2018 Annual American Control Conference (ACC)
June 27–29, 2018. Wisconsin Center, Milwaukee, USA

Learning constitutive equations of physical components with constraints discovery

Ion Matei, Johan de Kleer and Raj Minhas

Abstract— **We address the problem of learning constitutive equations of acausal physical components in partially known physical systems. The parameters of the constitutive equations satisfy a set of unknown constraints. We propose an iterative procedure for joint parameters and constraints learning and discuss practical aspects of its implementation. The procedure favors exploration during the first iterations. This enables learning a model for the constraints. As the constraints learning advances more weight is given to finding the constitutive equations. We test our method on a demonstrative example in which the model of a nonlinear resistor is learned.**

I. Introduction

We are interested in learning models of physical components in partially known systems. Such systems are typically acausal. For a system with continuous states only, its behavior can be represented as a differential algebraic equation (DAE) of the form

$$0 = \mathbf{F}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{z}; w) \quad (1)$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{z}; w), \quad (2)$$

where $\mathbf{F}$, $\mathbf{h}$ are vector valued continuous maps, $\mathbf{x}$ is the state vector, $\mathbf{z}$ is the vector of algebraic variables, $\mathbf{y}$ is the vector of outputs, and w is the vector of parameters. An acausal system is composed of variables attached to its components and relations between them. The relations are induced by the parameterized *constitutive equations* and the connections between components. The parameters of the constitutive equations are usually constrained within some feasibility set. For example, we know that the resistance of an electric resistor must always be positive.

Our approach is based on the following steps: (I) we select a parametric mathematical model for the constitutive equations; (II) we learn the parameters and a representation/model of the parameter constraints. For the latter, we use an explore-exploit strategy, where in the beginning we focus on exploration to learn the constraints representation. Later, as the constraints representation becomes more informative, the focus is shifted towards finding the best feasible parameters of the constitutive equations. Having a representation for parameter constraints serves to: decrease the complexity of the search; provide a good initial condition when new data is available for refining the component's parametric model and the constraints representation; enable a physical interpretation of the component model. Recent results in learning physical laws describing system/component behavior were fueled by the reignited interest in machine learning and its derivative, deep learning. In [5] the authors used a neural network architecture for learning predictive models of intuitive physics from images. A sparse regression method was used in [4], [16] to discover the (partial) differential equations describing the governing physical laws. A rather unscalable method based on a genetic algorithm to distill the free-form laws from measurements was proposed in [17].

Ion Matei, Johan de Kleer and Raj Minhas are with the Palo Alto Research Center (PARC), Palo Alto, CA (emails: ion.matei@parc.com, dekleer@parc.com, raj.minhas@parc.com)

Our working assumptions are: (i) the physical system is described by interconnections of physical components, (ii) the constitutive equations of the system components are known for a subset of the components only; (iii) observations of the system behavior are available. We do not necessarily have direct access to the component variable. Having a component based description of a physical system enables a range of model-based analytics methods for controls [9], diagnosis [1], [7] and prognosis [10]. The working assumption (ii) is more frequent than one may imagine. Often system manufacturers would like to improve their offerings by including (model-based) diagnosis and prognostics features in their products. However, they do not always have access to detailed specifications of the sub-components, when such sub-components are acquired from third-parties, due to proprietary reasons. The authors have faced several such cases in recent industry-related projects. In one such engagement [12], although we had the structural description of a rail switch, and part of the electrical specifications, we had challenges in finding out the specifications for some of the mechanical parts. The work described in this paper provides a solution to such a problem.

In the fortunate case where we have qualitative information about the behavior of the system, that is, we know the constitutive equations but not the parameter values, the learning problem becomes a standard parameter estimation problem, which has a long history [2]. In our scenario such information is not available. Hence we must propose an acausal model for the unknown component such that at minimum, when included in the system model, the DAE (1) is solvable. We call such models *feasible*. A higher goal is *reusability* of the component model. This means that when the unknown physical component is used in a different configuration, the real and simulated behaviors must be similar. Learning a reusable component however requires experimental data corresponding to a variety use cases for the component. In this paper we focus mainly on feasibility. From a numerical point of view, feasibility translates to solving the DAE (1) and determining the trajectory of $\mathbf{x}(t)$ and $\mathbf{z}(t)$ over some time interval and for some

978-1-5386-5428-6/$31.00 ©2018 AACC

initial conditions. This process requires the Jacobian $\frac{\partial \mathbf{F}}{\partial \boldsymbol{\eta}}(\boldsymbol{\eta}(t))$, with $\boldsymbol{\eta}^T = [\dot{\mathbf{x}}^T, \mathbf{z}^T]$ to be invertible along the trajectory of the system. If this property fails at some point along the trajectory, the numerical simulation will fail. We have the following definition of a feasible component model.

