Front. Chem. Sci. Eng. 2022, 16(2): 183–197
https://doi.org/10.1007/s11705-021-2073-7

RESEARCH ARTICLE

Using machine learning models to explore the solution space of large nonlinear systems underlying flowsheet simulations with constraints

Patrick Otto Ludl (✉)[1], Raoul Heese[1], Johannes Höller[1], Norbert Asprion[2], Michael Bortz[1]

1 Fraunhofer ITWM Optimization Department, Kaiserslautern 67663, Germany
2 Chemical and Process Engineering BASF SE, Ludwigshafen 67056, Germany

Abstract Flowsheet simulations of chemical processes on an industrial scale require the solution of large systems of nonlinear equations, so that solvability becomes a practical issue. Additional constraints from technical, economic, environmental, and safety considerations may further limit the feasible solution space beyond the convergence requirement. A priori, the design variable domains for which a simulation converges and fulfills the imposed constraints are usually unknown and it can become very time-consuming to distinguish feasible from infeasible design variable choices by simply running the simulation for each choice. To support the exploration of the design variable space for such scenarios, an adaptive sampling technique based on machine learning models has recently been proposed. However, that approach only considers the exploration of the convergent domain and ignores additional constraints. In this paper, we present an improvement which particularly takes the fulfillment of constraints into account. We successfully apply the proposed algorithm to a toy example in up to 20 dimensions and to an industrially relevant flowsheet simulation.

Keywords machine learning, flowsheet simulations, constraints, exploration

1 Introduction

Flowsheet simulation software is an important tool for designing new and optimizing existing chemical processes on an industrial scale. Solving these simulations reliably and with high computational efficiency is indispensable for practitioners. However, solution algorithms for these simulations often suffer from convergence issues, which eventually means that optimization potentials remain hidden. In this article, a method to guide and support the user in setting up convergent simulations is described.

Stationary flowsheet simulations as considered in this article are based on energy and mass balances as well as on physical property models like for example thermodynamic models for the phase equilibria or models for transport properties. A prototypical example is the equilibrium stage model. In this model, balance equations for mass and enthalpy are combined with phase equilibria, leading to a large, underdetermined system of nonlinear equations (called MESH equations), which has to be solved for the complete vector of process variables. To make this system uniquely solvable, a number of additional independent equations has to be included. Each of these additional equations corresponds to fixing a certain design variable of the simulation to a specified value. These design variables (also called specifications) are a subset of the process variables and their choice is responsible for the performance of the simulated process. In other words, a process engineer tunes the design variables until the resulting process behaves in a desired way.

An automatic tuning using an optimization algorithm to maximize or minimize an objective is known as process optimization and might include bounds on the design variables and additional constraints — like safety limits or load ranges of apparatuses — as inequality restrictions. These constraints are usually imposed by a process engineer based on expert knowledge. During the automatized optimization procedure, the flowsheet simulation has to be solved multiple times for different values of design variables until a stopping criterion is met. However, any flowsheet simulation may fail due to convergence

Received February 14, 2021; accepted May 20, 2021

E-mail: patrick.otto.ludl@itwm.fraunhofer.de

problems of the solver, which might be unable to find a solution (i.e., a complete vector of process variables) for the given system of equations. Also, only an infeasible solution might be found which violates the additionally imposed constraints. Therefore, the crucial question is: which design variable choices are allowed such that the complete problem can be solved in the sense that all equations and all constraints are fulfilled?

In this manuscript, we propose a machine learning (ML)-based algorithm to help answering this question by exploring the convergent and feasible solution domain. Once this domain is known, it can be estimated whether a particular, user-defined choice of design variables will lead to a feasible solution without running the actual simulation. To obtain the corresponding vector of process variables, the simulation can subsequently be run with a suitable initialization based on the gathered data. Moreover, after the algorithm is completed, an efficient process optimization can be performed based on the explored feasibility domain such that infeasible design variable domains are excluded.

The ML models we use operate on the space of design variables as inputs and a selected subset of process variables as outputs. That is, the flowsheet simulation is treated as a black-box, which is fed with values for the design variables, and returns (in the case of convergence) values for the outputs (including the inequality constraints). The ML models are therefore also called surrogate models.

Searching for the feasible domain of a black-box simulation is a well-known problem [1,2] with many applications. For the feasibility exploration with surrogates, Kriging models [3,4] or radial basis functions [5] are typically used in combination with an expected improvement function [6] in analogy to Bayes optimization [7]. A recent summary of different approaches can be found in ref. [8]. The aforementioned reference emphasizes on pharmaceutical processes, but most of the reviewed literature is also applicable to a broader field.

There are various previous works which are closely related to our method but focus on optimization rather than generic exploration. For example, Gramacy and Lee propose a method for optimization under unknown binary feasibility constraints using a Bayes-like scheme, where points are placed iteratively according to an expected improvement maximization [9]. In that work, points violating the binary feasibility constraint are still valid for building the surrogate model for the objective function as is the case for the continuous constraints in our paper. Similarly, Tran et al. considers a Bayesian optimization approach under both known and unknown constraints, including those that arise indirectly from divergent simulation runs [10]. The focus of that approach lies on parallelized simulation evaluations (i.e., high performance computing). In ref. [11], a method for Bayesian optimization of problems with continuous constraints is proposed, but without explicitly taking an additional binary feasibility into account. On the other hand, Griffiths and Hernández-Lobato study a constrained Bayesian optimization approach over the latent space of a variational autoencoder for automatic chemical design under a binary constraint without considering additional continuous constraints [12]. Finally, in ref. [13] the feasible region of a scheduling process is approximated with a ML model, which is subsequently transformed into a set of continuous constraints for a planning problem. In this sense, the constraint is not explicitly infused but instead extracted from the problem and further utilized to solve the combined scheduling and planning problem.

The approach we present in this manuscript is mainly based on ref. [14], in which a feasibility exploration method is presented for the case where instead of a continuous degree of feasibility only a discrete binary feasibility information (convergent/divergent) can be obtained from the simulator. Such a binary feasibility information distinguishes our problem from well-known approaches like, e.g., in refs. [1,2], where a continuous degree of feasibility violation is assumed to be known. In general, Bayesian optimization under constraints is also a well-known problem, which has been extensively studied before [15,16]. In this paper, which is inspired by these previous works, we improve and extend the approach from ref. [14] to also include continuous inequality constraints in addition to the binary feasibility. Moreover, we do not consider feasibility exploration with respect to a specific optimization goal but instead focus on a general (i.e., optimization-independent) feasibility exploration.

As outlined in Fig. 1, the main idea of the original method [14] is that an iterative design of computer experiments (i.e., flowsheet simulation runs) is performed to gather data, which are then used to iteratively improve ML models for the prediction of the convergence behavior of the simulation given the design variables. Such or similar loops are common for feasibility exploration strategies and are closely related to Bayes optimization methods [7].

The method we propose in this manuscript extends that original approach by requiring the fulfillment of additional continuous inequality constraints imposed on certain simulated quantities. Such constraints could be, for example, minimum quality requirements, maximum power consumption limits, total cost limits or a complicated set of restrictions for safety regulations. Imposing additional constraints divides the domain of convergent design variables into two subsets: first, into a feasible region for which the constraints are fulfilled; second, into an infeasible region for which they are violated. For divergent simulation runs, there is no solution, so the constraints cannot be evaluated. As sketched in Fig. 2, our improved approach enables us to put a preference on the exploration of convergent regions where the constraints are expected to be fulfilled.

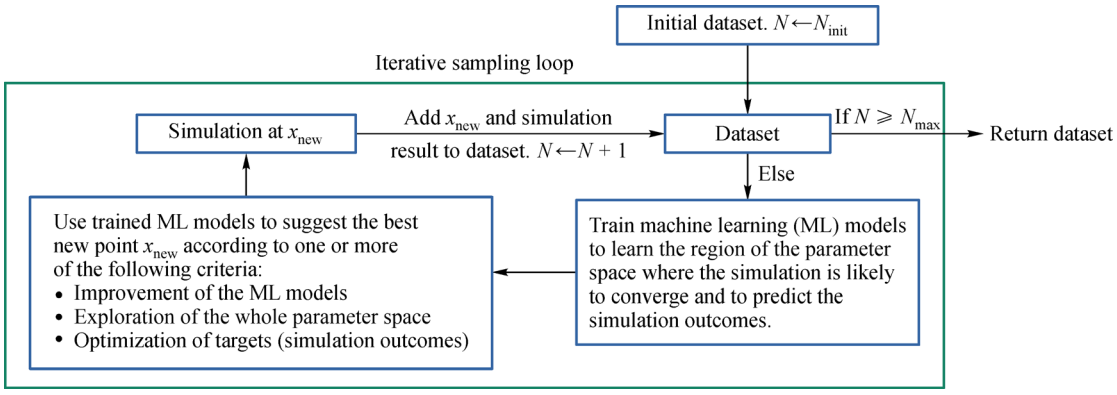

Fig. 1 Outline of the unconstrained adaptive sampling strategy [14].

