1}}{f_{n+1} - f_n}\,. \end{aligned}$$

(42)

The crystallization model is closed with a material balance for the liquid phase:

$$\begin{aligned} \frac{\mathrm{d}C_j}{\mathrm{d}t} &= 3\,k_v\,\rho_{cry}\,\mu_2 G\left(\delta_{j,tg} - \frac{C_j}{\rho_{liq}}\right), \qquad \forall\, j \in \{1, \ldots, n_{comp}\} \\ \frac{1}{\rho_{liq}} &= \sum_j \frac{w_j}{\rho_j}, \end{aligned}$$

(43)

with ρ_{cry} the density of crystals (kg m^{-3}), w_j the mass fraction of species j in solution, ρ_j the density of pure component j (kg m^{-3}), and $\delta_{j,tg}$ the Kronecker delta relating species j and the target crystallizing component indexed as tg ($j \in \{1 \ldots, n_{comp}\}$):

$$\delta_{j,tg} = \begin{cases} 1, & j = tg \\ 0, & j \neq tg \end{cases}$$

(44)

The nth moment of the CSD, f, can be calculated as:

$$\mu_n = \int_0^\infty x^n f(x,t)\,\mathrm{d}x,$$

(45)

which can be computed numerically from the model solution given a specified size grid for the x dimension.

HOLD01, receiving material from R01, can be described by the following system of ODEs:

$$\begin{aligned}\frac{\mathrm{d}w_j}{\mathrm{d}t} &= \frac{\dot{m}_{in}}{m_{tot}}(w_{j,in} - w_j), \qquad \forall\ j \in \{1, \ldots, n_{comp}\}\\ \frac{\mathrm{d}m_{tot}}{\mathrm{d}t} &= \dot{m}_{in}(t),\\ \frac{\mathrm{d}T}{\mathrm{d}t} &= \frac{\dot{m}_{in}}{m_{tot}\sum_j C_{p,j}w_j}(h_{in}(T_{in}) - h(T)),\end{aligned} \tag{46}$$

where $\dot{m}_{in}$ is the inlet mass flowrate (kg s^{-1}) and m_{tot} is the instantaneous mass contained in the tank (kg). Specific enthalpy is given by:

$$h(T) = \sum_{j=1}^{n_{comp}} w_j \left[\int_{T}^{T_{ref}} C_{p,j}\mathrm{d}T\right] \tag{47}$$

Finally, the filtration model for unit F01 is given by:

$$\frac{\mathrm{d}m_f}{\mathrm{d}t} = \frac{\Delta P}{\mu_{liq}\left(\frac{\alpha_{cake}C_s}{A_f^2\rho_{liq}^2}m_f + \frac{R_f}{A_f\rho_{liq}}\right)}, \tag{48}$$

with α_{cake} the average cake resistance (m kg^{-1}), R_f the filtration medium resistance (m^{-1}), A_f the cross sectional area of the filter (m^2), μ_{liq} the liquid phase viscosity (Pa s), and ΔP the delta pressure exerted on the filtration system (Pa). Solid concentration C_s and filtrate volume V_f are given by:

$$\begin{aligned}C_s &= \frac{V_{sl}C_{sl}}{V_f},\\ V_f &= V_{sl}\left[\varepsilon - \frac{C_{sl}}{\rho_{cry}}\left(\frac{\varepsilon}{1-\varepsilon}\right)\right].\end{aligned} \tag{49}$$

where C_{sl} is the solids concentration in the slurry ($kg_{cry}m_{sl}^{-3}$) and V_{sl} the slurry volume (m^3). Cake porosity ε is given by the Ouchiyama-Tanaka model (Ouchlyama and Tanaka, 1986), and mean specific cake resistance α_{cake} is given by (Bourcier et al., 2016):

$$\alpha_{cake}(t) = \frac{\int_0^\infty \left[f(t,x)\frac{180\,(1-\varepsilon)}{\varepsilon^2 x^2 \rho_{cry}}\right]\mathrm{d}x}{\int_0^\infty f(t,x)\mathrm{d}x}. \tag{50}$$

All general models are contained in the unit operation model library within PharmaPy. Using this library, a process is specified by declaring process kinetics, unit operation design specifications, and operating conditions, following the software architecture described in Sections 2.1, 2.2, and 2.3. The API is synthesized using the reaction network described by Eqs. (23) and associated nominal parameter values in Table 2. Reaction model is solved for a grid of 100 equally spaced volume elements, whereas crystallization uses 1 μm increments for the x coordinate, for a 1-1000 μm range. Crystallization kinetics for this case include primary and secondary nucleation, Eqs. (15)-(18), and growth and dissolution, Eqs. (19) and (20). These phenomena are represented by the parameters described in Table 3. For crystallization, the saturation concentration was modeled as a second order polynomial function in temperature, as shown below in Eq. (51):

$$C_{sat} = A + BT + CT^2 \tag{51}$$

Moreover, R01 is initially loaded with pure solvent at 298 K and HOLD01 is empty. After being specified within PharmaPy, the continuous section of the process (R01 and HOLD01) was simulated for 7 hours, and then CR01 was run for another 7 hours. Finally, F01 is run until the liquid mass m_f reduces to a value equivalent to the amount of liquid retained by the pores of the filtration cake. This latter operation mode is implemented by setting a state event for the dynamic solution of F01. A visual representation of processing times is shown in Fig. 12.

The results for API concentration with respect to position along the PFR are shown in Fig. 13 (left), where it is evident that steady state is reached after approximately 35 minutes of simulation. The reaction volume is set such that API concentration is maximum at the end of the reactor, i.e. a longer reactor would result in degradation of desired API C to undesired side-product D.

The results for HOLD01 are seen in Fig. 13 (right). Here, material from start-up until simulation end is sent to the vessel, including the initial solvent-only contents of R01. This results in a dilution effect, as it is only after 20 minutes that material with significant concentration of reactants and products from R01 reaches HOLD01. Fig. 13 also shows how different sets of composition units are used in PharmaPy depending on the requirements imposed by the material and energy balances. In accordance with the connectivity strategy (Section 2.6.1) PharmaPy links the UO models by converting upstream information from the last volume element of R01 this way: material flow from volumetric to mass units, and composition from mole concentration to mass fraction.

API concentration profile and solubility profile are plotted versus time for CR01, expressed in mass concentration (kg m^{-3}) given the units of the provided solubility curve, Eq. (44). After approximately 19 minutes, material begins crystallizing due to supersaturation resulting from cooling, accompanied by a quick depletion of API supersaturation, resulting from relatively fast growth kinetics. A rapid increase in the total number of crystals can also be observed in Fig. 14 (right), typical of crystallization systems, and correctly captured by the integration algorithms utilized, which are equipped to deal with highly stiff ODE systems. At the end of the crystallizer operation, a total of 0.83 dm^3 of solids are obtained, or an equivalent 1.18 kg of API crystals.

On the other hand, CSD, an important critical quality attribute affecting downstream unit operations such as filtration-drying and drug product UOs is shown in Fig. 15. It is seen that CSD is centered at higher particle sizes as time progresses, which directly translates in an increase of mean average size μ_1 / μ_0 up to 33 μm at the end of the 7 hours of operation.

F01 results are shown in Fig. 16, for a filter diameter of 20 cm. Parabolic profiles for filtrate mass are observed in Fig. 16 (left), and the consolidation of the crystal cake in Fig. 16 (right). Porosity model predicts a final cake with 27 wt% liquid, to be sent either to a deliquoring step or a drying process.

The complete consolidated results are shown in Table 4. PharmaPy offers a summary table in mass or molar units, which describes the phases present in the connecting streams, as well as their most important descriptors, for the end state of the system. The dynamic trajectories of each of the UOs can be independently extracted and plotted with convenience functions built into each UO. Furthermore, the hybrid nature of the studied process flowsheet is reflected in the specification of flowrate and mass depending on the UO in question. Overall, 37.5 kg of materials are consumed in the manufacturing of 1.18 kg of C (dry cake), with a remaining 1.43 wt% of C in the liquid phase after crystallization, for a crystal yield of 68.4%.

This flowsheet showcases several PharmaPy capabilities: support for distributed, dynamic UO models (PFRs), capacity to decouple continuous and discontinuous sections for a given hybrid process, and the support for different phases (liquid and solid). Although not shown here, there is also support for vapor phases in applicable unit operations, i.e. an evaporator unit for solvent swap or preconcentration that can be operated in batch or semi-batch mode. Among the physical properties of the solid phases, PharmaPy carries particle size distribution information, allowing for accurate modeling in the future drug product processing steps.

As seen from Eqs. (40)-(42), PharmaPy also allows for high-resolution FVM modeling in crystallization units. Given the future direction of the software, PharmaPy also allows a lower fidelity solution method, the standard method of moments, for crystallization units. This feature gives users the ability to weigh computational burden against model fidelity, especially when performing timewise expensive analysis techniques such as design space analysis and identification.