Definition 1.1 (Component feasibility): We say a component model is feasible if, when used in a system model, the Jacobian $\frac{\partial \mathbf{F}}{\partial \boldsymbol{\eta}}(\boldsymbol{\eta}(t))$ is invertible along solution path.

In what follows we assume that the unknown constitutive equations of a component are represented by a system of equations $\boldsymbol{G}(\boldsymbol{\xi}; w) = 0$, where $\boldsymbol{G}$ is a differentiable vector valued function, $\boldsymbol{\xi}$ is the vector of variables of the component and w is the vector of unknown parameters. To learn w, we assume we have access to a set of measurements $\mathbf{y}_m$ over some time interval, that give indirect information about the behavior of the component. These measurements are used to solve a nonlinear least-square problem with constraints:

$$\min_{w} \quad \mathbf{J}(w) \tag{3}$$

$$\text{subject to}: \quad w \in \mathcal{W} \subset \mathbb{R}^n, \tag{4}$$

where $\mathbf{J}(w) = \frac{1}{N}\sum_{i=0}^{N} \|\mathbf{y}_m(t_i) - \mathbf{y}(t_i; w)\|^2$, $\mathbf{y}(t_i; w)$ are the predicted measurements based on the current value of w, $\{t_i\}_{i\geq 0}$ represents a sequence of instances at which measurements are taken, and $\mathcal{W}$ is the *unknown* feasibility set. Our objective is to come up with a method that can jointly learn the optimal w and at least a local representation of the set $\mathcal{W}$.

II. Models for physical components

When modeling physical components we need to define a domain specific interface through which energy is exchanged. The interface is modeled through *connectors* that regardless of the physical domain have at least one flow variable and a potential like variable (in the thermo-fluid domain there may be several such variables). In the electrical domain the current is the flow variable and the electric potential is the across variable.

In this paper we consider two pins (connectors) components, as shown in Figure 1. Each connector has two variables: a flow variable f and a potential like variable x. Depending on the physical domain, the flow variable can be current (electric), force (translational mechanics), torque (rotational mechanics) or heat flow rate (thermal). The variable x can represent electric potential (electric), displacement (translational mechanics), angle (rotational mechanics) or temperature (thermal). The constitutive equations of the model are a set of equations that describe the relations between the pairs of variables (f_a, x_a) and (f_b, x_b). In the case of a memoryless component, we have $\boldsymbol{G}(f_a, f_b, x_a, x_b; w) = 0$, where $\boldsymbol{G} : \mathbb{R}^4 \to \mathbb{R}^2$ is a vector valued differentiable map that constrains the connector variables, and w is a vector of parameters. For example, in the case of the electrical resistor we have $f_a + f_b = 0$ and $wf_a = x_a - x_b$. Note that $\boldsymbol{G}$ defines only 2 equations for the 4 variables. When connected to other components, additional equations are generated from the constitutive equations of the other components and from the connection equations. In the simplest scenario, 2 of the 4 variables are independently computed from other equations. This is the case where one connector is connected to the ground (zero potential) and the remaining one is connected to a current or voltage source. In more complicated scenarios, the behavior of the system is described by a set of algebraic equations, where the variables of all components are computed simultaneously. We can model components with memory as well by including derivatives of the component's variables: $\boldsymbol{G}(\boldsymbol{\xi}; w) = 0$, where $\boldsymbol{\xi}^T = [f_a, f_a^{(1)}, \ldots, f_a^{(m)}, f_b, f_b^{(1)}, \ldots, f_b^{(m)}, x_a, x_a^{(1)}, \ldots, x_a^{(n)}, x_b, \ldots, x_b^{(n)}]$, with $x^{(n)}$ the n^{th} derivative of x. It may appear that we have $2\times(n+1)+2\times(m+1)$ variables and only two equations ($\boldsymbol{G}$ remains a two dimensional vector valued function). For these type of components however, only the higher order derivatives are unknown variables, while the lower order derivatives are assumed known from the previous integration step.