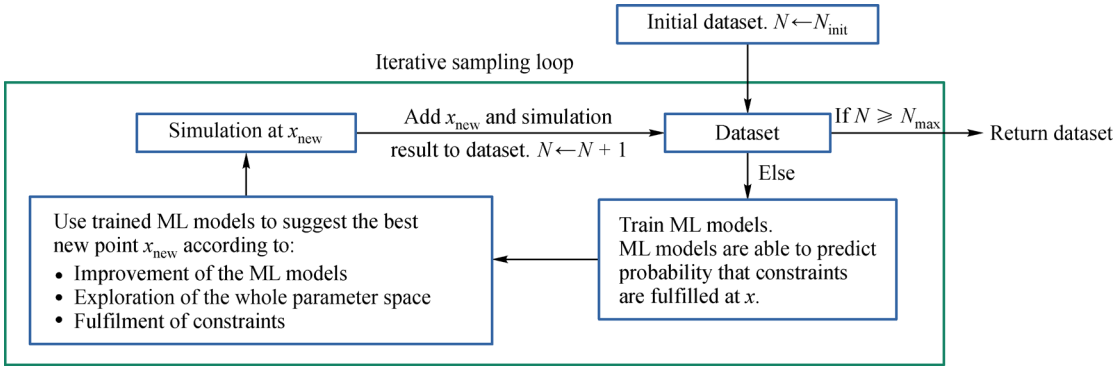

Fig. 2 Constrained adaptive sampling: starting from the results of flowsheet simulations, additional constraints can be imposed. Training ML models using the information how strong the constraints are violated allows to suggest new sampling points which are expected to fulfil the constraints. Subsequently, these new points x_{new} are evaluated by solving the system of equations of the flowsheet simulation.

Summarized, in addition to the binary information of convergence (i.e., divergent/convergent), our algorithm makes use of the continuous information on the violation of the constraints within the convergent domain. A point in the design variable space is feasible if and only if the simulation converges and the solution fulfills all constraints. However, results from convergent simulation runs which violate the constraints are still used to train ML models in order to guide the algorithm toward the convergent feasible region.

Moreover, we propose a refinement of the utility function of the algorithm of ref. [14], which is maximized to determine the sampling points, such that it becomes differentiable in the whole design variable space. This refinement allows the use of gradient-based solvers for finding the optimal next sample point and may prove useful for dealing with high-dimensional design variable spaces. As already mentioned above, we specifically do not consider optimization targets in this paper in contrast to ref. [14] and instead focus on a generalized feasibility exploration. Our improved framework of adaptive sampling with constraints is outlined in Fig. 2.

In general, the convergence of flowsheet simulations also depends on a reasonable initial guess for the whole vector of process variables. In practice, a common method is to use the result of the simulation of one set of design variables as an initial guess for the simulation of another set of design variables. Typically, the new simulation converges as long as the difference in the calculated design variables is not too large, but there is no guaranteed convergence. The proposed algorithm assumes that at least one convergent (but not necessarily feasible) point in the design variable space is known initially. This point can then also provide an initial guess for the whole vector of process variables. In this sense, the convergent domain identified by the method is the domain which converges given a certain initial guess.

The remaining paper is structured as follows. In Section 2, we present our proposed algorithm. We apply it to a toy example in Section 3 as a proof of concept. Subsequently, we apply it to a real-world industrial process in Section 4. For both the toy example and the real-world example, the new algorithm leads to a clear enhancement of the number of sampled points fulfilling the constraints in comparison with a random sampling approach. We conclude with a brief summary and outlook in Section 5.

2 Outline of the algorithm

In Section 2.1, we briefly summarize the adaptive sampling algorithm of ref. [14], which we denote as unconstrained adaptive sampling in the following. For this purpose, we make use of the notation of the aforementioned paper wherever possible. Subsequently, in Section 2.2, we present our extension to this method for adaptive sampling under additional constraints, which we denote as *constrained adaptive sampling*.

2.1 Unconstrained adaptive sampling

This section is meant as a brief review of the method [14] for the sake of completeness. The goal of the unconstrained adaptive sampling algorithm is the exploration of a hitherto unknown design variable space underlying a simulation as outlined in Fig. 1. The method relies on iterative improvement of the classification of the design variable space into a divergent and a convergent domain. This improvement is achieved by, in each step, optimizing a *utility function* estimating the "most useful" next point in the design variable space to be simulated. We show the main steps in algorithm 1.

The algorithm presumes a flowsheet simulation which can be described as a mapping $\mathcal{S}$ from a design variable space $\chi \subset \mathbb{R}^n$ to a result space $\eta \times \tau$ in the sense of

$$\begin{aligned} \mathcal{S}: \quad & \chi \rightarrow \eta \times \tau \\ & x \mapsto (y,t). \end{aligned} \tag{1}$$

The result space consists of the convergence space η = {convergent, divergent} and the optimization target space $\tau \subset \mathbb{R}$. Specifically, the convergence flag $y \in \eta$ indicates whether the simulation converges, whereas the optimization target $t \in \tau$ denotes an outcome (objective) from the simulation that a process engineer wants to optimize. As shown in algorithm 1, starting with an initial set D_{init} of data points (x,y,t) the algorithm suggests N new points by iteratively maximizing the utility function

$$\begin{aligned} & U(D_{\text{expl}},x,w) \\ & = \frac{w_s U_s(D_{\text{expl}},x) + w_o U_o(D_{\text{expl}},x) + w_r U_r(D_{\text{expl}},x)}{w_s \ + \ w_o \ + \ w_r}, \end{aligned} \tag{2}$$

with respect to the design variables x to determine the next sampling point

$$x_{\text{new}} = \text{argmax}_x U(D_{\text{expl}},x,w). \tag{3}$$

In each iteration loop as sketched in Fig. 1, the optimization problem, Eq. (3), is solved globally to provide a suggestion for a new design variable (i.e., sampling point) x_{new}. The results of this simulation are then added to the data set of already sampled points. In the following iteration, the augmented data set is used to re-train the ML models (which takes place in lines 13 and 14 of algorithm 1). It is important to emphasize that for practical purposes it is not necessary to find the exact solution of Eq. (3), but only a sufficiently good approximation. In fact, there might also be more than one solution to the problem, in which case it is sufficient to find an approximation to one of them.

The utility function U in Eq. (2) is by definition a map $\chi \rightarrow [0,1]$, which consists of a weighted sum of three utility components. Each of the three components reflects a different sampling behavior:

(1) U_s is the "border-finding term" which leads to a sampling at the border between convergent and divergent regions as given by y. It is defined as

$$\begin{aligned} U_s & \equiv U_s(D_{\text{expl}},x) \\ & = \frac{p(x)\ln p(x) + \Big(1-p(x)\Big)\ln\Big(1-p(x)\Big)}{\ln 2}, \end{aligned} \tag{4}$$

using an information entropic approach, where $p(x) \equiv p\Big(\mathcal{C}(D_{\text{expl}}),x\Big)$ is the probability of convergence predicted by the classification model $\mathcal{C}$. By definition, U_s becomes 1 on the border and reduces down to 0 further away from it such that $U_s \in [0,1]$. Here, D_{expl} denotes the set of already known data points (x,y,t).

(2) U_o is the "optimization term" which leads to a sampling of design points for which the optimization target t improves. By definition, $U_o \in [0,1]$, where a higher value corresponds to a more optimal and a lower value to a less optimal target. For a detailed definition of U_o we refer the reader to ref. [14]. We do not use this term in our approach, which we explain in more detail further below.

(3) U_r is the "exploration term" which leads to a sampling of design points in yet unsampled regions. It is defined as

$$U_r \equiv U_r(D_{\text{expl}},x) = 1 - \exp\left(-\gamma \min_{x' \in D_{\text{expl}}} ||x-x'||^2\right), \tag{5}$$

based on the distance to already sampled points. In analogy to the previous two terms, $U_r \in [0,1]$ with low values for design points with a close-by nearest neighbor and high values for design points with a far-away nearest neighbor in the set of already explored points.

By tuning the (nonnegative) weights w_s, w_o and w_r, the preference for the sampling behavior can be controlled according to the three criteria explained above: the higher the weight, the more emphasized the corresponding strategy.

Algorithm 1 Outline of the unconstrained adaptive sampling algorithm from ref. [14].

function EXPLORATION($D_{\text{init}},\chi,N_{\max},w$)
  $D_{\text{expl}} \leftarrow D_{\text{init}}$
  $N \leftarrow \text{size}(D_{\text{expl}})$
  while $N < N_{\max}$ **do**
    $x_{\text{new}} \leftarrow$ SUGGESTIONUNCONSTRAINED(D_{expl},χ,w)
    $D_{\text{expl}} \leftarrow D_{\text{expl}} \cup \{$ SIMULATION(x_{new})$\}$
    $N \leftarrow N + 1$
  end while
  return D_{expl}
end function

function SUGGESTIONUNCONSTRAINED(D_{expl},χ,w)
  $\mathcal{C} \leftarrow$ TRAINCLASSIFIER(D_{expl})
  $\mathcal{R}_t \leftarrow$ TRAINREGRESSOR(D_{expl})
  function UTILITY($x,D_{\text{expl}},\mathcal{C},\mathcal{R}_t,w$)
    $u \leftarrow \left(U_s(\mathcal{C},x),U_o(\mathcal{R}_t,x),U_r(D_{\text{expl}},x)\right)^T$
    return $w^T u/||w||_1$
  end function
  return $argmax_{x\in\chi}$ UTILITY($x,D_{\text{expl}},\mathcal{C},\mathcal{R}_t,w$)
end function

2.2 Constrained adaptive sampling

For our proposed algorithm, we take the framework from the unconstrained adaptive sampling algorithm from Section 2.1 and propose the following changes of the utility function (Eq. (2)): 1) The utility function is extended by an additional term that encourages the fulfillment of user-defined constraints; 2) The border-finding term is adapted to a new classifier; 3) The optimization term is removed; 4) The exploration term is refined to be differentiable in the whole design variable space.

