2023-01-5047 Published 28 Jul 2023

A General Matlab and COMSOL Co-simulation Framework for Model Parameter Optimization: Lithium-Ion Battery and Gasoline Particulate Filter Case Studies

Gabriele Pozzato and Simona Onori Stanford University, Energy Science and Engineering

Citation: Pozzato, G. and Onori, S., "A General Matlab and COMSOL Co-simulation Framework for Model Parameter Optimization: Lithium-Ion Battery and Gasoline Particulate Filter Case Studies," SAE Technical Paper 2023-01-5047, 2023, doi:10.4271/2023-01-5047.

Received: 23 Jan 2023 Revised: 15 May 2023 Accepted: 23 Jun 2023

Abstract

This paper develops a co-simulation framework based on the use of the package *LiveLink*™ *for Matlab* to perform parameters optimization of dynamical systems implemented in *COMSOL Multiphysics*. The identification problem is recast as an optimization problem which is solved in Matlab. Code for the key steps of the approach is described in detail, and an implementation based on the particle swarm optimization (PSO) algorithm is proposed. The effectiveness and general applicability of the framework are shown for two energy systems: lithium-ion battery (LIB) and gasoline particulate filter (GPF). Matlab codes and COMSOL models for both case studies are made publicly available and can be used as a starting point to solve parameter identification problems for systems beyond the case studies presented here.

Keywords

Matlab/COMSOL co-simulation, Particle swarm optimization, Battery modeling, Gasoline particulate filter

1. Introduction

Physics-based modeling is an integral part of the scientific research used for design optimization [1] and the development of control, management, and estimation strategies [2, 3]. Depending on the complexity and coupling of the spatial and time dynamics under investigation, different length scales—from atomic to system level—can be used to develop models with the desired level of accuracy. Since the dynamical behavior of a system is the outcome of the inherent interactions of different physics, the ability to simulate such phenomena is key for system design and has led to the development of multiphysics simulation tools which are used for system design, testing, and analysis.

COMSOL Multiphysics® is a finite element software platform specifically designed for multiphysics simulations of a wide range of physical phenomena (mechanical, fluid dynamics, chemical, etc.) [4]. An application to fluid dynamics and electromagnetics is shown in [5], where microwave heating of liquids is modeled in COMSOL and compared to other open-source software tools. In [6], energy and momentum transport equations are used to model a plasma chemical reactor for the disposal of waste material. In [7], the flow and hydraulic characteristics of an optimized Chinese dome digester—a domestic biogas plant—are analyzed and compared to a traditional design, showing that the addition of baffles to the geometry can improve mixing in the reactor. In [8], a coupled thermal-hydraulic-mechanical-chemical model is used to simulate the behavior of methane hydrate–bearing sediments during methane gas production. These are some examples from the recent literature showing the wide applicability of COMSOL, where the model development is assisted by a user-friendly interface offering the possibility to choose between built-in and user-defined equations and boundary conditions. Moreover, numerical solvers are highly configurable and ensure fast and stable solutions. An important feature of COMSOL is the *LiveLink*™ *for Matlab*®, which allows the communication between *Matlab*® and COMSOL with regard to the analysis and processing of simulation results, changing model parameters, and monitoring variables during the simulation of the model. This eases the analysis of COMSOL numerical solutions (e.g., for sensitivity studies) and, as shown in this work, model optimization.

A fundamental challenge associated with multiphysics, and in general, dynamical systems simulations, is the problem of parameter calibration or identification [9]. In COMSOL, this problem can be tackled with the built-in optimization module, which allows us to find the optimal shape of a geometry or to identify model parameters [10]. This module, however, is constrained by the use of the gradient-based and gradient-free optimization libraries provided by COMSOL.

Identification problems are generally recast into optimization problems and, depending on their characteristics (e.g., linear/nonlinear, convex/nonconvex, mixed integer), the solution is tackled differently. For example, convex optimization and solvers like Gurobi [11] are used for programs with convex objective functions subject to affine equality constraints and convex inequality constraints. For mixed-integer problems, branch-and-bound algorithms, based on the *divide and conquer* paradigm, are usually employed [12]. However, optimization problems can be characterized by rough and discontinuous optimization surfaces, making the evaluation of derivatives to find the optima difficult. In these cases, gradient-free techniques such as evolutionary algorithms are effective in solving problems with nonlinear objective functions subject to linear or nonlinear static and dynamic constraints. They are also easily implementable and performance does not deteriorate severely with the growth of the search space dimension; for this reason, they are valuable tools for the solution of a broad class of optimization problems [13]. These algorithms provide great flexibility; however, the current version of COMSOL does not provide routines based on such approaches.