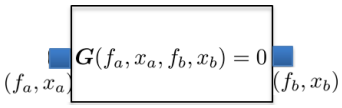


Fig. 1: Template for memoryless component

Without any prior information about the component, choosing the structure for $\boldsymbol{G}$ is more of an art than science. The simplest form can be a linear regression model $\boldsymbol{G}(\boldsymbol{\xi}; w) = \mathbf{W}\boldsymbol{\xi}_{ext}$ where $\mathbf{W}$ is a matrix of parameters, and $\boldsymbol{\xi}_{ext}$ is a vector of "features" whose elements are powers of $\boldsymbol{\xi}$' entries. For example, in the memoryless case we can have $\boldsymbol{\xi}_{ext} = [f_a, f_a^2, f_a^3, f_b, f_b^2, f_b^3, x_a, x_a^2, x_a^3, x_b, x_b^2, x_b^3]$. More complex behaviors can be achieved by considering also multiplications of powers of $\boldsymbol{\xi}$' entries, as proposed in [16]. Recalling that neural networks (NNs) are universal approximators [6], we can choose $\boldsymbol{G}$ to be described by a NN, where the input is the vector $\boldsymbol{\xi}$ and the outputs must follow targets that are the zero constants. Note however, that we would regard a NN as a bidirectional graph. That is, it represents a set of equations and not a set of assignments.

III. Joint learning and constraint discovery

Assume we have chosen a parametric model for $\boldsymbol{G}$ and would like to learn its parameters to best match the experimental data. In the beginning, no information about the feasibility set is available. Hence we need to gradually acquire data, optimize the parameters of $\boldsymbol{G}$ and build a model for the feasibility set. We propose a sequential scheme under which in the beginning we encourage exploration of points with high uncertainty, while trying to stay close to parameter values that decrease the cost function $\mathbf{J}(w)$. In what follows we assume that $\mathbf{J}(w)$ has a least one (local) minimizer. The scheme is enabled by the following result.

Proposition 3.1: Let $\{g_k\}_{k\geq 0}$, with $g_{k,i} : \mathbb{R}^n \to [a_i, b_i]$, $i = 1, 2$ be two convergent sequences of continuously differentiable scalar functions, with $\lim_{k\to\infty} g_{k,i} = g_i$. They induce the sequence of optimization problems

$$(\mathrm{P}_k): \quad \min_w \quad \mathbf{J}(w) \tag{5}$$

$$\text{subject to:} \quad g_{k,i}(w) \leq 0, \ i = 1, 2, \tag{6}$$

where each problem (P_k) admits at least a local minimizer w_k^* as solution. Then the limit points w^* of the sequence $\{w_k^*\}_{k\geq 0}$

are solutions of the problem

$$\begin{aligned} (\mathrm{P}): \quad & \min_w \quad \mathbf{J}(w) \\ & \text{subject to:} \quad g_i(w) \le 0,\ i = 1,2. \end{aligned}$$

The proof follows from a continuity argument. In our setup, $\mathbf{J}$ is the cost function introduced in (3) and $g_i(w)$ are constraint functions related to the feasibility set. They evolve with each iteration k, as our understanding of the feasibility set of w improves.

We model the feasibility of a point w by a binary function $p(w) \in \{0,1\}$, where $p(w) = 1$ if w is feasible and zero otherwise. If we could initially probe the feasibility of every possible point, we could build the map $p(\cdot)$ which would describe the feasibility space. Obviously, this is a hopeless scenario. The best we can hope for is to have a significantly large data set of points for which the feasibility was tested.

Let $S \subset \mathbb{R}^n$ be an (arbitrarily large) compact set. We want to find an approximation of $p(w)$ modeled as a smooth, parameterized function $q : S \to [0,1]$, with $q(w) = q(w;\beta)$. This approximation is obtained by minimizing the cross-entropy loss function $H(\beta;S) = -\int_S [p(w)\log_2 q(w;\beta) + (1-p(w))\log_2(1-q(w;\beta))]\,dw$. We denote by $q^*(w) = q(w;\beta^*)$ the best approximation of p, where $\beta^* = \arg\min_\beta H(\beta;S)$, and $H^*(S) = H(\beta^*;S)$. The function q^* represents a goal function in a ideal scenario where we can probe the feasibility of any point w in the set S. The following result shows how we can arrive at q^* through successive learning of approximations of p, as we keep probing the set S.

Proposition 3.2: Let $\{S_k\}_{k\ge 0}$ be an increasing sequence of sets of discrete points, such that $\cup_{k\ge 0} S_k = S$. Let $\{q_k(w)\}_{k\ge 0}$ be a sequence of approximations of $p(w)$, where $q_k(w) = q(w;\beta_k)$, $\beta_k = \arg\min_\beta \tilde{H}(\beta;S_k)$, with $\tilde{H}(\beta;S_k) = \frac{1}{|S_k|}\sum_{i=1}^{|S_k|}[p(w_i)\log_2 q(w_i;\beta) + (1-p(w_i))\log_2(1-q(w_i;\beta))]$. Then $\lim_{k\to\infty}\tilde{H}(\beta_k;S_k) = H^*(S)$, and $\lim_{k\to\infty} q_k = q^*$.