We outline the new method in Fig. 2 and algorithm 2, which are based on Fig. 1 and algorithm 1, respectively. In the following, we describe the changes of the utility function in more detail.

For our method, we presume flowsheet simulations of the form

$$\begin{aligned} \mathcal{S}: \quad & \chi \rightarrow \eta \times \varphi \\ & x \mapsto (y,f) \end{aligned}, \tag{6}$$

with p-dimensional simulation outcomes $f \in \varphi \subset \mathbb{R}^p$ instead of scalar outcomes as in Eq. (1). We furthermore presume that constraints

$$f \in D_f \subset \varphi, \tag{7}$$

can be imposed on these simulation outcomes. Hence, we call f the *constraint function values*.

In addition to the classification of convergent and divergent results, our extended approach allows us to also classify simulation results according to feasible (constraints fulfilled) and infeasible (constraints not fulfilled) simulations. Mathematically, we describe this by assigning three elements to the classification space

$$\eta = \{\text{divergent, feasible, infeasible}\}, \tag{8}$$

where,

$$y = \begin{cases} \text{feasible} & \text{if convergent and } f(x) \in D_f \\ \text{infeasible} & \text{if convergent and } f(x) \notin D_f \\ \text{divergent} & \text{otherwise} \end{cases}, \tag{9}$$

based on the constraint function values f. The probability $p(x)$ in the border-finding term U_s of Eq. (4) thus needs to be replaced by the probability of convergence, where feasible/infeasible are treated as one single class "convergent". The border-finding term thus has its maximum value 1 at the border divergent/convergent (irrespective whether it is feasible or not). In order to guarantee differentiability of U_s, $p(x)$ must be differentiable.

Since we consider a generalized, optimization-free feasibility exploration and therefore do not take optimization goals into account for the sampling strategy, we remove the optimization term U_o from the utility function (Eq. (2)). We remark, however, that it would in principle also be possible to keep U_o if it is provided in the form of a differentiable function like a differentiable prediction of a surrogate model for the optimization target.

Instead of the optimization target, we take constraints into account for the sampling. For this purpose, we include the additional term

$$U_c : \chi \rightarrow [0,1], \tag{10}$$

with weight w_c into Eq. (2). The purpose of U_c is to drive the suggested new points toward the region where the constraints are expected to be fulfilled. A suitable utility measure U_c for this purpose can be constructed as follows. In each loop iteration of the algorithm, we train a regression model $\mathcal{R}_f$ (in addition to the classifier $\mathcal{C}$), which provides probability distributions for the values of f at x, i.e.,

$$\mathcal{R}_f : x \mapsto \pi_f(f|x), \tag{11}$$

such that the predicted expectation value $\hat{f}$ of f is given by

$$\hat{f} = \int_{\phi} f \pi_f(f|x) \mathrm{d}f. \tag{12}$$

We can then compute the predicted probability

$$p(f \in D_f|x) = \int_{D_f} \pi(f|x) \mathrm{d}f, \tag{13}$$

that the constraints are fulfilled at x. If $f \in D_f$ can be reformulated in terms of inequality constraints and $\pi(f|x)$

is a multivariate Gaussian distribution, the integral in Eq. (13) can be evaluated explicitly as shown in Appendix A (cf. Electronic Supplementary Material, ESM). In all examples presented in this paper, we use GPR (Gaussian process regression) [17] as the regression model $\mathcal{R}_f$. In this case, $\pi(f|x)$ is a Gaussian distribution and we use the result of Appendix A for all of our applications. Otherwise, a numerical approximation might be necessary to solve the integral. Note that GPR is another name for Kriging models, which have already been successfully used in previous feasibility exploration approaches [3,4].

Since $p(f \in D_f|x) \in [0,1]$ quantifies the expectation that the constraints will be fulfilled at x, we define

$$U_c \equiv U_c(D_{\text{expl}},x) \equiv p(f \in D_f|x), \tag{14}$$

as the utility measure of exploring feasible domains. Similar expressions are also used in Bayesian optimization with inequality constraints [18]. For the case of GPR, as used throughout this paper, $p(f \in D_f|x)$ is differentiable with respect to x.

One key element of the presented algorithm is the optimization of the utility function in the sense of Eq. (3). For best results, we apply global optimization techniques to solve this nonlinear optimization problem. However, to provide the optimizer with a function that is as well-behaved as possible locally, we propose a refined exploration term in the utility function, which is differentiable in the whole design variable space. For this purpose, we reconsider the originally proposed exploration term [14,19] of Eq. (5), which, due to the local non-differentiability of the distance $||x-x'||$ of the design point x to its nearest neighbor x', violates the differentiability requirement. To overcome this problem, we replace it by the product

$$\begin{aligned} U_r' &\equiv U_r'(D_{\text{expl}},x) \\ &= \prod_{x' \in D_{\text{expl}}} 1-\exp(-\gamma||x-x'||_{[0,1]}^r), \end{aligned} \tag{15}$$

with an exponent $r \geqslant 2$ and $\gamma > 0$. Note that with the norm $||.||_{[0,1]}$ we do not denote the usual Euclidean norm, but the Euclidean norm in a space where all entries in the $x' \in D_{\text{expl}}$ have been linearly rescaled and shifted to the interval $[0,1]$. The maximal distance between two points in $x' \in D_{\text{expl}}$ is then $\sqrt{\dim \chi} = \sqrt{n}$. The parameter γ has the dimension of an inverse length to the power r (we choose $r = 2$ in all of our examples). In order for $U_r' \in [0,1]$ to not converge to an almost constant function for a large number of sampled points (large set D_{expl}), γ has to be adapted to the number $N = |D_{\text{expl}}|$ of sampled points. We choose this adaptive value as follows. Suppose all N sampled points have the same distance $\overline{d}$ from the test point x and we want the repulsion term U_r' at this point to have the value $\overline{U_r'}$. (Recall that we only consider distances in the rescaled input space. The chosen value of $\overline{d}$ has thus to be compared to the maximal distance $\sqrt{n}$ between two already known points.) To achieve this, we set

$$\gamma = -\frac{1}{\overline{d}^r} \ln\left(1-\overline{U_r'}^{1/N}\right). \tag{16}$$

We define $\overline{U_r'} = 1/2$ as the value exactly in the center of the range of U_r'. Moreover, we allow for an overall factor γ_0 to allow a user to choose more or less dense explorations. Thus,

$$\gamma \equiv \gamma(N) = -\frac{\gamma_0}{\overline{d}^r} \ln(1-1/2^{1/N}). \tag{17}$$

The length scale $\overline{d}$ and the dimensionless factor γ_0 are parameters of the algorithm which can be selected by the user. For $\gamma_0 = 1$, $\overline{d}$ is the distance which fulfills

$$||x-x'||_{[0,1]} \geqslant \overline{d} \quad \forall x' \in D_{\text{expl}} \Rightarrow U_r'(x) > 1/2. \tag{18}$$

Summarized, the new utility function reads

$$\begin{aligned} &U(D_{\text{expl}},x,w) \\ &= \frac{w_s U_s(D_{\text{expl}},x) + w_r U_r'(D_{\text{expl}},x) + w_c U_c(D_{\text{expl}},x)}{w_s + w_r + w_c}, \end{aligned} \tag{19}$$

with the border-finding term U_s (Eq. (4)), the refined exploration term U_r' (Eq. (15)) and the constraint term U_c (Eq. (14)). According to this utility function, algorithm 2 samples points where the border between convergent and divergent regions is expected to be, no design points have been evaluated before and the constraints are fulfilled.

All three goals are assessed by the corresponding functions and weighted by the corresponding weights such that a single scalarized utility goal arises.

Algorithm 2 Outline of our proposed constrained adaptive sampling algorithm. The new function SUGGESTIONCONSTRAINED replaces SUGGESTIONUNCONSTRAINED in Line 5 of Algorithm 1.

```
1: function SuggestionConstrained(D_expl, χ, D_f, w)
2:     C ← TrainClassifier(D_expl)
3:     R_f ← TrainRegressor(D_expl)
4:     function Utility(x, D_expl, C, R_f, D_f, w)
5:         u ← (U_s(C,x), U_r'(D_expl,x), U_c(R_f,D_f,x))^T
6:         return w^T u / ||w||_1
7:     end function
8:     return argmax_{x∈χ} Utility(x, D_expl, C, R_f, D_f, w)
9: end function
```

3 Application to a toy example

As a proof of principle, we demonstrate the performance of our method, algorithm 2, on a simple toy example in n

dimensions. First, we define the toy example. Subsequently, we discuss the performance of the algorithm for $n \geqslant 2$.

In all examples presented in this paper, we use GPR [17] as the regression model $\mathcal{R}_f$. As the classification model $\mathcal{C}$ we use the recently developed CASIMAC approach [20]. For global optimization of the utility function, we use the stochastic differential evolution method with the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm with box constraints (L-BFGS-B) [21,22] to further refine ("polish") the results. In this polishing step, explicit gradients of the utility function are provided to the L-BFGS-B algorithm. We realize our numerical experiments using Python with the help of SciPy [23], scikit-learn [24], and GPy [25].