3.3. Flowsheet optimization

Flowsheet optimization is a key element in promoting and facilitating design of end-to-end optimal pharmaceutical manufacturing processes. As mentioned previously, any flowsheet optimization tool that can utilize function callbacks within Python can use a PharmaPy simulation to perform flowsheet optimization. Here, we will utilize two techniques to demonstrate how flowsheet optimization can be performed in concert with PharmaPy simulations using two different tools as frameworks. First, a grid-based approach is used, leveraging process feasibility resulting from an operating constraint to fix one of the operating variables and subsequently choosing an operating point from the reduced, grid-based design space. This entire process is automated as the framework may call a PharmaPy flowsheet at varying operating conditions in succession without the need for manually respecifying those conditions. Similarly, the second approach utilizes an open-source, gradient-free optimization package, Nevergrad (Rapin and Teytaud, 2018), which calls a PharmaPy simulation to determine an optimal operating point over a specified operating region.

Using the case study in Section 3.2, a grid-based optimization procedure is employed to maximize API production, m, while considering the ratio of necessary filtration time, t_{F01}, to batch crystallization time, t_{CRO1}, as a secondary objective. In this case a value of 0.01 was chosen as the weight, β, for the secondary objective. Three operating variables, $\{\tau_{R01}, C_{A,B,fed}, t_{CR01}\}$, are discretized by the following schemes: R01 residence time from 600s to 3600s with 11 evenly spaced points, feed concentration of A and B from 0.6 mol/L to 1.5 mol/L with 10 evenly spaced points, and batch crystallization time from 3 to 10 hours with 8 evenly spaced points, respectively.

$$\max_{\tau_{R01}, C_{A,B,feed}, t_{CR01}} \{m - \beta(t_{F01} / t_{CR01})\} \tag{52}$$

The volume of R01 is held constant at 2.4L for each simulation at which volumetric flowrate is adjusted to account for the residence time design specification. The simulation time specification is matched for the reactor and batch crystallization unit, for instance if the batch crystallization time is 3 hours, the upstream system is run for 3 hours to generate the material to be sent to the batch crystallization unit. To account for disparity in total mass output for longer simulation times, the objective function for API mass output is described as a rate (kg/h) and filtration time is measured as a fraction of batch time. Although the process constraints are primarily design specification variable bounds, concentration of API in R01 must not exceed the solubility limit at any time:

$$C(t, V) < \alpha C_{sat}(t, V), \qquad \forall t \in (0, t_{sim}), \; V \in (0, V_{R01}) \tag{53}$$

In this case, we choose $\alpha = 0.95$ for simplicity, however if uncertainty information on model parameters is available, for instance by the methods described in Section 3.1, α should be chosen to ensure Eq. (1.50) is not violated with acceptable probabilistic certainty.

The 880 design specification points were simulated in serial for a total computational time of 1180s, or just about 20 minutes, averaging roughly 1.34s per simulation instance. First, the solubility constraint was analyzed by the data shown in Fig. 17, where the maximum relative supersaturation is plotted for residence time versus feed concentration, averaged over simulation time. Any value above 0 represents a violation of Eq. (1.50), with points in the lower left region representing feasible operating conditions. An input concentration of 1.0 mol/L was chosen based on maximizing API output while adhering to the solubility limit within R01.

Next, the two simultaneous objectives were compared over the remaining two design variables (residence time for R01 and batch time). Fig. 18 shows the rate of mass production on the left, as well as total filtration time as a fraction of batch time for residence time versus total batch time. Generally, more total mass processed leads to higher filtration times and rate of mass production is relatively constant for fixed residence time values.

These trends are combined to generate a pareto-efficiency curve in Fig. 19. Thus, choosing a low residence time (600s) produces API at a high rate, and a medium batch time (6 hours) ensures filtration time as a fraction of batch time is minimized. However, the filtration time in all cases is not near the batch time (<25%) for all cases, indicating that batch time may be better chosen via another design criteria, for instance to accommodate the most convenient hybrid operation schedule.

A more direct optimization implementation could also employ any gradient-free optimization platform. To illustrate this capability, Nevergrad (Rapin and Teytaud, 2018), a gradient-free optimization package which internally invokes evolutionary-type search algorithms, was chosen. Nevergrad allows the user to choose from various implemented gradient-free optimization approaches to optimize input-output functions in Python without the use of derivative information. In this case, the so-called One Plus One algorithm implementation within Nevergrad was used to perform gradient-free optimization. For this strategy, one defines the objective function in Equation (1.49) along with a function of the associated simulation of the PharmaPy flowsheet for the API synthesis-purification process. Variable bounds are added as simple constraints and objective function penalties are tuned by the user on a case-to-case basis. Performing flowsheet optimization with a One Plus One algorithm in Nevergrad using a budget of 100 nodes and equivalent bounds to the previous approach renders an optimal point with residence time at 603 seconds, fed concentration of species A and B at 1.105 mol/L, and a batch crytallization time of 3.51 hours. These results align similarly with the parameter sweep approach, indicating that one can achieve improved mass production at higher concentrations so long as the reactor does not exceed the solubility limit. Although implementing a multi-objective penalty function for constraint violation can be successful in some cases, the cumbersome process of tuning objective penalties may outweigh the reduction in computational burden by limiting the number of flowsheet simulations. In this case, the flowsheet and objective are straightforward enough to realize improved solution quality with limited parameter tuning. When ample computational resources are available, performing both forms of optimization analysis could lead to deeper insight and compel little user effort beyond that required for the initial specification and simulation of the process within PharmaPy. Even so, the optimality of such points should still be verified on a case-to-case basis given that the methods shown here provide no mathematical guarantees of optimality. On a similar note, for practical applications, experimental validation of the accuracy and feasibility of the theoretically optimal operating conditions, should also be performed.

4. Discussion

The PharmaPy functionalities discussed in this paper represent the core use cases essential to support the design of hybrid processes, which spans parameter estimation, process simulation, and process analysis/optimization. The core PharmaPy feature are its object-oriented architecture coupled with efficient modular structure, which permits fast model solution and data transfer, as well as the construction of future models by the declaration of a set of consistent model abstractions (Phases, Unit Operations, etc). The latter is a key component that facilitates to expand the PharmaPy library, aiming to cover drug product UOs, which is intended for the analysis of end-to-end optimal pharmaceutical processes.

Important additional benefits of the architecture are that it can readily accommodate both generation of flowsheet superstructure and conversion to equation-based mode for efficient execution of superstructure optimization strategies. Detailed discussion of these elements are beyond the scope of this paper – here we only offer a sketch of these developments which will be described in follow-on publications.

Built into PharmaPy will be an optimization component that utilizes the Python-based optimization modeling tool, Pyomo. Flowsheets created for sequential-modular simulation within PharmaPy will have an option to be translated into a simultaneous equation-oriented Pyomo model. Given the well-known convergence challenges that go along with solving large-scale NLPs, simulation is a required first step in translating a flowsheet into one large mathematical model as simulation provides important model-specific integration timesteps as well as state variable profiles for initialization. The same automation process required to express a flowsheet as a mathematical model for flowsheet optimization is also required when generating flowsheet alternatives from a general design superstructure. By developing key elements of this automation process, flowsheets may be generated from a design superstructure and consequently be translated to Pyomo models where the user may utilize state-of-the-art solvers such as IPOPT (Wächter and Biegler, 2006), for local solution, or BARON (Sahinidis, 1996), for global solution, of rich, simultaneous equation-oriented flowsheet and process optimization problems.

5. Conclusions

Here, we present the structure and main capabilities of the Python package PharmaPy, aimed to support the simulation and analysis of hybrid pharmaceutical process. This is made possible by i) a collection of model representations for unit operation modeling and solution under the sequential-modular approach, ii) well defined communication interfaces for fast and reliable data transfer and object connection, and iii) a material transfer logic that covers different hybrid manufacturing scenarios involving batch, semi-batch and continuous unit operations. Moreover, the availability of solid, liquid and vapor modeling objects allow the construction of the current library of unit operation models, ranging from API synthesis all the way to separation, and sets the basis to easily and intuitively create new drug product or drug substance unit operation models.

The platform structure also allows the use of PharmaPy for a variety of purposes, namely, parameter estimation, process simulation, and process analysis. Any process analysis technique requiring simulation as an intermediate step may exploit the utility of PharmaPy. We demonstrated that utility by enumerating many combinations of process operating conditions and direct, derivative-free optimization relying on repeated process simulation. Strategies such as traditional design of experiments, with repetitive experimental and parameter estimation steps, may be automated within PharmaPy and are currently being explored for future work.