The sets S_k contain samples from the set S. These samples enable the approximation of the entropy $H(\beta;S)$ through the quantity $\tilde{H}(\beta;S_k)$, approximation that becomes more accurate as S_k approach S. By making S arbitrarily large, we can cover the entire $\mathbb{R}^n$. A possible choice for q is the softmax function $q(w;\beta) = e^{\beta^T w}/[1+e^{\beta^T w}]$. We can have more complex representations such as NNs where the last layer is the softmax function. Learning the parameters β_k is equivalent to training a binary classifier, using a cross-entropy loss function.

We are now ready to define the functions $g_{k,i}$. In particular we have

$$g_{k,1}(w) = 1 - \delta_{k,1} - q_k(w) \tag{7}$$

$$g_{k,2}(w) = 1 - \delta_{k,2} + q_k(w)\log_2 q_k(w) + [1-q_k(w)]\log_2[1-q_k(w)], \tag{8}$$

where $\{\delta_{k,1}\}_{k\ge 0}$ is a positive sequence with $\lim_{k\to\infty}\delta_{k,1} = \varepsilon$, $0 < \varepsilon \ll 1$ and $\delta_{0,1} = 1$, and $\{\delta_{k,2}\}_{k\ge 0}$ is an increasing sequence of positive values such that $\lim_{k\to\infty}\delta_{k,2} = 1+\varepsilon$, with $\delta_{0,2} = \varepsilon$. The function $q_k(w)$ was introduced in Proposition 3.2. The second constraint (8) uses an entropy like term, whose purpose is to encourage exploration of points with higher uncertainty. This term loses importance as we improve the model of q_k. At the limit, the constraint (7) becomes $q^*(w) \ge 1-\varepsilon$, while (8) becomes $q^*(w)\log q^*(w) + [1-q^*(w)]\log[1-q^*(w)] \le \varepsilon$, which is satisfied for any w. Hence, this constraint disappears in the limit. This ensures that the solution of the optimization problem is feasible with high likelihood. At the first iteration, when no data is available, we make sure that $q_0(w) = 0.5$ for all w. This means that all points have maximum uncertainty. Since the entropy at this value is 1, it also ensures that (P_0) has a solution. In the case of the softmax function, this happens if we enforce $\beta_0 = 0$. Therefore, at the initial iteration, we solve an unconstrained optimization problem. At the second iteration, $q_1(w)$ is a smooth function that passes through the point 0.5, hence there will exist w such that the constraint (8) is satisfied. The scalar ε accounts for the fact that $q^*(w)$ is in the end an approximation of $p(w)$.

So far we have not addressed the procedure of accumulating data for training the intermediate feasibility models $q_k(w)$. At each iteration k, the optimization algorithm generates data points in its search for the minimizer. The search requires function evaluations, which in turn require the simulation of the model. Hence each point tested while searching for a better point at iteration k is labeled as feasible or infeasible, depending on the success of the simulation. In the beginning of the iterative process, the second constraint (8) pushes the search towards points with high uncertainty, helping this way explore points that were not visited before. This strategy however cannot guarantee that when we stop iterating after k steps, the data-set S_k is sufficiently large and informative so that the resulting feasibility model $q_k(w)$ closely matches the entire landscape defined by $p(w)$. Still, it can provide a good local representation of the feasibility set along the trajectory generated as part of the optimization process. We summarize the approach described so far in Algorithm 1. Line 6 solves the constrained optimization problem (5)-(6) and returns the solution and all points visited plus their labels during the solution search. Since it used a previous model of function q, can be interpreted as containing also a prediction step, since the feasibility of any visited point is predicted by q. In line 8, the feasibility model is retrained based on the newly acquired data. If the data set $\mathcal{D}$ becomes significantly large, we can use online learning. For example, we can execute a batch learning update step, where the gradient that dictates the update direction is computed using the newly acquired data only.