Introduced in this paper, the Matlab-COMSOL co-simulation framework is formulated to develop parameter optimization routines based on evolutionary algorithms for a general class of systems described by partial differential equations (PDEs).[1] Within this framework, the multiphysics system is implemented in COMSOL, where geometry (1D, 2D, or 3D), governing equations, mesh, and numerical solver settings are defined. On the other hand, Matlab is used in a co-simulation environment to perform parameter optimization by minimizing a user-defined objective function. The framework is developed for particle swarm optimization (PSO), but it can be adapted to other optimization techniques such as genetic algorithms (GA) and differential evolution (DE) algorithms. The identification process is described theoretically, and codes for the key steps of the parameter optimization procedure are analyzed. The general applicability and effectiveness of the proposed framework is shown in two case studies: a lithium-ion battery (LIB) and a gasoline particulate filter (GPF).

LIB are energy storage devices used in today's portable electronics, hybrid and electric vehicles, power tools, etc. In this work, COMSOL is used to implement the Doyle-Fuller-Newman (DFN) battery model [14], which considers charge and mass transport dynamics in the electrode (solid) and electrolyte (liquid) phases to describe the motion of lithium ions and their intercalation/deintercalation. Geometrical (e.g., positive electrode, negative electrode, and separator thicknesses), stoichiometric, and transport parameters (e.g., diffusion coefficients) are identified minimizing the discrepancy between experimental and simulated voltage profiles.

The GPF is a filtration device preventing the release in the atmosphere of the particular matter generated during gasoline combustion in engines. As shown in [15], to describe the exhaust gas motion inside the filter porous structure, energy, mass, and momentum balance equations are used, and parameters—namely, coefficients for inlet temperature and velocity profiles and the external convective heat transfer coefficient—are determined from the identification framework proposed in this paper.

COMSOL models and Matlab identification codes for the two case studies are made available to the public on the *Mendeley Data* repository reported at the end of the paper and can be used as starting point to solve identification problems for systems holding similar characteristics.

The remainder of the paper is organized as follows. In Section 2, basic concepts on the development and implementation of COMSOL models are presented. The Matlab-COMSOL co-simulation framework is described in Section 3, where the parameter identification framework and corresponding code are introduced. In Section 4, the effectiveness of the identification framework is proved in two case studies: LIB and GPF. The paper is concluded in Section 5.

2. COMSOL Model Implementation: Basics

A generic COMSOL model is defined by a tree composed of four principal nodes [16]:

1. *Global Definitions*: In this node, global parameters, variables, functions, and couplings are defined. By default, there are two subnodes:
   - *Parameters*: storing the list of global model parameters;
   - *Materials*: storing the material properties.
2. *Component*: In this node, geometry (1D, 2D, or 3D), model equations, and mesh are defined.
3. *Study*: This node defines the type of study to be performed, e.g., stationary or time dependent, and the corresponding solver settings.
4. *Results*: This node stores the solution of a simulation. There are five additional subnodes:
   - *Datasets*: containing a list of solutions;
   - *Derived values*: storing values derived from the postprocessing of a solution;
   - *Tables*: storing solutions from probes (i.e., "virtual sensors");
   - *Export*: defining numerical data or images to be exported;
   - *Reports*: containing custom or automatically generated reports.

[1] The proposed approach can be also applied to ordinary differential equations and differential-algebraic equations.

FIGURE 1 COMSOL model tree.

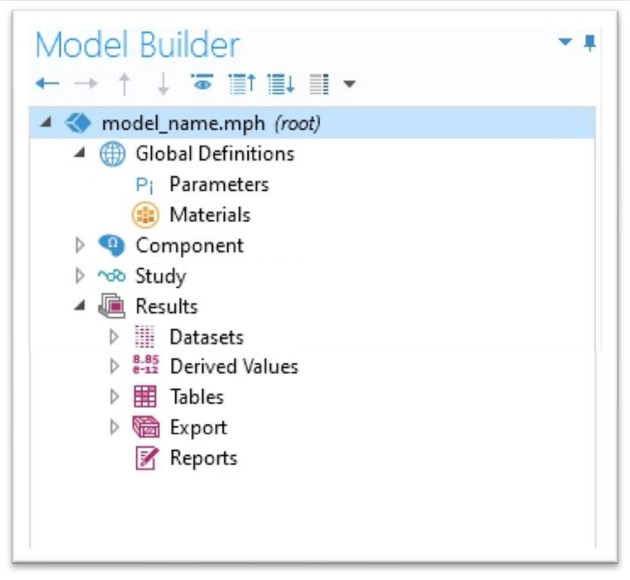


Figure 1 shows a typical COMSOL model tree inclusive of the nodes listed above.