3.1 Definition of the toy example

We consider an example on the design variable space $\chi = [-2,2]^n$ with $n \geqslant 2$ and define the simulation outputs $\mathcal{S}_n$, see Eq. (6), as

$$\begin{aligned} \mathcal{S}_n : \quad & \chi \longrightarrow \eta \times \varphi \\ & x \longmapsto (y,f), \quad \text{where} \\ & f(x) \equiv x_1 x_2. \end{aligned} \tag{20}$$

The feasibility flags are assigned as

$$y = \begin{cases} \text{feasible} & \text{if } ||x||_2^2 \leqslant 1 \text{ and } f(x) > 0.1, \\ \text{infeasible} & \text{if } ||x||_2^2 \leqslant 1 \text{ and } f(x) \leqslant 0.1, \\ \text{divergent} & \text{if } ||x||_2^2 > 1. \end{cases} \tag{21}$$

Summarized, the convergent domain is the unit sphere centered at the origin and we divide the convergent domain into a feasible and an infeasible space by imposing the constraint

$$f(x) = x_1 x_2 \in D_f \equiv [0.1,\infty), \tag{22}$$

i.e., $f(x) \geqslant 0.1$. Figure 3 shows the design variable space χ for $n = 2$ dimensions. Note that the feasible domain is not connected, which imposes an additional challenge to the exploration.

3.2 Two-dimensional adaptive sampling

To illustrate the qualitative behavior of the algorithm, we use the $n = 2$ toy example and generate a uniformly distributed random initial configuration D_{init} of 15 points in the lower left quadrant. We then adaptively sample points with different choices of the weights $w = (w_s, w_r, w_c)$ of the three terms U_s, U_r' and U_c of the utility function. The parameters of the exploration term U_r' were chosen as $\gamma_0 = 10$, $r = 2$ and $\overline{d} = 1/4$. The utility function was maximized using differential evolution with a population

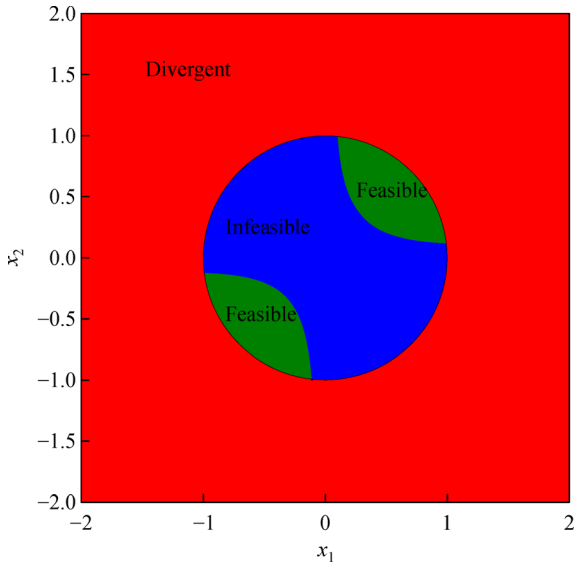

Fig. 3 The divergent (red), convergent feasible (green) and convergent infeasible (blue) regions of the design variable space χ of the toy example for $n = 2$ dimensions.

size of 128 and maximal number of iterations of 1024.

We emphasize that the problem of finding a sufficient number of non-divergent (in the sense of Eqs. (9) and (21)) initial points in D_{init} is not solved by algorithm 2. If random sampling does not yield non-divergent simulation runs, the user must provide at least one non-divergent initial point (irrespective whether feasible or infeasible) to enable training of the necessary ML models. However, we can straightforwardly provide suitable initial configurations for this toy example.

The results of our experiments are shown in Fig. 4. The first column corresponds to algorithm 2 with inactive constraint term U_c (i.e., $w_c = 0$). The remaining three columns are generated with nonzero constraint weight w_c leading to preferred sampling of points in the feasible region. Note, however, that the border-finding term is constructed such that the border between the divergent and convergent region (irrespective whether feasible or infeasible) is sampled. Therefore, even for nonzero w_c, the algorithm does not search for the border between the feasible and infeasible region. This can be seen best from the plot in the last row of column three in Fig. 4. The border of the feasible domain there is sampled well only in the part where it overlaps with the border to the divergent domain. The algorithm could straightforwardly be adapted to fully explore the border between the feasible and infeasible domain by training the classifier $\mathcal{C}$ on the two classes "feasible" and "infeasible/divergent" instead of "feasible/infeasible" and "divergent". In the fourth column, the exploration term has highest weight. This leads to sampling of a large part of the design variable space while showing a preference toward the feasible (green) region.

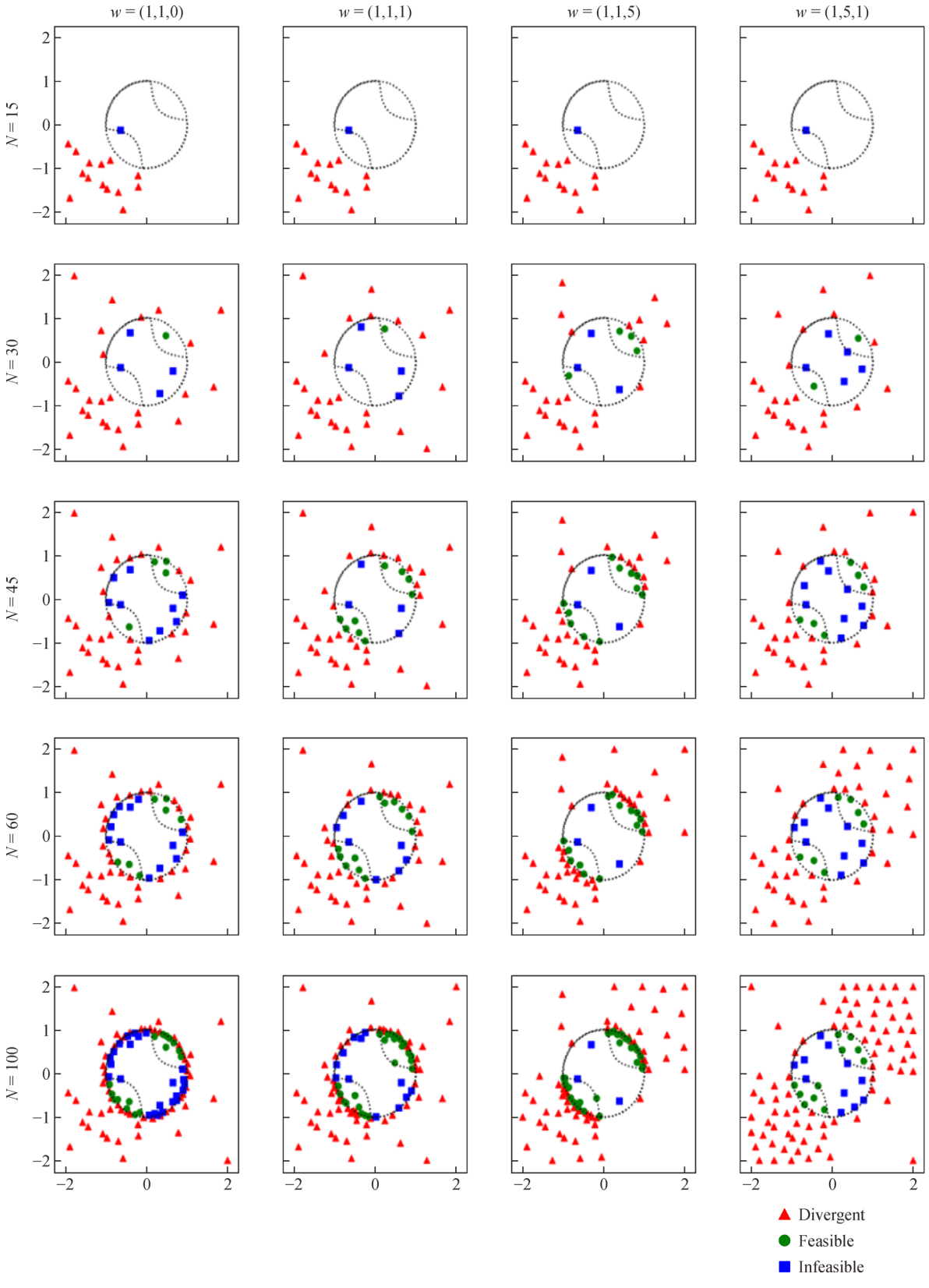

Fig. 4 An illustration of the behavior of the algorithm for the $n = 2$ toy example. As initial state D_{init}, we use 15 randomly generated points in the lower left quadrant. Each column of the above matrix of plots corresponds to one run of the sampling algorithm. The column titles show the chosen weights $w = (w_s,w_r,w_c)$. The different rows show the generated points at different stages, the total number of sampled points (including the 15 initial points) being shown on the very left of the figure. The three different types of points are plotted as red triangles (divergent), blue squares (convergent but infeasible) and green circles (convergent and feasible), respectively. The black dashed lines show the boundaries of the regions.