Algorithm 1 Learning with constraints dicovery

```
1: procedure OptWithConstrDiscovery(w_0)
2:     w ← w_0
3:     D ← {∅}
4:     q(w) ← 0.5, ∀w
5:     while solution improvement do
6:         (w,data)← compute solution of (5)-(6) given w,q
7:         D ← D ⋃ data
8:         q ← retrain feasibility model based on D
9:     return w,q
```

Remark 3.1: Before starting the actual optimization process, we can sample from the component parameter space, execute a simulation for each sample, and use the data for pre-training the probability model $q(w;\beta)$. Although feasible, this approach has the potential to be both time and computational expensive, especially when the system model is complex. A cheaper alternative is to generate much simpler models that include the component we want to learn in different configurations. Such configurations will describe different scenarios for computing the unknown variables of the component's constitutive equations described by $\boldsymbol{G}(\boldsymbol{\xi};w)=0$, shedding light this way on at least part of the feasible space. That is, they will generate different structures for the Jacobian matrix of the model that need to be invertible for the simulations to succeed.

IV. Practical implementation aspects

Both the search for the best parameters w and the constraints discovery depend on the success of the model simulation for the parameter values that are tested during the optimization procedure. When a simulation fails, we are unable to compute the cost function $\mathbf{J}(w)$. Still, we need to let the optimization algorithm know that it hit an infeasible point and encourage it to leave that search region. In an ideal scenario where $p(w)$ is known, an equivalent cost function is $\tilde{\mathbf{J}}(w)=p(w)\mathbf{J}(w)-\log[p(w)+\mu]$, where $\mu \ll 1$ is a very small positive value. This cost function however is not differentiable. Based on the points explored so far, we approximate $p(w)$ by a smooth pseudo-approximation $\tilde{p}(w)$. Let $\mathcal{U}=\{w_1,w_2,\ldots,w_m\}$ be the infeasible points discovered so far. We approximate $p(w)$ by

$$\tilde{p}(w)=1-\sum_{i=1}^{m} e^{\frac{\|w-w_i\|^2}{2\sigma^2}}+\mu \tag{9}$$

where σ is small enough so that no two Gaussian functions overlap significantly, and μ is a very small scalar that makes sure $\tilde{p}(w)$ does not become negative for infeasible points.We use $\tilde{p}(w)$ in the definition of the equivalent cost function: $\tilde{\mathbf{J}}(w)=\tilde{p}(w)\mathbf{J}(w)-\log[\tilde{p}(w)+\mu]$. Under this formulation, for an unsuccessful simulation we can reasonably approximate $\tilde{\mathbf{J}}(w)$ by $-\log[\tilde{p}(w)+\mu]$. This idea is detailed in Algorithm 2. Note that unlike $q(w)$, $\tilde{p}(w)$ does not require learning and does not generalize well to points that were not visited yet. Using $q(w)$ instead of $\tilde{p}(w)$ is not a good idea since at least in the beginning $q(w)$ is not guaranteed to be (very close to) zero at an infeasible point. The intermediate optimization problems (5)-(6) are optimization problems with inequality constraints. There are several methods available to solve such problems: penalty methods, barrier methods, primal methods, or sequential quadratic programming (SQP) Among many technical conditions, many of them require twice continuous differentiability of the cost function. For a complex system, even computing the gradient can be challenging. The gradient of loss function $\mathbf{J}$ is given by $\nabla\mathbf{J}(w)=-\frac{1}{N}\sum_{i=0}^{N}\left(\mathbf{y}_m(t_i)-\mathbf{y}(t_i)\right)\frac{\partial\mathbf{y}(t_i)}{\partial w}$, where $\frac{\partial\mathbf{y}}{\partial w}=\frac{\partial\mathbf{h}}{\partial\mathbf{x}}\frac{\partial\mathbf{x}}{\partial w}+\frac{\partial\mathbf{h}}{\partial\mathbf{z}}\frac{\partial\mathbf{z}}{\partial w}+\frac{\partial\mathbf{h}}{\partial w}$, as per equation (2). Even in a simple scenario where we can symbolically compute $\frac{\partial\mathbf{h}}{\partial\mathbf{x}}$ and $\frac{\partial\mathbf{h}}{\partial\mathbf{z}}$ (e.g., $\mathbf{h}$ is linear in $\mathbf{x}$ and $\mathbf{z}$), and does not depend on w, we still need $\frac{\partial\mathbf{x}}{\partial w}$ and $\frac{\partial\mathbf{z}}{\partial w}$. Sensitivity analysis capabilities that use forward and backwards methods are already integrated in DAE solvers [14] so in theory, we could extract $\frac{\partial\mathbf{x}}{\partial w}$ and $\frac{\partial\mathbf{z}}{\partial w}$ from their results. Still obtaining the Hessian is by far not a trivial matter. Luckily it can be iteratively approximated using the gradient [13]. When the gradient is not explicitly available, optimization algorithms resort to numerical approximations. If we do not trust these approximations, another alternative is to resort to gradient free methods such as simplex search based on Nelder-Mead algorithm, pattern search, particle swarm optimization, genetic algorithms evolution strategy or Powell's algorithm with its different versions. Many of the aforementioned algorithms however accept at most box type of constraints.