3. Matlab and COMSOL Co-simulation

LiveLink for Matlab is a powerful feature of COMSOL Multiphysics that is used to set up the co-simulation with Matlab [17]. Thanks to this tool, Matlab can be used for the analysis and processing of simulation results, to change model parameters for sensitivity analysis, for example, using local sensitivity approaches, and for parameter identification [18]. This utility simplifies the analysis of COMSOL simulation results and model optimization.

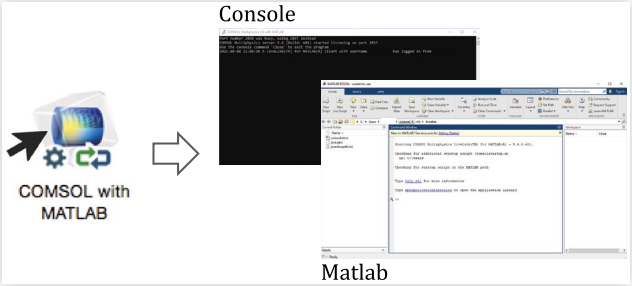
In this paper, we focus on developing an optimization framework that can be used for parameter identification for systems implemented in COMSOL. General Matlab commands to simulate a COMSOL model are introduced next, followed by the description of the identification framework.

3.1. Establishing the Communication

To establish the communication between Matlab and COMSOL, double-click on the *COMSOL with Matlab* icon as shown in Figure 2. This action automatically opens a console and a Matlab instance. The console shows the status of the communication and must remain open during the entire duration of the co-simulation. To track the COMSOL numerical solver progress, the following code should be typed in the Matlab command window:

```
ModelUtil.showProgress(true);                    [a]
```

FIGURE 2 COMSOL with Matlab.



3.2. Simulating COMSOL Models with Matlab

The first step to simulate a model in Matlab is to load the COMSOL file `model_name.mph` with the following command:

```
model = mphload('model_name.mph');               [b]
```

Before running the model, the initial (`t_init`) and final (`t_final`) simulation time instants are defined using the method `set`:

```
model.param.set('t_init',...                     [c]
   [num2str(t_init) '[s]']);
model.param.set('t_final',...
   [num2str(t_final) '[s]']);
```

When using `set`, the arguments must be converted into a string and then passed to the method. A parameter can be modified in Matlab with `set` only if it is defined inside the *Parameters* node of *Global Definitions* (Figure 1). Additionally, when performing a time-dependent study, the following syntax can be used to specify the time instants at which results of a simulation are stored:

```
model.param.set('t_step',...                     [d]
   [num2str(t_step) '[s]']);
model.study('<study>').feature('time').set(
   'tlist',['range(' num2str(t_init) ',',...
                     num2str(t_step) ',',...
                     num2str(t_final) ')']);
```

In COMSOL, the previous code defines a time vector between `t_init` and `t_final`, with sampling time `t_step`. The tag `<study>` is used as identifier for a generic COMSOL study. It is worth noticing that the previous code does not control the time-step taken by the numerical solver, which could use an adaptive or fixed time-stepping.

Once the model is loaded and the simulation time window is defined, we can run a COMSOL study with the following code:

```
model.study('<study>').run;                      [e]
```

The outcome of the study is saved in *Results*. As an example, to export data from a table with identifier <table>, one can use the following code:

```
model.result.export('<table>').run;        [f]
```

As shown in Figure 3, to export data from a table or plot, subnodes *Table* and *Plot* must be added to *Export*.