To quantitatively evaluate the performance of the algorithm, we additionally perform 50 runs of the algorithm with different random initial configurations. For each run, we record the numbers of divergent, feasible and infeasible points at all stages of sampling. Averages over multiple runs are considered to reduce the dependence on the initial data set. The initial configurations are chosen as 15 random points (different for each of the 50 runs) uniformly distributed in the lower left quadrant $\chi_{\text{init}} = [-2,0]^2$. The restriction of the initial configuration lower left quadrant is done in order to demonstrate the ability of the algorithm to nevertheless sample the border of the whole non-divergent region. For each of these 50 initial configurations 100 points are sampled adaptively, the results are shown in Fig. 5.

While for $(w_s,w_r,w_c) = (1,1,0)$ the number of sampled infeasible points is higher than the number of feasible points, this behavior changes for $w_c = 1$ and $w_c = 5$, where the number of feasible points exceeds the number of infeasible ones for sufficiently large total point numbers. For $w_c = 5$ after about 10 adaptive sampling steps no infeasible points are sampled anymore, while the number of feasible points increases approximately linearly.

3.3 Higher-dimensional adaptive sampling

In this section, we investigate the ability of algorithm 2 to sample the border divergent/convergent and the effect of the constraint driving term U_c on the number of sampled feasible points in high dimensions. For this purpose, we compare runs of the algorithm with $(w_s,w_r,w_c) = (1,1,0)$ and $(w_s,w_r,w_c) = (1,1,1)$ for different dimensions n of the design variable space χ.

Random sampling in n dimensions would give a chance of hitting a non-divergent point of

$$\frac{\text{Vol(unit sphere)}}{\text{Vol}(\chi)} = \frac{\text{Vol(unit sphere)}}{\text{Vol}([-2,2]^n)} = \frac{\pi^{\frac{n}{2}}}{4^n \Gamma\left(1+\frac{n}{2}\right)}, \tag{23}$$

which is 0.20 for $n = 2$, 2.43×10^{-6} for $n = 10$ and 2.35×10^{-14} for $n = 20$. Due to the low probabilities for large n, and since the algorithm needs at least one non-divergent point in D_{init}, we have to make the regions $\chi_{\text{init}} \subseteq \chi$ (from which the initial configurations D_{init} are drawn randomly) sufficiently small. For each of the 50 runs we thus take a different random initial configuration D_{init} consisting of 30 points in a region $\chi_{\text{init}} \subseteq \chi$ as shown in Table 1.

We then adaptively sample 100 points according to algorithm 2 with $(w_s,w_r,w_c) = (1,1,0)$. The parameters of the exploration term U_r' are chosen as $\gamma_0 = 10$, $r = 2$ and $\overline{d} = 1/4$. We use differential evolution to optimize the

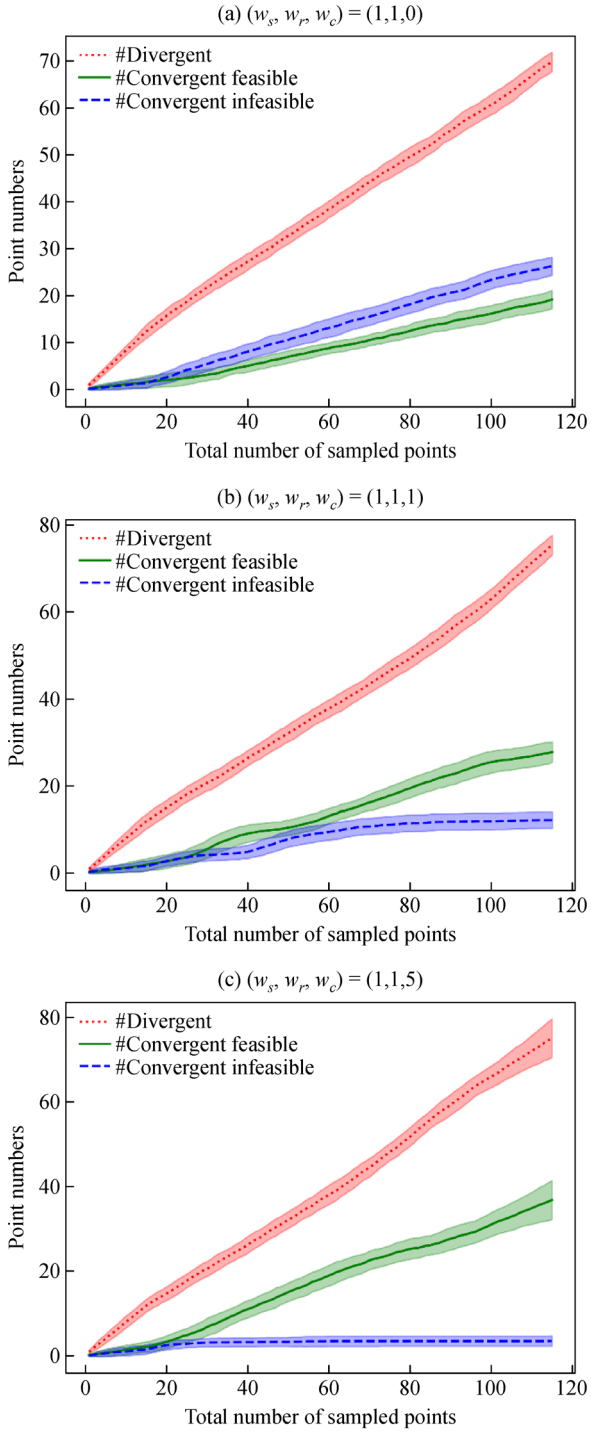

Fig. 5 The number of divergent (red), convergent feasible (green) and convergent infeasible (blue) points as a function of the total number of sampled points for the $n = 2$ toy example. For a fixed $w = (w_s,w_r,w_c)$, we perform 50 runs of algorithm 2 with different random initial configurations D_{init} consisting of 15 points each. The lines are the averages of the 50 runs and the shaded regions are the 1σ-error bands. The results for $w = (1,1,0),(1,1,1)$ and $(1,1,5)$ are shown in plots (a), (b) and (c), respectively.

Table 1 The regions $\chi_{init} \subseteq \chi$ from which the initial configurations D_{init} for the tests of the n-dimensional toy example are drawn randomly

Toy example	$n = 2$	$n = 3$	$n = 5$	$n = 10$	$n = 20$
χ_{init}	$[-2,2]^2$	$[-1.5,1.5]^3$	$[-0.8,0.8]^5$	$[-0.6,0.6]^{10}$	$[-0.4,0.4]^{20}$

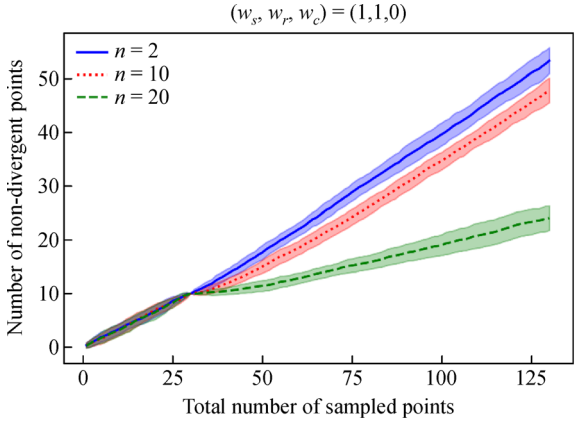

Fig. 6 Mean number of non-divergent (feasible + infeasible) points with 1σ-error bands for the dimensions $n = 2$, $n = 10$ and $n = 20$ of the toy example. The subsets $\chi_{init} \subseteq \chi$ of the design variable space from which the initial configurations are drawn randomly (see Table 1) are chosen in such a way that a comparable number of non-divergent points in D_{init} is achieved for all dimensions n (The curves for $n = 3$ and $n = 5$ are not shown for the sake of clarity. They run between those for $n = 2$ and $n = 10$, as expected).

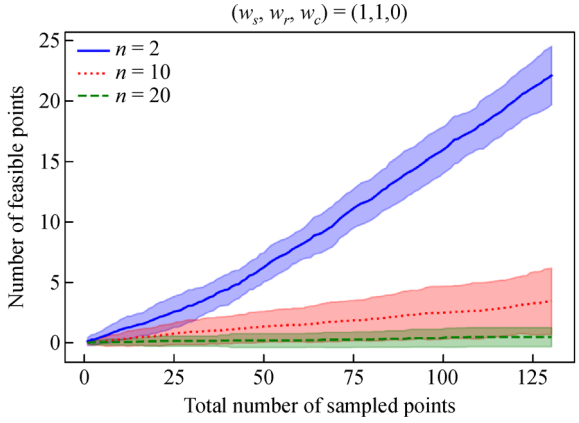

Fig. 7 Mean number of feasible points with 1σ-error bands for the dimensions $n = 2$, $n = 10$ and $n = 20$ of the toy example. The subsets $\chi_{init} \subseteq \chi$ of the design variable space from which the initial configurations are drawn randomly (see Table 1) are chosen in such a way that a comparable number of non-divergent points in D_{init} is achieved for all dimensions n (The curves for $n = 3$ and $n = 5$ run between those for $n = 2$ and $n = 10$ and are not shown for the sake of clarity).

utility function with a population size of 512 and a maximal number of iterations of 2048. For each choice of dimension, we compute average and variance of the number of non-divergent sampled points as a function of the total number of sampled points based on 50 runs. The results are shown in Figs. 6 and 7. We find that even for dimensions as high as $n = 20$ the algorithm is capable of finding non-divergent points. However, without the constraint term, the number of found feasible points is very low for $n = 10$ and almost zero for $n = 20$, see Fig. 7. Repeating the same analysis with active constraint term using $(w_s,w_r,w_c) = (1,1,1)$ leads to the results shown in Fig. 8. From this figure we see that also in $n = 20$ dimensions the algorithm is capable of finding feasible points.