PharmaPy is built by making extensive use of object-oriented programming concepts, and benefits from the flexibility, software development, and scripting capabilities of the Python programming language. Moreover, this also enables the use of a rich set of open-source tools that are illustrated in this contribution to expand the capabilities of PharmaPy. For example, by using techniques based solely on process simulation, or more elaborate techniques involving model decomposition and rigorous optimization techniques such as those facilitated by Pyomo. We are currently exploring analysis techniques using simulation and rigorous optimization to improve the utility and native functionality of PharmaPy.

The convergence of these features makes PharmaPy a powerful tool for the analysis of numerous processing schemes encountered in pharmaceutical manufacturing systems. Also, the package serves as a valuable tool for supporting the current paradigms for pharmaceutical process improvement such as Quality by Design, which heavily relies on the construction and maintenance of robust digital process models.

Acknowledgments

This project was supported by the United States Food and Drug Administration through grant U01FD006738. The views expressed by the authors do not necessarily reflect the official policies of the Department of Health and Human Services; nor does any mention of trade names, commercial practices, or organization imply endorsement by the United States Government.

References

Abrams DS, Prausnitz JM, 1975. Statistical thermodynamics of liquid mixtures: A new expression for the excess Gibbs energy of partly or completely miscible systems. AIChE J 21, 116–128. doi:10.1002/aic.690210115.

Andersson C, Führer C, Åkesson J, 2015. Assimulo: A unified framework for ODE solvers. Math. Comput. Simul 116, 26–43. doi:10.1016/j.matcom.2015.04.007.

Aspen Technology Inc., 2020. Aspen Plus [WWW Document]. Aspen Eng. Suite URL https://www.aspentech.com/en (accessed 12.24.20).

Bano G, Facco P, Ierapetritou M, Bezzo F, Barolo M, 2019. Design space maintenance by online model adaptation in pharmaceutical manufacturing. Comput. Chem. Eng 127, 254–271. doi:10.1016/j.compchemeng.2019.05.019.

Bard Y, 1974. Nonlinear Parameter Estimation, Nonlinear Parameter Estimation. Academic Press, London.

Barton PI, 1992. The modelling and simulation of combined discrete/continuous processes. Imperial College of Science. Tehnology and Medicine.

Bates D, Watts DG, 1988. Nonlinear Regression Analysis and Its Applications. John Wiley & Sons, New York.

Biegler LT, 2010. Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes. Siam, Philadelphia doi:10.1137/1.9780898719383.

Bilardello P, Joulia X, Le Lann JM, Delmas H, Koehret B, 1993. A general strategy for parameter estimation in differential—algebraic systems. Comput. Chem. Eng 17, 517–525. doi:10.1016/0098-1354(93)80040-T.

Bogusch R, Marquardt W, 1997. A formal representation of process model equations. Comput. Chem. Eng 21, 1105–1115. doi:10.1016/S0098-1354(96)00321-3.

Boukouvala F, Muzzio FJ, Ierapetritou MG, 2010. Design space of pharmaceutical processes using data-driven-based methods. J. Pharm. Innov 5, 119–137. doi:10.1007/s12247-010-9086-y.

Boukouvala F, Niotis V, Ramachandran R, Muzzio FJ, Ierapetritou MG, 2012. An integrated approach for dynamic flowsheet modeling and sensitivity analysis of a continuous tablet manufacturing process. Comput. Chem. Eng 42, 30–47. doi:10.1016/j.compchemeng.2012.02.015.

Bourcier D, Féraud JP, Colson D, Mandrick K, Ode D, Brackx E, Puel F, 2016. Influence of particle size and shape properties on cake resistance and compressibility during pressure filtration. Chem. Eng. Sci 144, 176–187. doi:10.1016/j.ces.2016.01.023.

Braunschweig B, Gani R (Eds.), 2002. Software Architectures and Tools for Computer-Aided Process Engineering. Computer-Aided Chemical Engineering. Elsevier Science B. V., Amsterdam.

Casas-Orozco D, Villa AL, Guerra OJ, Reklaitis GV, 2018. Dynamic parameter estimation and identifiability analysis for heterogeneously-catalyzed reactions: Catalytic synthesis of nopol. Chem. Eng. Res. Des 134, 226–237. doi:10.1016/j.cherd.2018.04.002.

Dalcín L, Paz R, Storti M, 2005. MPI for Python. J. Parallel Distrib. Comput 65, 1108–1115. doi:10.1016/j.jpdc.2005.03.010.

Dosta M, 2019. Modular-Based Simulation of Single Process Units. Chem. Eng. Technol 42, 699–707. doi:10.1002/ceat.201800671.

Esposito JM, Kumar V, 2007. A state event detection algorithm for numerically simulating hybrid systems with model singularities. ACM Trans. Model. Comput. Simul 17. doi:10.1145/1189756.1189757.

García-Muñoz S, Dolph S, Ward HW, 2010. Handling uncertainty in the establishment of a design space for the manufacture of a pharmaceutical product. Comput. Chem. Eng 34, 1098–1107. doi:10.1016/j.compchemeng.2010.02.027.

Gmehling J, Li J, Schiller M, 1993. A modified UNIFAC model. 2. Present parameter matrix and results for different thermodynamic properties. Ind. Eng. Chem. Res 32, 178–193. doi:10.1021/ie00013a024.

Gunawan R, Fusman I, Braatz RD, 2004. High resolution algorithms for multidimensional population balance equations. AIChE J 50, 2738–2749. doi:10.1002/aic.10228.

Hairer E, Wanner G, 1991. Solving Ordinary Differential Equations: Stiff and Differential-Algebraic Problems, 2nd ed. Springer, Leipzig.

Hillestad M, Hertzberg T, 1986. Dynamic simulation of chemical engineering systems by the sequential modular approach. Comput. Chem. Eng 10, 377–388. doi:10.1016/0098-1354(86)87008-9.

Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS, 2005. Sundials. ACM Trans. Math. Softw 31, 363–396. doi:10.1145/1089014.1089020.

Içten E, Maloney AJ, Beaver MG, Zhu X, Shen DE, Robinson JA, Parsons AT, Allian A, Huggins S, Hart R, Rolandi P, Walker SD, Braatz RD, 2020. A Virtual Plant for Integrated Continuous Manufacturing of a Carfilzomib Drug Substance Intermediate, Part 2: Enone Synthesis via a Barbier-Type Grignard Process. Org. Process Res. Dev doi:10.1021/acs.oprd.0c00188.

Kulikov V, Briesen H, Grosch R, Yang A, Von Wedel L, Marquardt W, 2005. Modular dynamic simulation for integrated particulate processes by means of tool integration. Chem. Eng. Sci 60, 2069–2083. doi:10.1016/j.ces.2004.11.037.

Laky D, Xu S, Rodriguez JS, Vaidyaraman S, Muñoz SG, Laird C, 2019. An optimization-based framework to define the probabilistic design space of pharmaceutical processes with model uncertainty.. Processes 7. doi:10.3390/pr7020096.

LeVeque RJ, 2002. Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, New York.

Maclaurin D, Duvenaud D, Johnson M, Townsend J, 2020. Autograd.

Marquardt DW, 1963. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. J. Soc. Ind. Appl. Math 11, 431–441. doi:10.1137/0111030.

Marquardt W, 1992. An object-oriented representation of structured process models. Comput. Chem. Eng 16, S329–S336. doi:10.1016/S0098-1354(09)80039-2.

Nagy ZK, Braatz RD, 2007. Distributional uncertainty analysis using power series and polynomial chaos expansions. J. Process Control 17, 229–240. doi:10.1016/j.jprocont.2006.10.008.

Nagy ZK, El Hagrasy A, Lister J, 2020. Continuous Pharmaceutical Processing. Springer, Gewerbestrasse.

Nielsen HB, 1999. Damping parameter in Marquardt's method.. Tech. Rep 16.

Ochoa MP, Deshpande A, García-Muñoz S, Stamatis S, Grossmann IE, 2019. Flexibility Analysis For Design Space Definition. Comput. Aided Chem. Eng doi:10.1016/B978-0-12-818597-1.50051-5.

Oh M, Moon I, 1998. Framework of dynamic simulation for complex chemical processes. Korean J. Chem. Eng 15, 231–242. doi:10.1007/BF02707077.

Ouchlyama N, Tanaka T, 1986. Porosity estimation from particle size distribution. Ind. Eng. Chem. Fundam25, 125–129. doi:10.1021/i100021a019.

Pantelides CC, Barton PI, 1993. Equation-oriented dynamic simulation current status and future perspectives. Comput. Chem. Eng 17, S263–S285. doi:10.1016/0098-1354(93)80240-N.