3.3. Parameter Identification of COMSOL Models with Matlab

We define the parameter vector for a generic dynamical system as

$$\theta = \left[\vartheta_1 \ \dots \ \vartheta_i \ \dots \ \vartheta_{N_{par}} \right] \in \mathbb{R}^{1 \times N_{par}} \qquad \text{Eq. (1)}$$

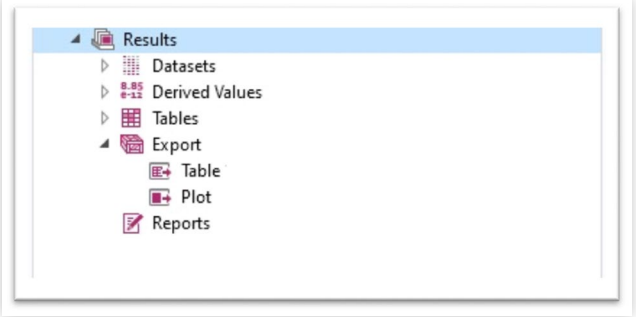
where N_{par} is the number of parameters to be identified and ϑ_i, with $i = 1, \dots, N_{par}$, is the i-th unknown parameter. The optimal parameter vector θ^* is obtained by solving the following optimization problem:

$$\underset{\theta}{\textbf{minimize}} \qquad J(\theta) = f(\theta;\ \mathcal{Y}, Y)$$

subject to

$$\begin{aligned} &(a)\ \text{Governing equations (PDEs)} \\ &(b)\ \begin{cases} \text{lb}_1 \le \vartheta_1 \le \text{ub}_1 \\ \dots \\ \text{lb}_i \le \vartheta_i \le \text{ub}_i \\ \dots \\ \text{lb}_{N_{par}} \le \vartheta_{N_{par}} \le \text{ub}_{N_{par}} \end{cases} \end{aligned} \qquad \text{Eq. (2)}$$

FIGURE 3 Export data from *Results*: *Table* and *Plot* subnodes.



The objective function $J(\theta)$ is a function of $\mathcal{Y} \in \mathbb{R}^{1 \times N}$ and $Y \in \mathbb{R}^{1 \times N}$, i.e., of the experimental data and simulation results, respectively.[2] The minimization of $J(\theta)$ is subject to the governing equations (a) and inequalities (b) that define the feasible search space for the parameters to be identified. Given ϑ_i, $\text{lb}_i \in \text{LB}$ and ub_i 2 UB define the lower and upper bounds, respectively. LB and UB are $1 \times N_{par}$ real vectors collecting the lower and upper bounds for all the parameters inside θ.

3.4. PSO-Based Framework

The optimization problem in Equation 2 is solved using the PSO algorithm [19]. PSO is a computational method that solves optimization problems iteratively, starting with a population of candidate solutions, called particles, and moving these particles in the search space to find the optimum. PSO can deal with nonlinearities in both the objective function and constraints, proving to be a good candidate algorithm to solve identification problems in multiphysics simulations. The first step of PSO is to define the swarm size N_{swarm}, i.e., the number of particles in the swarm, which controls the number of candidate solutions used to explore the search space. The initial position of the swarm is defined by the following matrix:

$$\Theta_0 = \begin{bmatrix} \theta_0^1 \\ \vdots \\ \theta_0^j \\ \vdots \\ \theta_0^{N_{swarm}} \end{bmatrix} = \begin{bmatrix} \vartheta_{1,0}^1 & \cdots & \vartheta_{i,0}^1 & \cdots & \vartheta_{N_{par},0}^1 \\ & & \vdots & & \\ \vartheta_{1,0}^j & \cdots & \vartheta_{i,0}^j & \cdots & \vartheta_{N_{par},0}^j \\ & & \vdots & & \\ \vartheta_{1,0}^{N_{swarm}} & \cdots & \vartheta_{i,0}^{N_{swarm}} & \cdots & \vartheta_{N_{par},0}^{N_{swarm}} \end{bmatrix} \in \mathbb{R}^{N_{swarm} \times N_{par}} \qquad \text{Eq. (3)}$$

where each row in Equation 3 defines the initial guess for the j-th particle, with $j = 1, \dots, N_{swarm}$.[3] During the first PSO iteration ($i_{PSO} = 0$), the COMSOL model is simulated for each candidate solution in Equation 3, and simulation results are stored in the following matrix:

$$\mathbf{Y}_0 = \begin{bmatrix} Y_0^1 \\ \vdots \\ Y_0^j \\ \vdots \\ Y_0^{N_{swarm}} \end{bmatrix} \in \mathbb{R}^{N_{swarm} \times N} \qquad \text{Eq. (4)}$$

[2] For the LIB, experimental and simulated voltages are defined as $\mathcal{V}$ and V, respectively. For the GPF, experimental and simulated temperatures are defined as $\mathcal{T}$ and T, respectively.

[3] If the number of rows in Θ_0 is lower than N_{swarm}, the PSO algorithm will create initial guesses for the remaining particles.