Algorithm 2 Cost function

```
1: Unfeasible set U = {∅}
2: function costFunction(w)
3:     Simulate model for parameters w
4:     if simulation fails then
5:         Update the set of unfeasible points U ← U ⋃ {w}
6:         Compute p̃(w) according to (9)
7:         Compute the cost function J̃(w) ← −log p̃(w)
8:     else
9:         Compute p̃(w) according to (9)
10:        Compute the cost function J̃(w) ← J̃(w) = p̃(w)J(w) −
       log p̃(w)
11:    return J̃(w)
```

Hence we need to transform (5)-(6) into unconstrained optimization problems. We implement this using the barrier function idea. Namely, we replace the cost function $\tilde{\mathbf{J}}(w)$ in (P_k) by $\mathbf{J}_k(w)=\tilde{\mathbf{J}}(w)-\lambda_{k,1}\log q_k(w)+\lambda_{k,2}\left[q_k(w)\log q_k(w)+(1-q_k(w))\log(1-q_k(w))\right]$, where $\{\lambda_{k,i}\}_{k\geq 0}$, $i=1,2$ are sequences of decreasing positive scalars. The new cost function represents a tradeoff between minimizing $\mathbf{J}(w)$, choosing points with high feasibility probability and exploring points with high uncertainty. The new sequence of optimization problems (P_k) is a sequence of unconstrained optimization problems for which gradient-free methods can be used.

If sub-optimal solutions are acceptable, we can eliminate the iterative procedure, and solve a one shot optimization problem with cost function: $\mathbf{J}'(w)=\tilde{\mathbf{J}}(w)-\lambda_1\log q(w)+\lambda_2\left[q(w)\log q(w)+(1-q(w))\log(1-q(w))\right]$, where $q(w)$ is the feasibility model updated every time a model simulation is executed. This heuristic has a good chance to provide satisfactory results since, as we explore more and more points, the entropy term becomes roughly zero. Moreover, if the search is performed in a feasible region, the second term becomes zero. Hence, the search is mainly focused on minimizing $\mathbf{J}$. We only need to modify the way the cost function is computed, and we can use any unconstrained optimization algorithm. The computations inside the cost function are summarized in Algorithm 3.

In line 10 of Algorithm 3, rather than re-training the parameters β from scratch, we can instead use the on-line gradient descent algorithm, where the gradient of $\tilde{H}(\beta;\mathcal{D})$

Algorithm 3 Cost function for the constraint discovery heuristic

```
D: current training set for q(w;β)
function costFunctionHeuristics(w)
    Compute J̃(w) as in Algorithm 2
    if simulation failed then
        y ← 0
    else
        y ← 1
    Compute the cost function J′(w) ← J̃(w) − λ1 log q(w) + λ2 [q(w) log q(w) + (1 − q(w)) log(1 − q(w))]
    Augment the training data set: D ← D ⋃ {(w, y)}
    Update the parameters of the feasibility probability q(w;β) based on the augmented set D
    return J′(w)
```

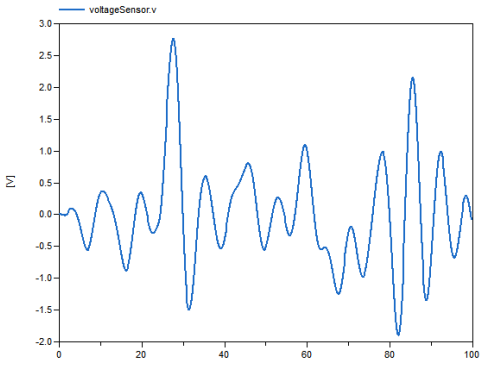


Fig. 3: Experimental data: resistor R4 voltage

defined in Proposition 3.2 is approximated by the gradient of a single example $\beta \leftarrow \beta - \alpha \nabla h(\beta;(w,y))$, where $h(\beta;(w,y)) = -y \log q(w;\beta) - (1-y)\log[1-q(w;\beta)]$. During every call of the cost function either to approximate the gradient or to test directions of search in derivative free algorithms, the feasibility probability is updated.

V. Demonstrative example

The goal is to learn the model of the nonlinear resistor `R_nonlin` in the Cauer filter shown in Figure 2. The output measurement is the voltage across the resistor `R4` over a time interval of 100s. The voltage provides indirect information about the nonlinear resistor behavior. The circuit is powered by a voltage source whose input is a band-limited white noise with normal distribution signal.