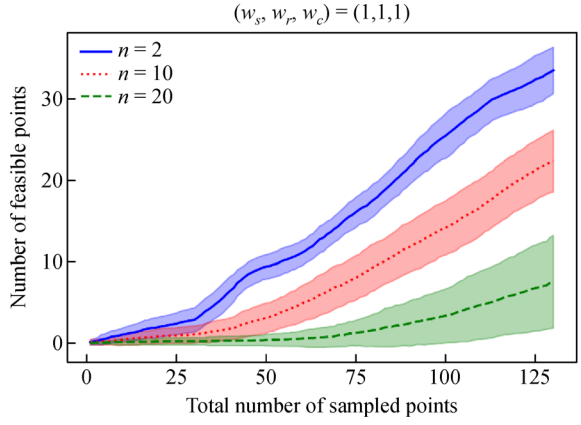

Fig. 8 Mean number of feasible points for the analysis with nonzero constraint weight with 1σ-error bands for the dimensions $n = 2$, $n = 10$ and $n = 20$ of the toy example. The subsets $\chi_{init} \subseteq \chi$ of the design variable space from which the initial configurations are drawn randomly (see Table 1) are chosen in such a way that a comparable number of non-divergent points in D_{init} is achieved for all dimensions n. The curves for $n = 3$ and $n = 5$ are very similar to the one for $n = 2$ and are therefore not shown for the sake of clarity.

4 Application to a pressure swing distillation

In this section, we apply algorithm 2 to a real-world, industrially relevant flowsheet simulation. We specifically consider the well-known pressure swing distillation of a mixture of chloroform and acetone, which has already been discussed in ref. [14]. In the following, we briefly summarize this chemical process and describe the corresponding simulation. We subsequently show exemplary results of applying algorithm 2 to the presented scenario. Finally, we perform a statistical analysis of the number of feasible and infeasible points sampled for this example for different weights (w_s,w_r,w_c) of the utility function.

4.1 The pressure swing distillation

A binary mixture of acetone and chloroform exhibits a pressure dependent azeotropic point: as soon as this point is reached, such a mixture cannot be separated by distillation anymore (at the azeotropic point the concentrations of acetone and chloroform are equal in the liquid and vapor phase, respectively). Pressure swing distillation makes use of the pressure dependency of the azeotropic point (the higher the pressure, the higher the chloroform concentration in the azeotrope) to circumvent this limitation. The schematic sketch (flowsheet) of a plant for our example is shown in Fig. 9. We start on the very left of the flowsheet with the feed, a mixture of chloroform and acetone, which contains more chloroform than the azeotrope at the pressure $p = 1$ bar. This mixture, being mixed with a recycle stream from column C2, enters the first distillation column C1 operating at 1 bar. There chloroform will enrich in the vapor phase and can be collected in the top (distillate) stream of C1. The bottom liquid (sump) stream of C1 has a composition close to the azeotropic point at 1 bar. This stream is fed into the second column C2 at a higher pressure of 10 bar. Since the input stream of C2 contains less chloroform (and thus more acetone) than at the azeotropic point at 10 bar, in C2 chloroform will enrich in the liquid phase and the product (distillate) stream will be rich in acetone. Finally, the bottom (sump) stream of C2, which has a composition close to the azeotropic point at 10 bar, is recycled by combining it with the feed mixture.

4.2 Simulation of the pressure swing distillation

In the flowsheet simulation each of the two columns of Fig. 9 is modeled with an equilibrium stage model [26] with 28 stages, the feed stage being stage 14. For describing the interaction between acetone and chloroform we use the non-random two-liquid model [27] with the model parameters given in ref. [28]. The complete process simulation involves the solution of about 400 coupled nonlinear equations. The simulations are carried out using the BASF in-house flowsheet simulator *Chemasim*. As inputs of the simulation, we use the purities (mass fractions) m_{ac} and m_{cl} of the acetone (column C2) and chloroform (column C1) distillate (product) streams as indicated in Fig. 9 as well as the reflux ratios r_1, and r_2 of C1 and C2, respectively:

$$x = \begin{pmatrix} m_{ac} \\ m_{cl} \\ r_1 \\ r_2 \end{pmatrix}. \tag{24}$$

As the design variable space we choose

$$\chi = [0.1, 1.0] \times [0.8, 1.0] \times [5, 35] \times [5, 35]. \tag{25}$$

The mass flow of the feed stream of the plant (which is combined with the recycle stream before it enters column C1) is set to 1250 $\mathrm{kg \cdot h^{-1}}$ and has a constant composition of 86 mass percent chloroform and 14 mass percent acetone.

We consider a simulation result of Chemasim as convergent if the simulation numerically converges to a physically feasible point and as divergent otherwise. Extending the analysis of [14], we impose an additional constraint. One important result of the simulation — beyond the information whether it was successful and lead to a physically feasible result — is the total heat duty $\dot{Q}$ needed to operate the plant in a stationary state. Therefore, we impose the constraint $\dot{Q} \leqslant 1500$ kW on the total heat duty, i.e.,

$$f(x) \equiv \dot{Q}(x) \in D_f = [0, 1500\ \mathrm{kW}]. \tag{26}$$

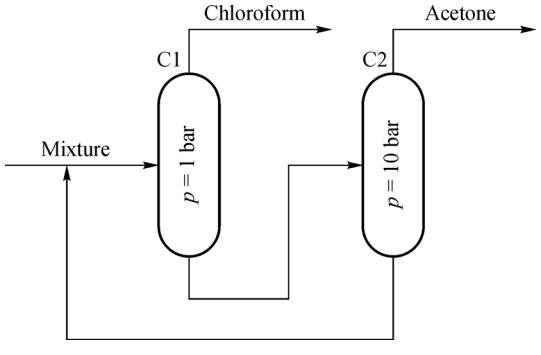

Fig. 9 Simplified flowsheet for the pressure swing distillation of a mixture of chloroform and acetone. A mixture containing 86 mass percent chloroform and 14 mass percent acetone is fed into the column C1 operating at 1 bar. Since the feed contains more chloroform than the azeotropic point at 1 bar, chloroform will enrich in the top (distillate) stream. The bottom liquid (sump) stream of C1 is fed into column C2 operating at 10 bar. The distillate stream of C2 is rich in acetone. The bottom liquid stream of C2 is recycled by combining it with the input mixture stream.

4.3 Adaptive sampling of the flowsheet simulation

In the following, we apply algorithm 2 to a simulation of the pressure swing distillation under constraints. The parameters of the exploration term U_r' are chosen as $\gamma_0 = 1$, $r = 2$ and $\overline{d} = 1/4$. The optimizer used for maximizing the utility function is differential evolution with a population size of 15 and a maximal number of 100 iterations.

A qualitative illustration of the application of algorithm 2 to the pressure swing distillation can be found in Fig. 10. As initial state, we use 10 uniformly distributed random points in the design variable space given by Eq. (25). The first column corresponds to algorithm 2 with inactive constraint term (i.e., $w_c = 0$). The other two columns are generated with nonzero constraint weight w_c leading to preferred sampling of points in the region fulfilling the

constraint Eq. (26) on the total heat duty. To quantitatively investigate the performance of the algorithm, we apply algorithm 2 for 50 different initial random configurations D_{init} of 10 data points each. We adaptively sample 100 points for each of these initial configurations. For each D_{init} we extract the numbers of divergent, convergent feasible and convergent infeasible points as a function of the total number of points. Finally, we compute the mean and variance of the three resulting curves based on the 50 initial random configurations. The results are shown in Fig. 11.