Patrascu M, Barton PI, 2018. Optimal campaigns in end-to-end continuous pharmaceuticals manufacturing. Part 2: Dynamic optimization. Chem. Eng. Process. - Process Intensif 125, 124–132. doi:10.1016/j.cep.2018.01.015.

Press WH, Teukolsky SA, Vetterling WT, Flannery BP, 2001. Numerical Recipes in Fortran 77, 2nd ed. Press Syndicate of the University of Cambridge, Cambridge.

Process Systems Enterprise, 2020. Process Systems Enterprise, gPROMS [WWW Document]. URL www.psenterprise.com/products/gproms (accessed 12.24.20).

Puigjaner L, Reklaitis GV, Graells M, 2002. Computer tools for Discrete/Hybrid Production Systems. In: Braunschweig B, Gani R (Eds.), Software Architectures and Tools for Computer-Aided Process Engineering. New York, p. 661.

Pulsipher JL, Zavala VM, 2019. A scalable stochastic programming approach for the design of flexible systems. Comput. Chem. Eng 128, 69–76. doi:10.1016/j.compchemeng.2019.05.033.

Randolph AD, Larson MA, 1988. Theory of Particulate Processes: Analysis and Techniques of Continuous Crystallization, 2nd ed. Academic Press, San Diego, CA.

Rapin J, Teytaud O, 2018. Nevergrad - A gradient-free optimization platform. GitHub Repos.

Rawlings J, Ekerdt J, 2002. Chemical reactor analysis and design fundamentals. Nob Hill Publishing, Madison, WI.

Rawlings JB, Miller SM, Witkowski WR, 1993. Model Identification and Control of Solution Crystallization Processes: A Review. Ind. Eng. Chem. Res 32, 1275–1296. doi:10.1021/ie00019a002.

Saad Y, Schultz MH, 1986. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM J. Sci. Stat. Comput 7, 856–869. doi:10.1137/0907058.

Sahinidis NV, 1996. BARON: A general purpose global optimization software package. J. Glob. Optim 8, 201–205. doi:10.1007/bf00138693.

Skorych V, Dosta M, Hartge EU, Heinrich S, 2017. Novel system for dynamic flowsheet simulation of solids processes. Powder Technol 314, 665–679. doi:10.1016/j.powtec.2017.01.061.

Sridharan S, Balakrishanan R, 2019. Discrete Mathematics: Graph Algorithms, Algebraic Structures, Coding Theory, and Cryptography. CRC Press, Boca Ratón, FL.

Su Q, Ganesh S, Moreno M, Bommireddy Y, Gonzalez M, Reklaitis GV, Nagy ZK, 2019. A perspective on Quality-by-Control (QbC) in pharmaceutical continuous manufacturing. Comput. Chem. Eng 125, 216–231. doi:10.1016/j.compchemeng.2019.03.001. [PubMed: 36845965]

Szilagyi B, Majumder A, Nagy ZK, 2020. Fundamentals of Population Balance Based Crystallization Process Modeling, in: The Handbook of Continuous Crystallization. Royal Society of Chemistry, Cambridge, pp. 51–101. doi:10.1039/9781788013581-00051.

Tu H, Rinard IH, 2006. ForeSee—A hierarchical dynamic modeling and simulation system of complex processes. Comput. Chem. Eng 30, 1324–1345. doi:10.1016/j.compchemeng.2005.12.007.

Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat I, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Methods 17, 261–272. doi:10.1038/s41592-019-0686-2. [PubMed: 32015543]

Wächter A, Biegler LT, 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program 106, 25–57. doi:10.1007/s10107-004-0559-y.

Wang Z, Ierapetritou M, 2017. A Novel Surrogate-Based Optimization Method for Black-Box Simulation with Heteroscedastic Noise. Ind. Eng. Chem. Res 56, 10720–10732. doi:10.1021/acs.iecr.7b00867.

Watts DG, 1994. Estimating parameters in nonlinear rate equations. Can. J. Chem. Eng 72, 701–710. doi:10.1002/cjce.5450720420.

Yu LX, Amidon G, Khan MA, Hoag SW, Polli J, Raju GK, Woodcock J, 2014. Understanding pharmaceutical quality by design. AAPS J 16, 771–783. doi:10.1208/s12248-014-9598-3. [PubMed: 24854893]

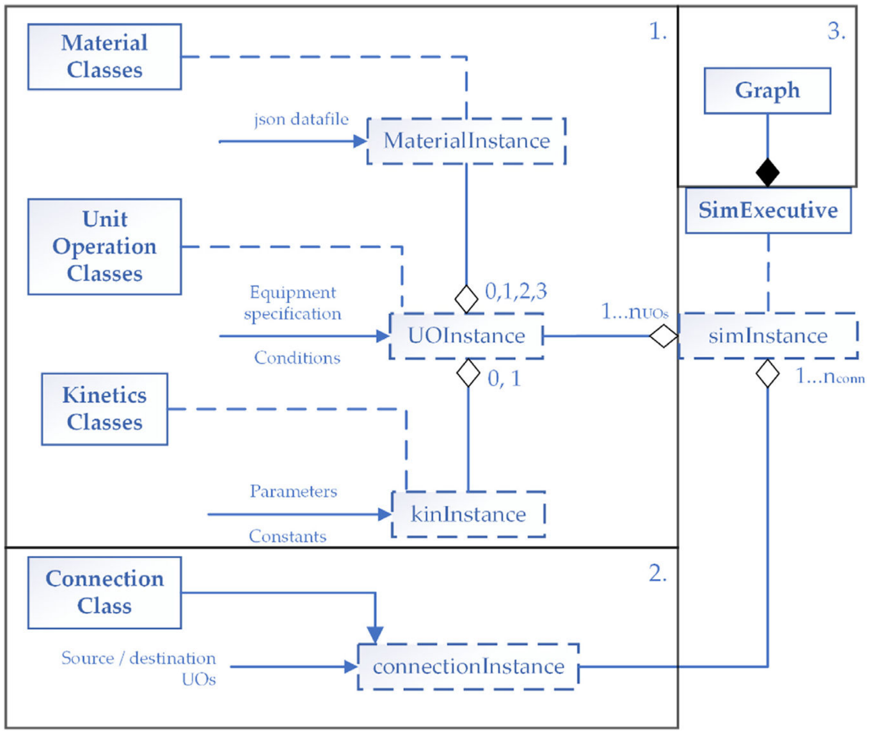

Fig. 1.

Object diagram of the architecture of PharmaPy. Hollow diamonds: aggregation, filled diamonds: composition, dashed lines: instantiation, arrows: entering arguments.

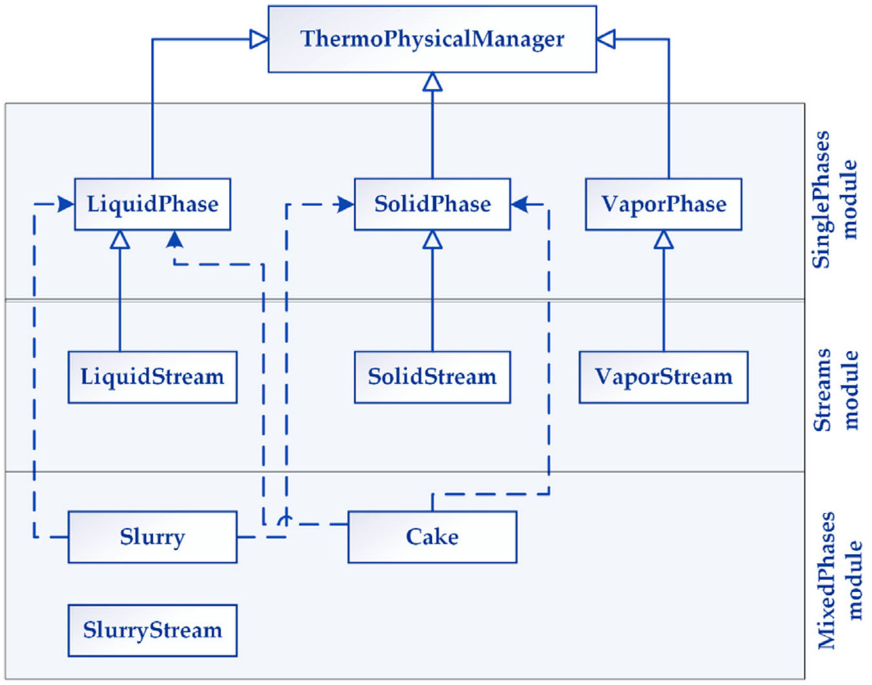

Fig. 2.
Simplified class diagram for the material classes of PharmaPy. Hollow arrows: inheritance, filled arrows: aggregation.