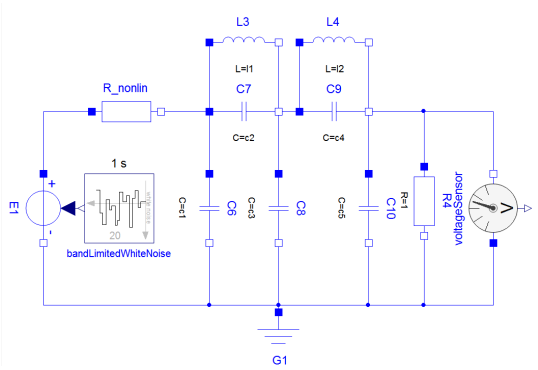


Fig. 2: Cauer filter with nonlinear resistor

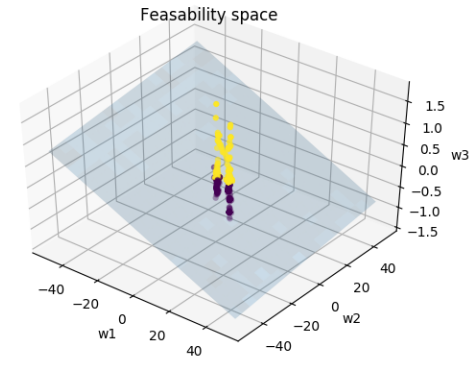
The "experimental" measurements are shown in Figure 3. The "real" real behavior of the nonlinear resistor is described by $i = 0.01 \cdot v^3$, where i and v represent the current and voltage of the resistor, respectively. In the learning process, we model the constitutive equations by the polynomial $w_1 i = w_2 v^2 + w_3 v^3$. In addition to the parameter values, we would like to learn a representation of the the feasibility set corresponding to w. We chose a three dimensional vector of parameters to have a descriptive 3D representation of the feasible sets.

The probability of a feasible is modeled using the softmax function $q(w;\beta) = e^{\beta^T w}/(1+e^{\beta^T w})$. This representation will provide us with the hyperplane separating the feasible and infeasible subspaces. Note that we are making an assumption that indeed such a separation is possible. That may not be the case. For more complicated representations, a NN can precede the softmax function. The separating hyperplane is given by $\beta_1 w_1 + \beta_2 w_2 + \beta_3 w_3 = 0$. Therefore w is feasible provided $\beta^T w > 0$.

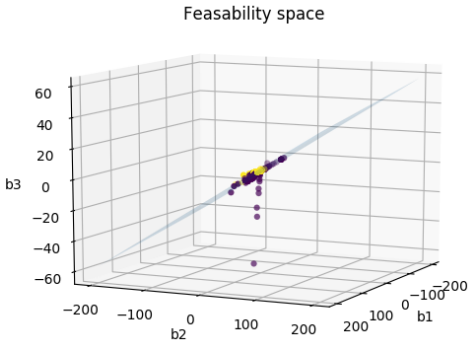
The learning procedure was implemented in Python by using algorithms from the optimization package in Python's SciPy library. The evaluation of the optimization cost function is based on simulations of a Functional Mockup Unit (FMU) [3] generated from the Modelica [8] description of the electrical circuit model.

We first implemented Algorithm 1. The inner loop was implemented using two algorithms that accept constraints: COBYLA [15], a derivative-free algorithm for constrained optimization, and sequential least square programming (SLSQP) [11]. We executed 25 outer iterations, and for each internal iteration we limited the maximum number of iterations to 300. The relative error in the minimizer and minimum value were set to `xtol=ftol`=1e-16. The sequences $\delta_{k,1}$ and $\delta_{k,2}$ were chosen as $\delta_{k,1} = \varepsilon + (1-\varepsilon)(\lambda+\varepsilon)^k$ and $\delta_{k,2} = 1 + \varepsilon - \lambda^k$, with ε =1e-7 and $\lambda = 0.8$. The step-size for the gradient approximation was set to `eps`=1e-04. The data accumulated using COBYLA shows a clear separation between feasible and infeasible points (see Figure 4) The SLSQP explored mostly points around the separating hyperplane.