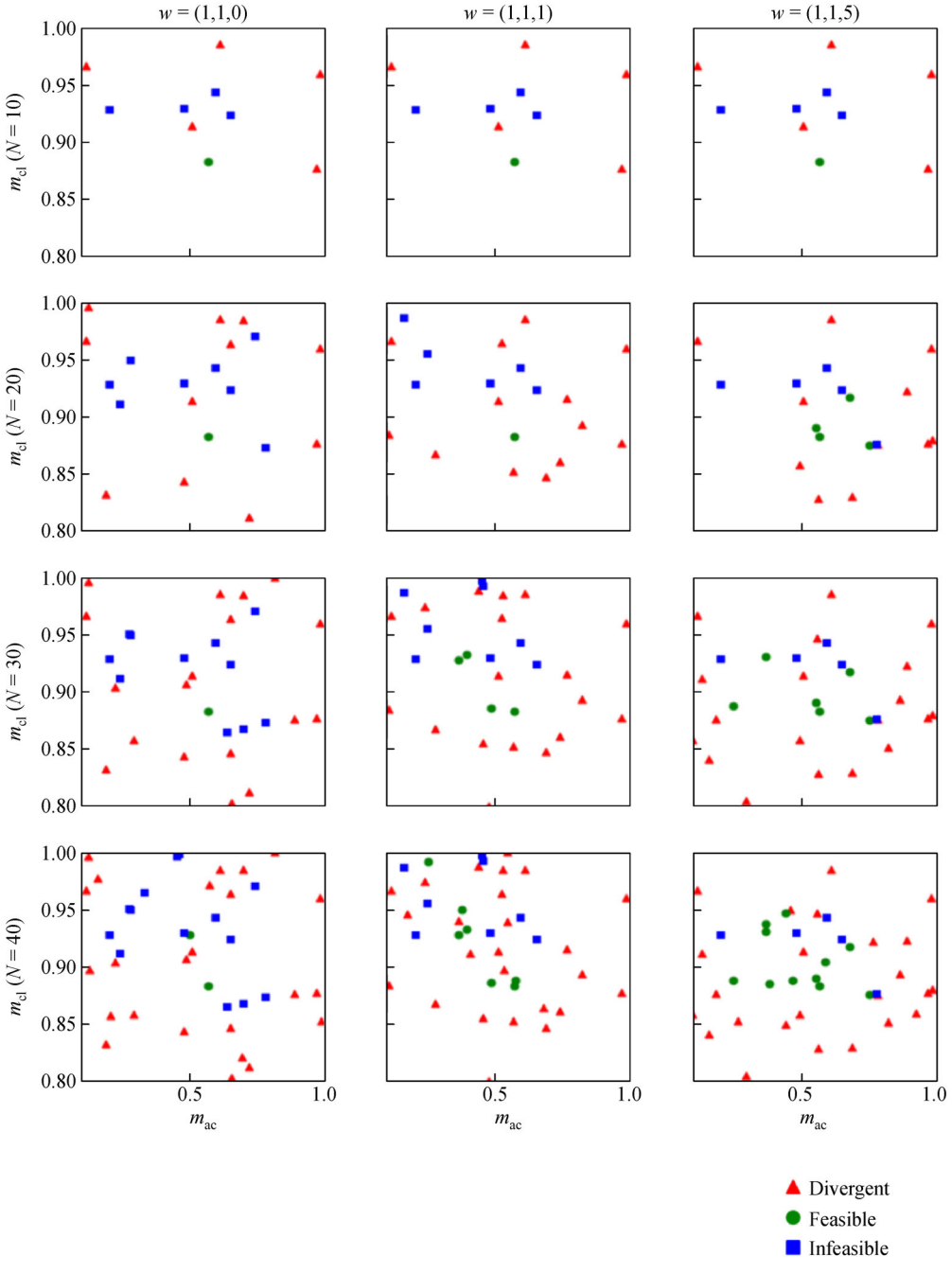

Fig. 10 An illustration of the behavior of the algorithm for the chloroform/acetone pressure swing distillation. As initial state D_{init} we use 10 randomly generated points in the four-dimensional design space χ. The plots themselves show projections into the $(m_{\text{ac}},m_{\text{cl}})$-plane. Each column of the above matrix of plots corresponds to one run of the sampling algorithm. The column title shows the chosen weights $w = (w_s,w_r,w_c)$. The different rows show the generated points at different stages, the total number of sampled points (including the 10 initial points) being shown on the very left of the figure. The three different types of points are plotted as red triangles (divergent), blue squares (convergent but infeasible) and green circles (convergent and feasible), respectively.

As can be seen from the plots there, all three point numbers increase more or less linearly. Subsequently turning on the constraint weight from $w_c = 0$ to $w_c = 1$ and $w_c = 5$ increases the slope of the green (feasible) curve at the cost of the blue (infeasible) one.

Additionally, we compare our sampling approach to a random sampling strategy. With the same 50 initial random seeds as before we now randomly sample in the four-dimensional design variable space χ and compute the means of the three curves, which in the limit of infinitely many experiments would be linear.

This comparison is shown in Fig. 12 for the feasible and infeasible points up to a total point number of 200. As can be seen from this figure, already for $(w_s,w_r,w_c) = (1,1,0)$, which corresponds to algorithm 2 with inactive constraint term, the numbers of feasible and infeasible (i.e., the non-divergent) points is higher than for mere random sampling. Turning on the constraint weight w_c strongly enhances the sampling rate of convergent feasible (green) points. While random sampling on average needs more than 150 samples to find 10 convergent feasible points, algorithm 2 with $(w_s,w_r,w_c) = (1,1,1)$ on average needs only about 50 samples. These results demonstrate the ability of our algorithm to enhance the sampling rate of points fulfilling a constraint on one of the simulation outputs in a real-world industrial example.

5 Conclusions and outlook

We have presented an adaptive sampling strategy based on ML models to explore the convergent and feasible domains of design variables in flowsheet simulations with (user-defined) inequality constraints. Our work is an extension of a recently published adaptive sampling algorithm which does not consider constraints. The behavior of our algorithm can be tuned to the use case at hand, for example as a space-filling exploration or as a reliable exploration of the border separating the convergent and divergent regions. The performance of our method has been compared to random sampling schemes, which were found to be significantly inferior.

We remark that the trained classification model can be used to estimate, for each point in the design space, whether a corresponding simulation is expected to be convergent or not without actually running the simulation. After the adaptive sampling has been performed, random or space filling samplings (for example using Sobol sequences or Latin hypercubes) can consequently be performed in regions where convergence is expected based on the trained classifier. With such an approach, the expected convergent region can be sampled in more detail using the information gathered from the adaptive sampling strategy.

A prospective application of the presented algorithm is the improvement of the convergence behavior of flowsheet

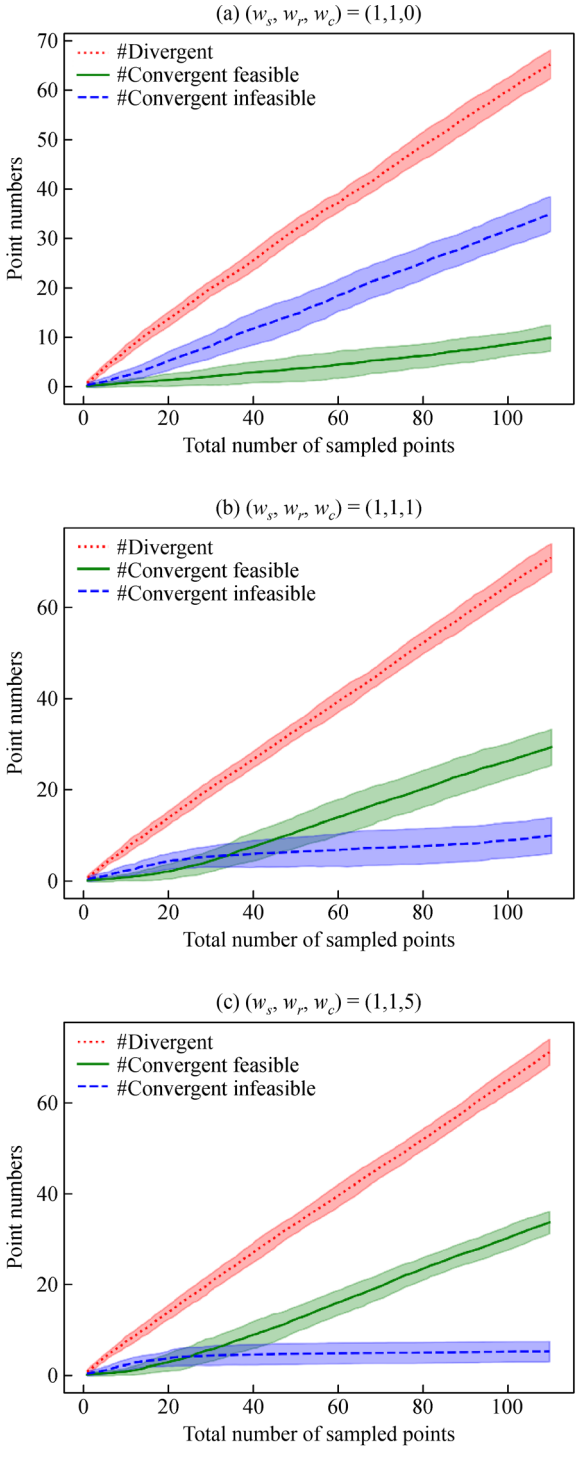

Fig. 11 The number of divergent (red), convergent feasible (green) and convergent infeasible (blue) points as a function of the total number of sampled points for the pressure swing distillation example. For a fixed $w = (w_s,w_r,w_c)$, we perform 50 runs of algorithm 2 with different random initial configurations D_{init} consisting of 10 points each. The lines are the averages of the 50 runs and the shaded regions represent the 1σ-error bands. The results for $w = (1,1,0)$, $(1,1,1)$ and $(1,1,5)$ are shown in plots (a), (b) and (c), respectively.