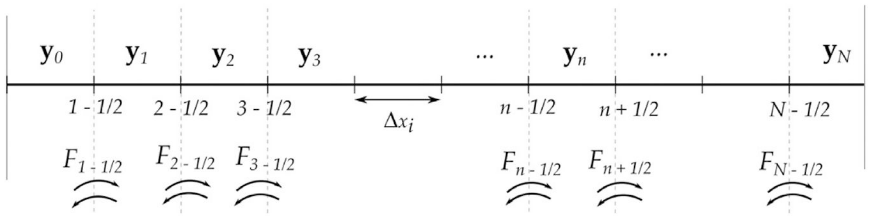

Fig. 3.
Scheme of the discretization model for the HR discretization method for a spatial coordinate x_i.

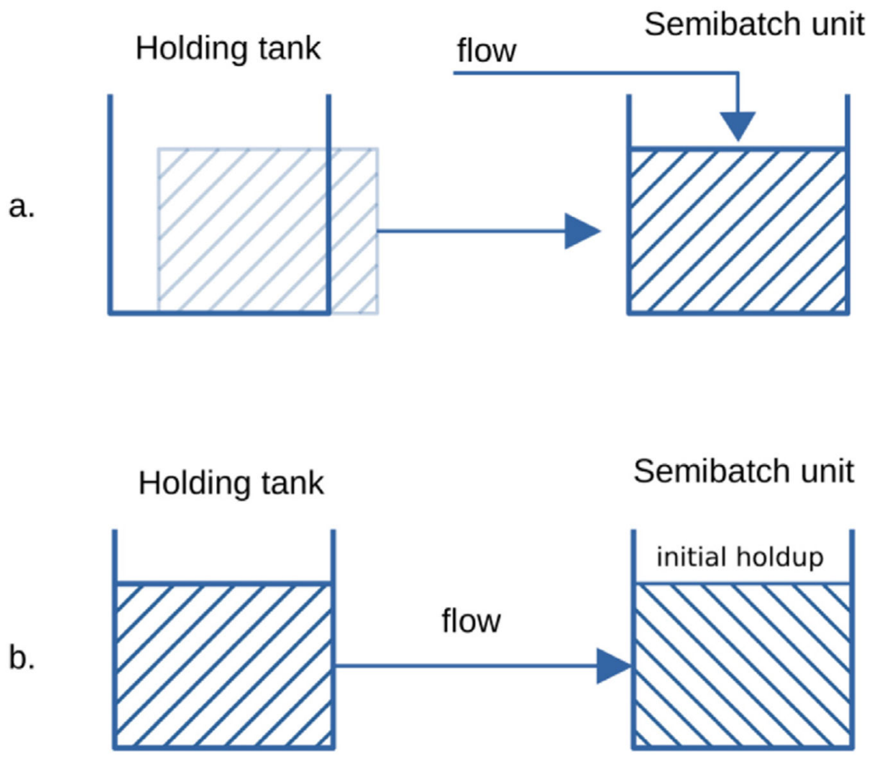

Fig. 4.
Material transfer policies for semi-batch reactors.

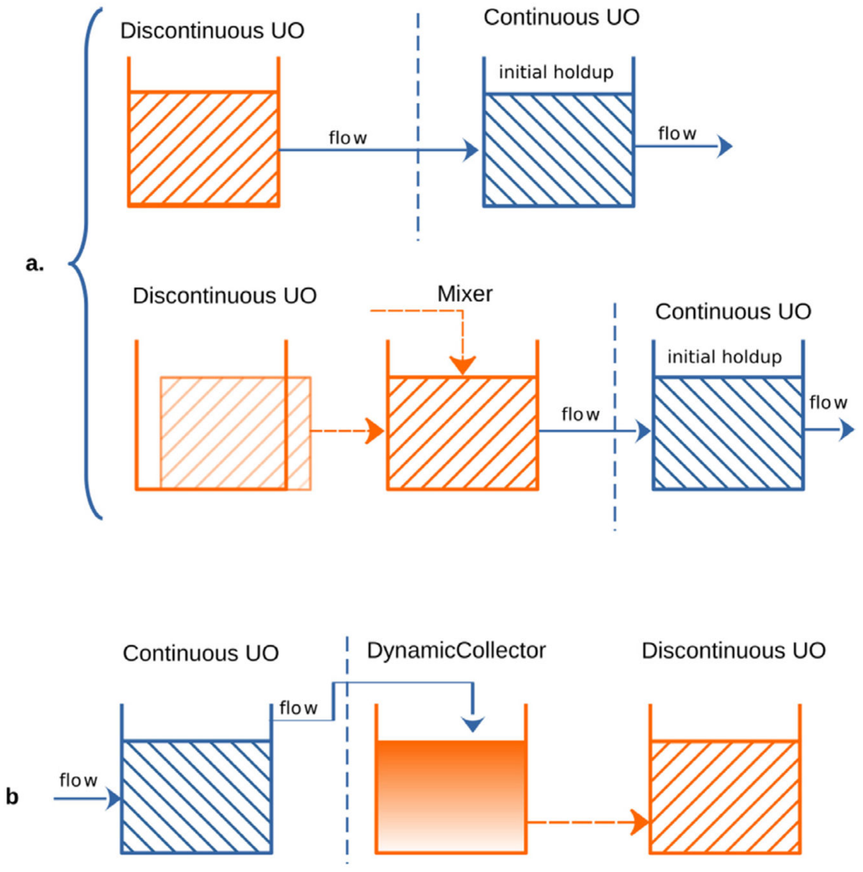

Fig. 5.

Transfer policies for hybrid flowsheets. Continuous arrows: material flow. Dashed arrows: material transfer after the completion of the discontinuous operation, or material added at one time. Dashed vertical lines separate continuous from discontinuous operation

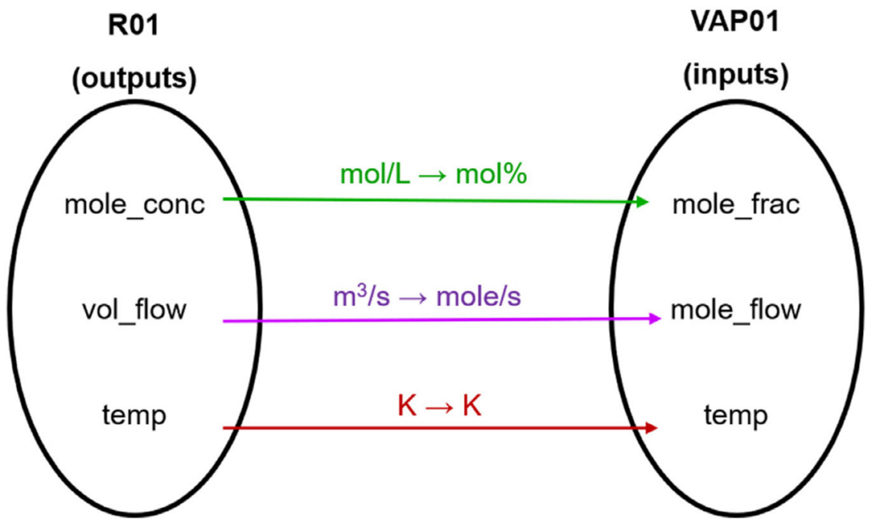

Fig. 6.

Bipartite graph between the set of names of two connected unit operations: a continuous reactor R01 and a continuous vaporizer VAP01.

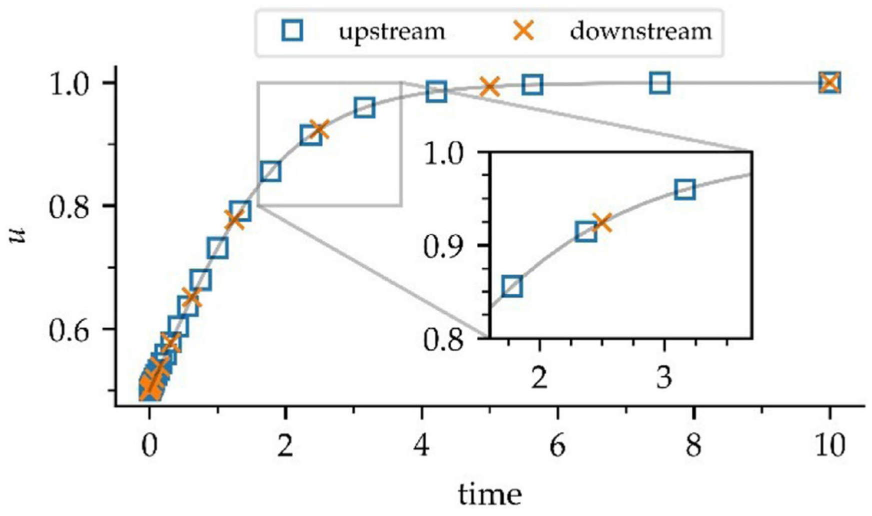

Fig. 7.