We tested Algorithm 3 on both gradient free algorithms and algorithms based on gradient approximations. Comparing the two classes of algorithms is not easy due to incompatible hyper-parameters. We limited the maximum number of iterations and the maximum number of function evaluations to 1000 and 1500, respectively. The hyper-parameters of Algorithm 3 were chosen as $\lambda_1 = 1$ and $\lambda_2 = 0.1$. We use the same initial condition for all algorithms, namely $w = [0.1, 0.1, 0.1]$. The relative error in the minimizer and minimum value were set to `xtol=ftol`=1e-16. Since the solution w^* is invariant to scalar multiplication, we can divide the solution by the first component $w^* \leftarrow \frac{1}{w_1} w^*$. Using this observation we could have reduced the dimension of the



(a) COBYLA



(b) SLSQP

Fig. 4: Separating hyperplane generated by Algorithm 1

parameter vector to two, by forcing $w_1 = 1$. We did not do it since we have empirically observed that the optimization algorithms behave much better with a three dimensional vector of parameters. We are interested in w_2^* and w_3^*. In particular, for all three algorithms w_3^* gets close to its real value.

Next we used algorithms that use gradient approximations: (limited memory) Broyden-Fletcher-Goldfarb-Shanno (L-BFGS), and nonlinear conjugate gradient algorithm (CG). In addition to the settings used in the case of gradient-free algorithms, we set the step size for the gradient approximation to `eps`=1e-4. The results of the optimization procedures are summarized in Table I.

TABLE I: Parameter learning results

Algorithm	**Solution** w^*	**Average error**
COBYLA	[1,-9.31240083e-04,9.22597610e-03]	0.0028635
SLSQP	[1, -0.00170486, 0.00951815]	0.00273461
Powell	[1,-2.20299821e-04,9.85091226e-03]	0.00274897
Nelder	[1,-2.37604458e-04,9.75878109e-03]	0.00253461
BFGS	[1,8.28670889e-05,9.76706821e-03]	0.00276546
CG	[1,-0.00137244,0.00865319]	0.00464650
LBFGS	[1,-6.96951339e-04,9.79259844e-03]	0.00254597

VI. Conclusions

We addressed the problem of jointly learning parameters of acausal components and models for their feasibility constraints. We proposed an iterative procedure based on an exploit-explore strategy. In the beginning of the procedure exploration is encourage to accumulate data for learning the model for the feasibility constraints. As the feasibility constraint model becomes more accurate, the weight is shifted to the loss function induced by the parameter estimation problem. The feasibility constraint model depends on the optimization algorithm used since each of them generates different search trajectory. We proposed also a heuristics under which the feasibility model is updated at each iteration of the optimization algorithm. Under this heuristic less data is accumulated and therefore the feasibility models tend to be more localized. The proposed algorithms were used for learning the parameters of a non-linear resistor. In the future we plan to include additional component related constraints such as passivity in the learning process.

References

[1] H. Alwi, C. Edwards, and C.P. Tan. *Fault Detection and Fault-Tolerant Control Using Sliding Modes*. Springer, London, 1974 (ISBN: 0-12-078250-2).

[2] Y. Bard. *Nonlinear Parameter Estimation*. Academic, New York, 2011 (ISBN: 978-0-85729-649-8).

[3] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clau, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. v. Peetz, S. Wolf, Atego Systems Gmbh, Qtronic Berlin, Fraunhofer Scai, and St. Augustin. The functional mockup interface for tool independent exchange of simulation models. In *In Proceedings of the 8th International Modelica Conference*, 2011.

[4] S. Brunton, J.N. Kutz, and J. Proctor. Data-driven discovery of governing physical laws. *SIAM News*, 50(1), 2017.

[5] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[7] Johan de Kleer and Brian C Williams. Diagnosing multiple faults. *Artificial intelligence*, 32(1):97–130, 1987.

[8] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. IEEE Press. Wiley, Hoboken, NJ, 2 edition, 2015.

[9] Carlos E. Garca, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice - A survey. *Automatica*, 25(3):335 – 348, 1989.

[10] Andrew K.S. Jardine, Daming Lin, and Dragan Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20(7):1483 – 1510, 2006.

[11] D. Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988.

[12] I. Matei, A. Ganguli, T. Honda, and J. de Kleer. The case for a hybrid approach to diagnosis: A railway switch. In *Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015)*, pages 225–232, Aug 2015.

[13] Jorge Nocedal. Updating Quasi-Newton Matrices with Limited Storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[14] Linda Petzold, Shengtai Li, Yang Cao, and Radu Serban. Sensitivity analysis of differential-algebraic equations and partial differential equations. *Computers and Chemical Engineering*, 30(10):1553 – 1559, 2006.

[15] M.J.D. Powell. A view of algorithms for optimization without derivatives. Technical report, University of Cambridge, UK, May 2007.

[16] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4), 2017.

[17] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.