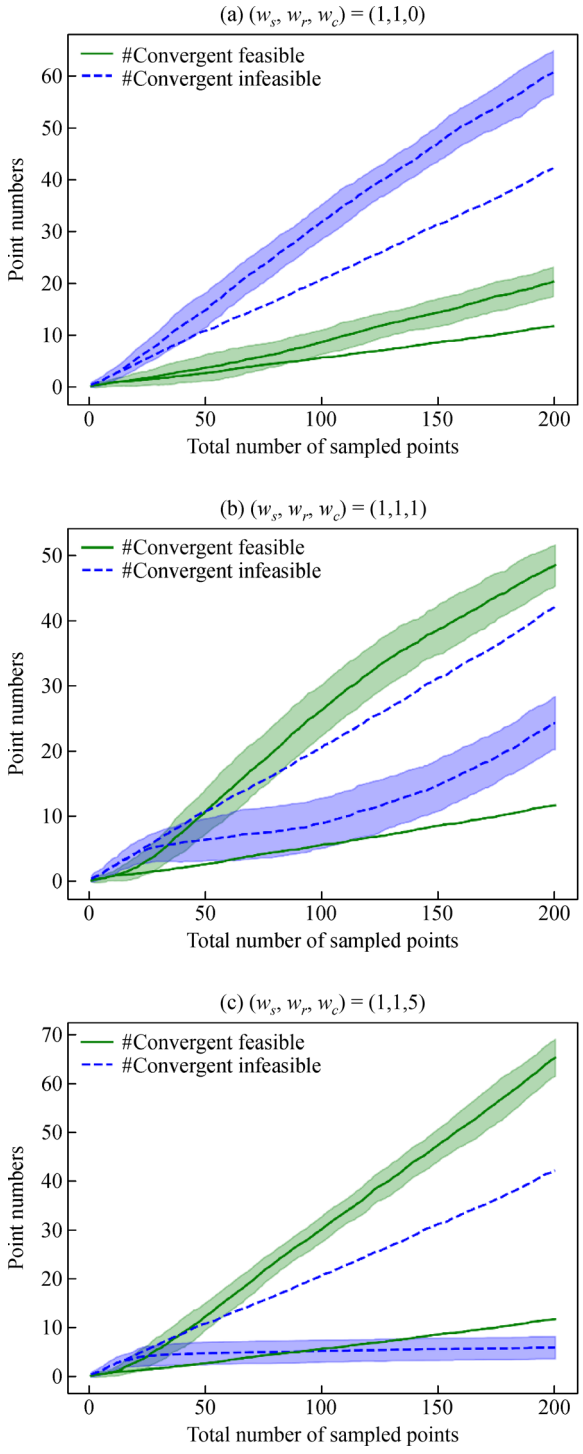

Fig. 12 Comparison of the results of algorithm 2 (lines with 1σ-error bands) to mere random sampling (lines without error bands; for the sake of clarity, the error bars for the random sampling results are not shown) for the pressure swing distillation example. For a fixed $w = (w_s,w_r,w_c)$, we perform 50 runs with different random initial configurations D_{init} consisting of 10 points each. The results for $w = (1,1,0),(1,1,1)$ and $(1,1,5)$ are shown in plots (a), (b) and (c), respectively.

simulations under constraints. For this purpose, the database of convergent simulation runs aggregated during the sampling process may be used to suggest better starting values for the complete vector of process variables. In particular, better starting values can help the solver to more easily converge to a solution. As a consequence, design variables that have formerly led to a divergent simulation run, can lead to a convergent simulation run for different starting values. An extension of our algorithm which takes the solver configuration and possible restarts of the simulation into account could serve as a promising point of origin for further studies.

As mentioned in the introduction, the proposed algorithm assumes that at least one convergent point is known initially and the explored convergent domain is the domain which converges given this initial guess. However, the convergent domain can possibly be enlarged by using not one but several initial guesses from the already obtained database of convergent design points (for example, the n nearest neighbors of the currently evaluated design point). The downside of this approach is that the time for "true" divergent points (i.e., design points which are divergent independent of the choice of the starting values) increases and there is a certain influence of the order in which points are found. However, such a "multi-start" approach could serve as the basis for an improved feasibility exploration strategy.

Another possible extension of the proposed method consists in the consideration of dynamic processes. In this case, the feasible region becomes a function of time and the constant design variables are complemented by time-dependent recipes. We expect that various modifications of our algorithm might be necessary to apply it to such a dynamic scenario, but consider it a valuable research direction.

Finally, there is an alternative version of our proposed algorithm, which is briefly outlined in Appendix B (cf. ESM). The main idea is to directly include the inequality constraints in the optimization problem of the utility function instead of introducing a new utility measure for feasibility. It remains an open question in which scenario or use case one of the two versions of our algorithm is superior to the other.

Acknowledgements This work was developed within the Fraunhofer Cluster of Excellence "Cognitive Internet Technologies".

Electronic Supplementary Material Supplementary material is available in the online version of this article at https://dx.doi.org/10.1007/s11705-021-2073-7 and is accessible for authorized users.

References

1. Grossmann I E, Sargent R W H. Optimum design of chemical plants with uncertain parameters. AIChE Journal. American Institute of Chemical Engineers, 1978, 24(6): 1021–1028

2. Halemane K P, Grossmann I E. Optimal process design under uncertainty. AIChE Journal. American Institute of Chemical Engineers, 1983, 29(3): 425–433
3. Boukouvala F, Ierapetritou M G. Feasibility analysis of black-box processes using an adaptive sampling Kriging-based method. Computers & Chemical Engineering, 2012, 36: 358–368
4. Boukouvala F, Ierapetritou M G. Derivative-free optimization for expensive constrained problems using a novel expected improvement objective function. AIChE Journal. American Institute of Chemical Engineers, 2014, 60(7): 2462–2474
5. Wang Z, Ierapetritou M G. A novel feasibility analysis method for black-box processes using a radial basis function adaptive sampling approach. AIChE Journal. American Institute of Chemical Engineers, 2017, 63(2): 532–550
6. Rogers A, Ierapetritou M G. Feasibility and flexibility analysis of black-box processes Part 1: surrogate-based feasibility analysis. Chemical Engineering Science, 2015, 137: 986–1004
7. Shahriari B, Swersky K, Wang Z, Adams R P, de Freitas N. Taking the human out of the loop: a review of Bayesian optimization. Proceedings of the Institute of Electrical and Electronics Engineers, 2016, 104(1): 148–175
8. Bano G, Wang Z, Facco P, Bezzo F, Barolo M, Ierapetritou M G. A novel and systematic approach to identify the design space of pharmaceutical processes. Computers & Chemical Engineering, 2018, 115: 309–322
9. Gramacy R B, Lee H K H. Optimization Under Unknown Constraints, Bayesian Statistics 9: Proceedings of the Ninth Valencia International Meeting, 2011, 9, 229–256
10. Tran A, Sun J, Furlan J M, Pagalthivarthi K V, Visintainer R J, Wang Y. A batch parallel known/unknown constrained Bayesian optimization with feasibility classification and its applications in computational fluid dynamics. Computer Methods in Applied Mechanics and Engineering, 2019, 347: 827–852
11. Gelbart M A, Snoek J, Adams R P. Bayesian optimization with unknown constraints. arXiv:1403.5607, 2014
12. Griffiths R, Hernández-Lobato J M. Constrained Bayesian optimization for automatic chemical design using variational autoencoders. Chemical Science (Cambridge), 2020, 11(2): 577–586
13. Dias L S, Ierapretrou M G. Data-driven feasibility analysis for the integration of planning and scheduling problems. Optimization and Engineering, 2019, 20(4): 1029–1066
14. Heese R, Walczak M, Seidel T, Asprion N, Bortz M. Optimized data exploration applied to the simulation of a chemical process. Computers & Chemical Engineering, 2019, 124: 326–342
15. Schonlau M, Welch W J, Jones D R. Global versus local search in constrained optimization of computer models. Institute of Mathematical Statistics Lecture Notes—Monograph Series, 1998, 34: 11–25
16. Gelbart M A. Constrained Bayesian optimization and applications. Dissertation for the Doctoral Degree. Cambridge (Massachusetts): Harvard University, 2015
17. Rasmussen C E, Williams C K I. Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). Cambridge (Massachusetts): The MIT Press, 2005
18. Gardner J R, Kusner M J, Xu Z, Weinberger K Q, Cunningham J P. Bayesian optimization with inequality constraints. ICML'14: Proceedings of the 31st International Conference on International Conference on Machine Learning, 2014, 32: 937–945
19. Schölkopf B. The kernel trick for distances. In: Advances in Neural Information Processing Systems. Cambridge (Massachusetts): The MIT Press, 2001, 301–307
20. Heese R, Walczak M, Bortz M, Schmid J. Calibrated simplex mapping classification. arxiv.org/abs/2103.02926, 2021
21. Byrd R H, Lu P, Nocedal J, Zhu C. A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing, 1995, 16(5): 1190–1208
22. Zhu C, Byrd R H, Lu P, Nocedal J. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software, 1997, 23(4): 550–560
23. Virtanen P, Gommers R, Oliphant T E, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods, 2020, 17(3): 261–272
24. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. Scikit-learn: machine learning in Python. Journal of Machine Learning Research, 2011, 12: 2825–2830
25. GPy. GPy: a gaussian process framework in python. The website of github, 2012
26. Biegler L T, Grossmann I E, Westerberg A W. Systematic Methods for Chemical Process Design. New Jersey: Prentice Hall, 1997
27. Renon H, Prausnitz J M. Local compositions in thermodynamic excess functions for liquid mixtures. AIChE Journal. American Institute of Chemical Engineers, 1968, 14(1): 135–144
28. Bortz M, Burger J, Asprion N, Blagov S, Böttcher R, Nowak U, Scheithauer A, Welke R, Küfer K H, Hasse H. Multi-criteria optimization in chemical process design and decision support by navigation on pareto sets. Computers & Chemical Engineering, 2014, 60: 354–363