Time grids for two connected unit operations. For local evaluation, a Newton interpolation formula of order 3 is employed to evaluate the inputs to the downstream UO from upstream UO data.

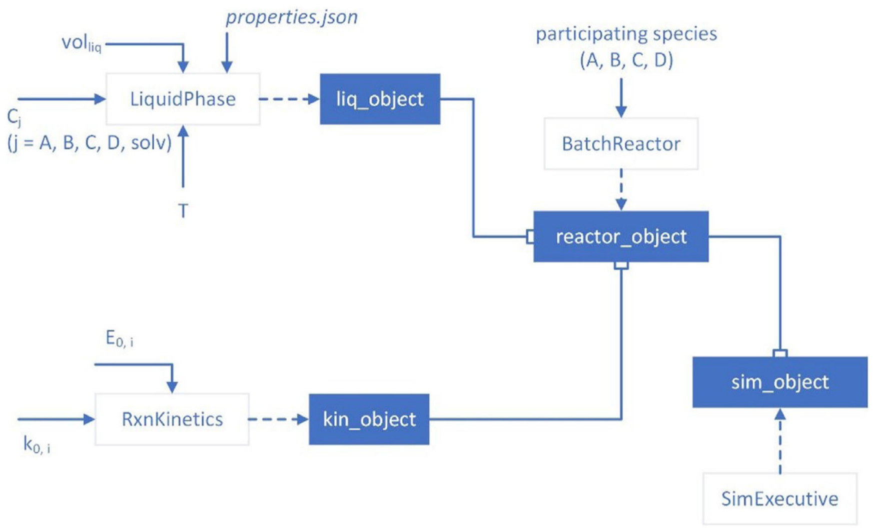

Fig. 8.

PharmaPy object creation for a parameter estimation problem. Continuous arrows: user-provided arguments for object instantiation, dashed arrows: object instantiation, lines with squared hollow tips: object aggregation.

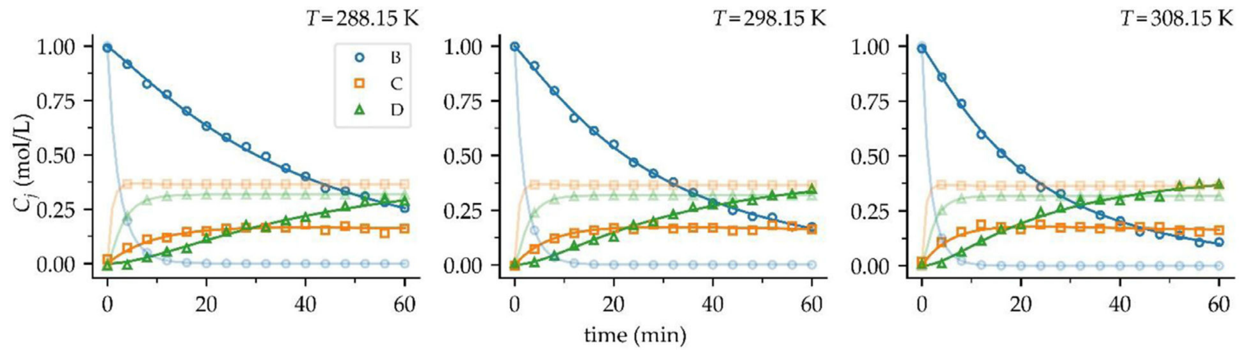

Fig. 9.

Data fitting for the reaction datasets. Markers: experimental data, continuous lines: model predictions, shaded line with markers: model predictions using initial estimate parameters.

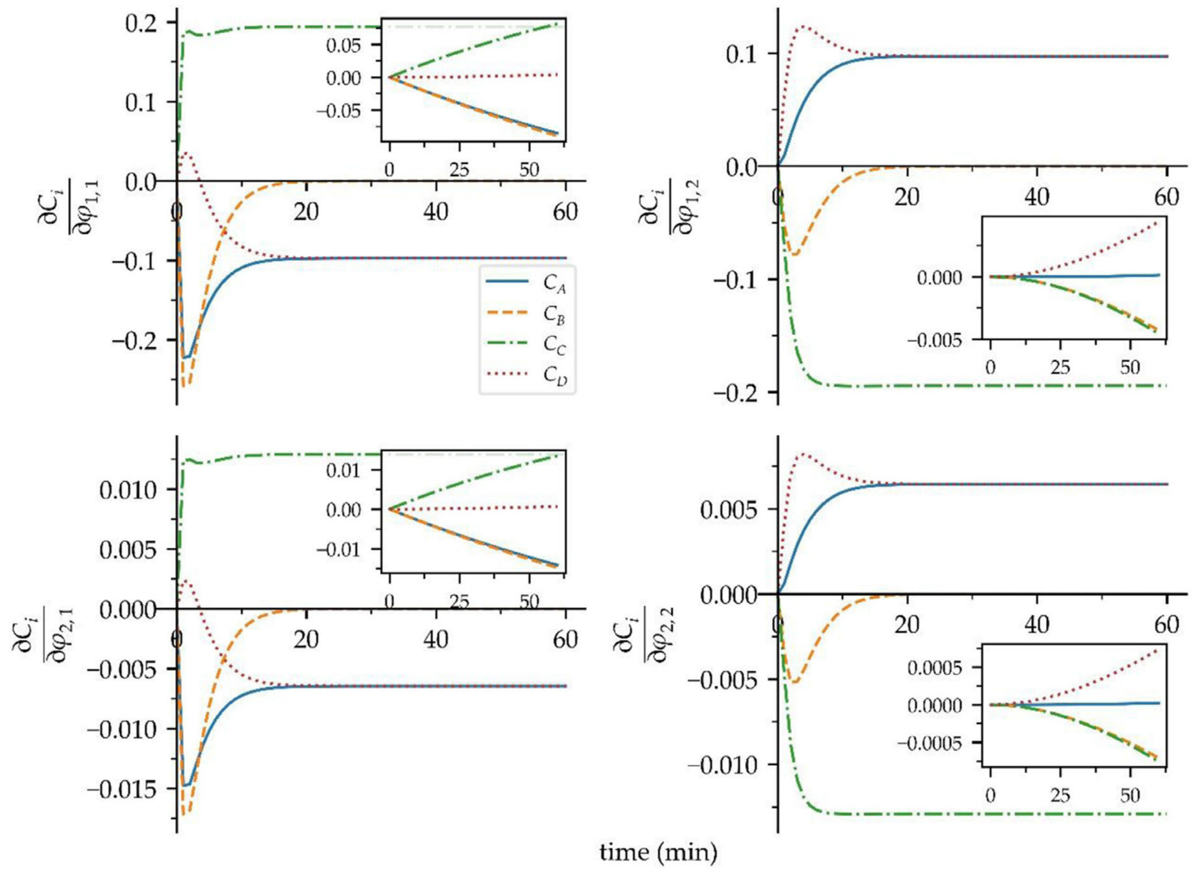

Fig. 10.

Parametric sensitivity for $\mathbf{E}_{\mathbf{a},seed} = [10^4, 10^4]$ (main plots) and $\mathbf{E}_{\mathbf{a},seed} = [2.5 \times 10^4, 2.5 \times 10^4]$(inset plots) at 308 K.

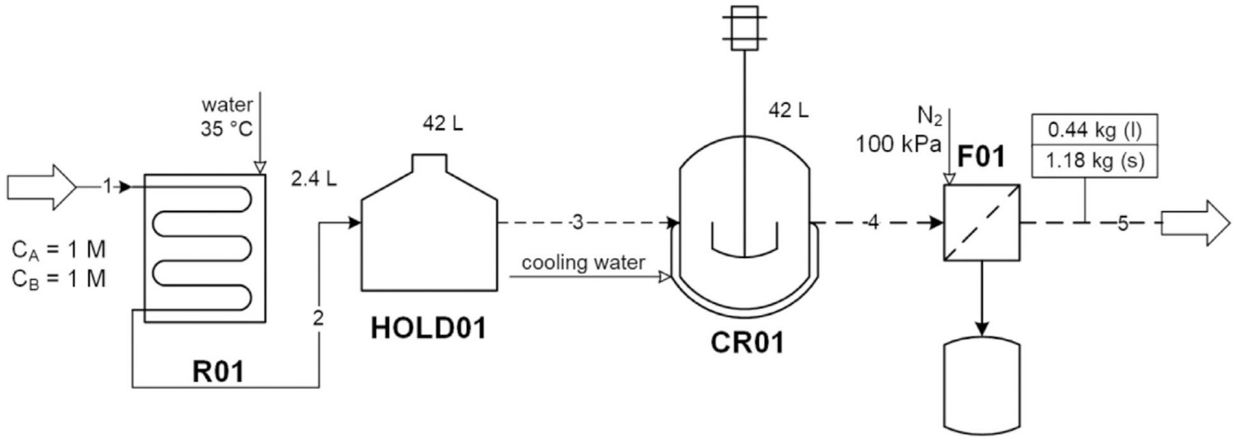

Fig. 11.

Process flow diagram for the production of the API C. Continuous lines: continuous flows. Dashed lines: material transfer after the completion of the batch.

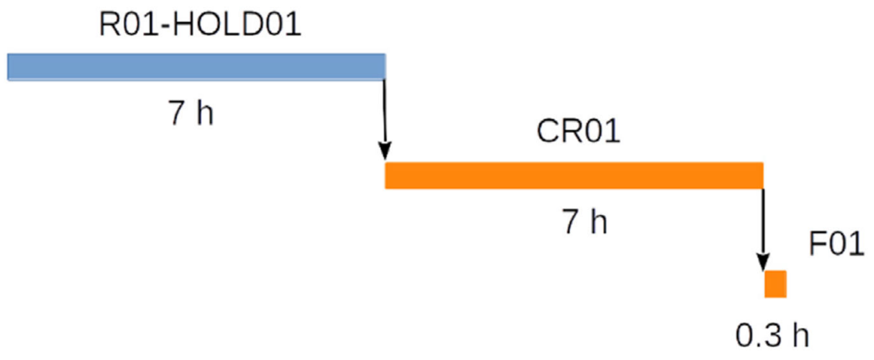

Fig. 12.
Gantt chart of the simulated case study. R01-HOLD01: continuous UOs. CR01 and F01: batch UOs.

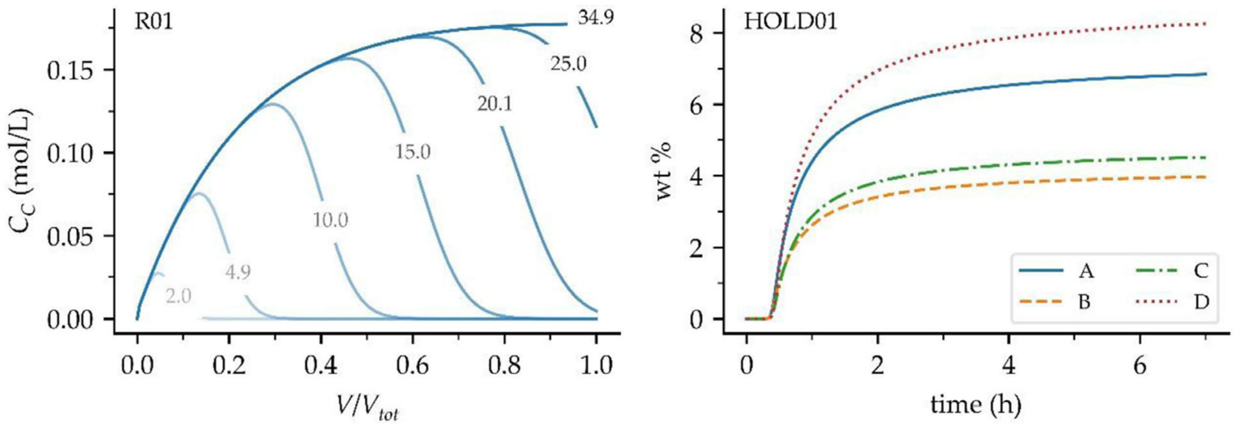

Fig. 13.

Left: transient API concentration as a function reactor position (time tags in minutes). b. Mass fraction of the accumulated reactor outlet over time.

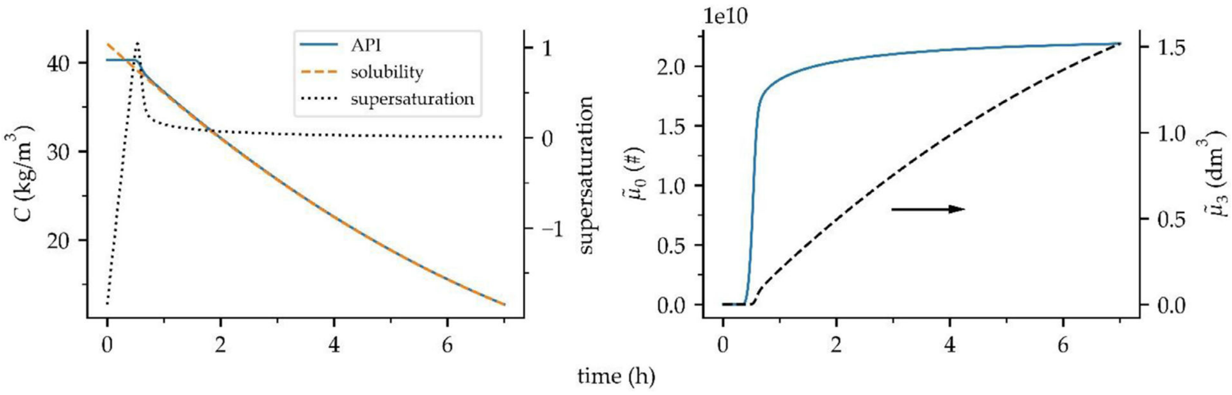

Fig. 14.

Results for batch crystallizer CR01. Left: API concentration profile along with the solubility profile over time. Right: zeroth and third moment of the CSD over time.

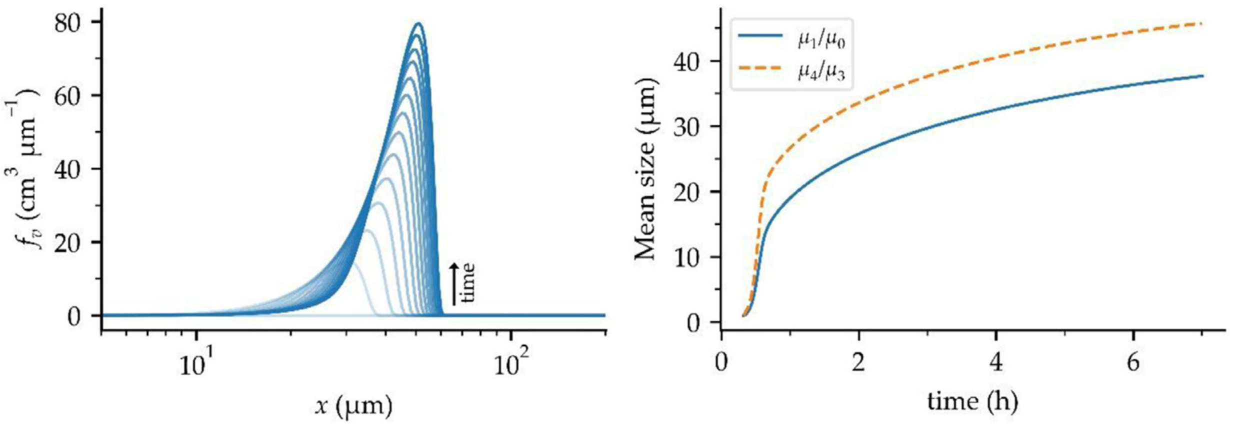

Fig. 15.

Left: Time evolution of the volume-based CSD for the simulated case study (distribution sampled every 0.5 h). Right: evolution of mean crystal size.

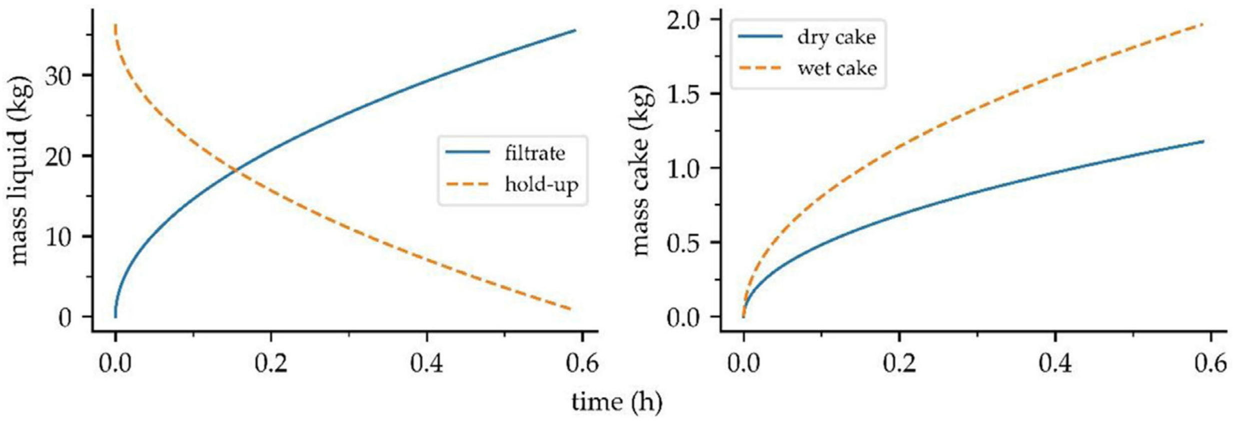

Fig. 16.
Filtrate (left) and cake (right) mass in F01.

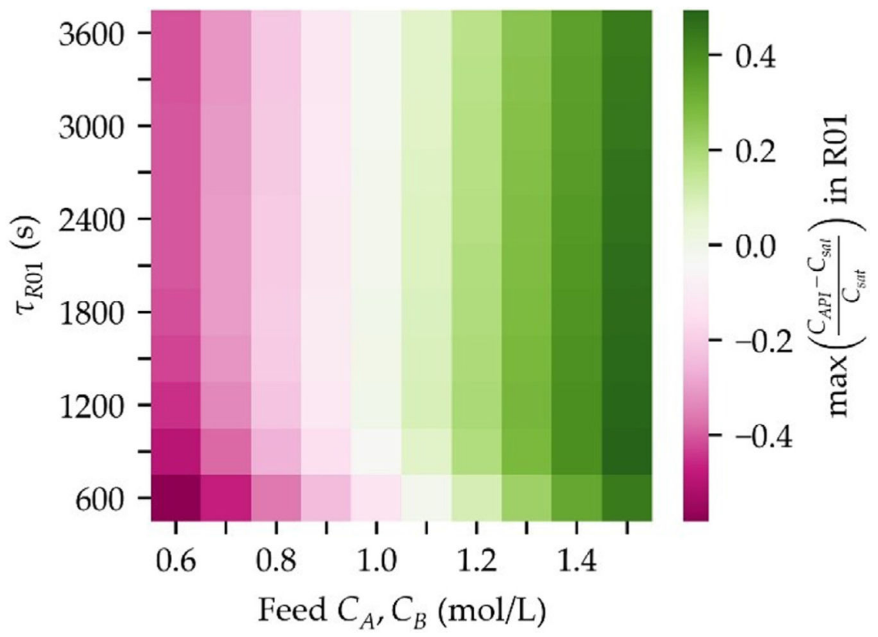

Fig. 17.

Effect of reactor residence time and reactant input concentrations in relative supersaturation.

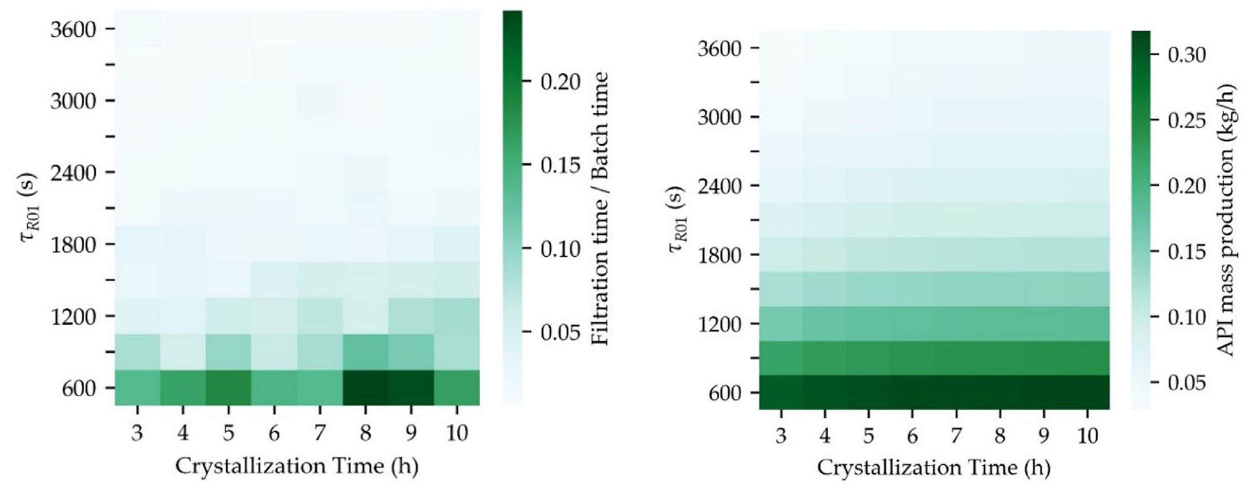

Fig. 18.

Left: Effect of reaction and crystallization times on filtration time. Right: Effect of reaction and crystallization times on total API production.

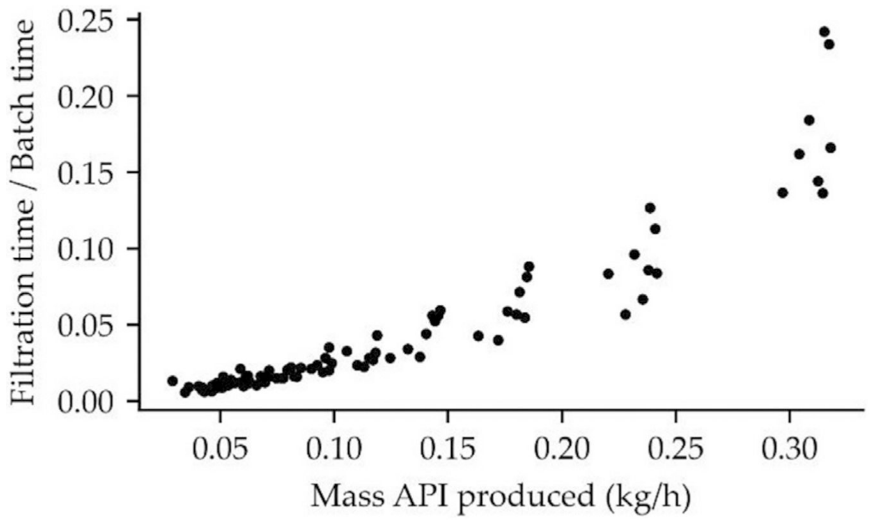

Fig. 19.

Pareto efficiency curve for filtration time as a fraction of total batch crystallization time versus mass production rate of API (kg/h) for unique simulated design specifications.

Table 1

Kinetic parameters used to generate reaction data and seeds used during parameter estimation.

Reaction (i)	$k_{0,i}(\mathrm{L\ mol}^{-1})$		$E_{a,i}(\mathrm{J\ mol}^{-1})$	
	Nominal	**Initial estimate**	**Nominal**	**Initial estimate**
1	3	0.5	22 000	10 000
2	2.1	0.5	18 000	10 000

Table 2

95% confidence intervals for the regressed parameters.

Parameter	**Lower bound**	**Converged value**	**Upper bound**
k_1 (L mol^{-1})	2.404	2.938	3.597
k_2 (L mol^{-1})	1.659	2.397	3.485
$E_{a,1}$ (J mol^{-1})	21 460.06	21 952.84	22 456.94
$E_{a,2}$ (J mol^{-1})	17 421.14	18 320.18	19 265.62

Table 3

Crystallization parameters for nucleation, growth and dissolution.

Kinetic Mechanism	Parameter Names	Nominal Values	Units
Primary Nucleation	$\{k_p, E_p, p\}$	{3e8, 0, 3}	# m^{-3} s^{-1}, J mol^{-1}, n/a
Secondary Nucleation	$\{k_s, E_s, s_1, s_2\}$	{4.46e10, 0, 2, 0}	# m^{-3} s^{-1}, J mol^{-1}, n/a, n/a
Growth	$\{k_g, E_g, g\}$	{5, 0, 1.32}	m s^{-1}, J mol^{-1}, n/a
Dissolution	$\{k_d, E_d, d\}$	{1, 0, 1}	m s^{-1}, J mol^{-1}, n/a

Table 4

Stream table for the final state of the analyzed flowsheet (Fig. 11).

Stream	**Material object**	$T(\mathrm{K})$	$m(\mathrm{kg})$	$V(\mathrm{L})$	$\dot{m}(\mathrm{kg\,s^{-1}})$	$\dot{V}(\mathrm{L\,min^{-1}})$	w_j / (wt %)				
							A	**B**	**C**	**D**	**solvent**
1	LiquidStream	308.15			0.0015	0.1000	12.36	11.61	0	0	76.02
2	LiquidStream	309.2			0.0015	0.1000	7.25	4.20	4.79	8.78	74.99
3	LiquidPhase	309.0	37.46	42.0			6.84	3.97	4.52	8.25	76.42
4	LiquidPhase	279.0	36.32	40.7			7.06	4.10	1.43	8.52	78.89
	SolidPhase	279.0	1.18	0.0			0	0	100	0	0
5	LiquidPhase	279.0	0.44	0.5			7.06	4.10	1.43	8.52	78.89
	SolidPhase	279.0	1.18	0.0			0	0	100	0	0