Evaluating the objective function $J(\theta)$ for each particle j in Equation 3, the following vector is obtained:

$$\mathbf{J}(\Theta_0)=\left[J\left(\theta_0^1\right)\ \dots\ J\left(\theta_0^j\right)\ \dots\ J\left(\theta_0^{N_{swarm}}\right)\right]\in\mathbb{R}^{1\times N_{swarm}} \quad \text{Eq. (5)}$$

The best candidate solution θ_0^* is the one that minimizes the vector (5), i.e.,

$$\begin{aligned}\theta_0^* &= \arg\min_{\theta}\left(\mathbf{J}(\Theta_0)\right)\\ J_0^* &= \min\left(\mathbf{J}(\Theta_0)\right)\end{aligned} \quad \text{Eq. (6)}$$

with J_0^* the value of the objective function at θ_0^*. In the second PSO iteration ($i_{PSO}=1$), the position of the particles is updated according to the following matrix:

$$\Theta_1=\begin{bmatrix}\theta_1^1\\ \vdots\\ \theta_1^j\\ \vdots\\ \theta_1^{N_{swarm}}\end{bmatrix}=\begin{bmatrix}\vartheta_{1,1}^1 & \cdots & \vartheta_{i,1}^1 & \cdots & \vartheta_{N_{par},1}^1\\ & & \vdots & & \\ \vartheta_{1,1}^j & \cdots & \vartheta_{i,1}^j & \cdots & \vartheta_{N_{par},1}^j\\ & & \vdots & & \\ \vartheta_{1,1}^{N_{swarm}} & \cdots & \vartheta_{i,1}^{N_{swarm}} & \cdots & \vartheta_{N_{par},1}^{N_{swarm}}\end{bmatrix}\in\mathbb{R}^{N_{swarm}\times N_{par}} \quad \text{Eq. (7)}$$

The motion of particles in the search space is governed by the weighting factors w_{self} and w_{social} controlling the attraction of a particle to the best location it has visited and to the best locations visited by the neighboring particles, respectively. For each θ_1^j in Equation 7, the COMSOL model is solved and results are used to evaluate the objective function $J(\theta)$.

Values of the objective function are collected in the following vector:

$$\mathbf{J}(\Theta_1)=\left[J\left(\theta_1^1\right)\ \dots\ J\left(\theta_1^j\right)\ \dots\ J\left(\theta_1^{N_{swarm}}\right)\right]\in\mathbb{R}^{1\times N_{swarm}} \quad \text{Eq. (8)}$$

The best candidate solution θ_1^* is updated as follows:

$$\begin{aligned}\theta_1^* &= \arg\min_{\theta}\left(\left[J_0^*\ \ \mathbf{J}(\Theta_1)\right]\right)\\ J_1^* &= \min\left(\left[J_0^*\ \ \mathbf{J}(\Theta_1)\right]\right)\end{aligned} \quad \text{Eq. (9)}$$

where the minimization is performed while accounting for the best solution at the previous iteration ($i_{PSO}=0$). For a generic PSO iteration, the following update rule is used:

$$\begin{aligned}\theta_{i_{PSO}}^* &= \arg\min_{\theta}\left(\left[J_{i_{PSO}-1}^*\ \ \mathbf{J}\left(\Theta_{i_{PSO}}\right)\right]\right)\\ J_{i_{PSO}}^* &= \min\left(\left[J_{i_{PSO}-1}^*\ \ \mathbf{J}\left(\Theta_{i_{PSO}}\right)\right]\right)\end{aligned} \quad \text{Eq. (10)}$$

The PSO algorithm is terminated when the relative change in $J_{i_{PSO}}^*$ over the last *#iter* PSO iterations is less than the tolerance *tol*. Eventually, the optimal $\theta_{N_{PSO}}^*$ and $J_{N_{PSO}}^*$, solution of the optimization problem in Equation 2, are returned. Settings of the PSO algorithm, in terms of N_{swarm}, w_{self}, w_{social}, *#iter*, and *tol*, determine how the search space is explored. PSO generally cannot guarantee the global optimality of the solution, and given a search space, it provides a solution that is optimal only compared to its neighbors. Therefore, depending on the characteristics of the optimization problem, the settings should be carefully tuned to ensure PSO convergence. This can be done by trial and error or by understanding the effect of different settings on PSO [13].

The flowchart in Figure 4 summarizes the identification process described in the previous paragraph and shows the

FIGURE 4 Identification framework exploiting Matlab and COMSOL co-simulation.

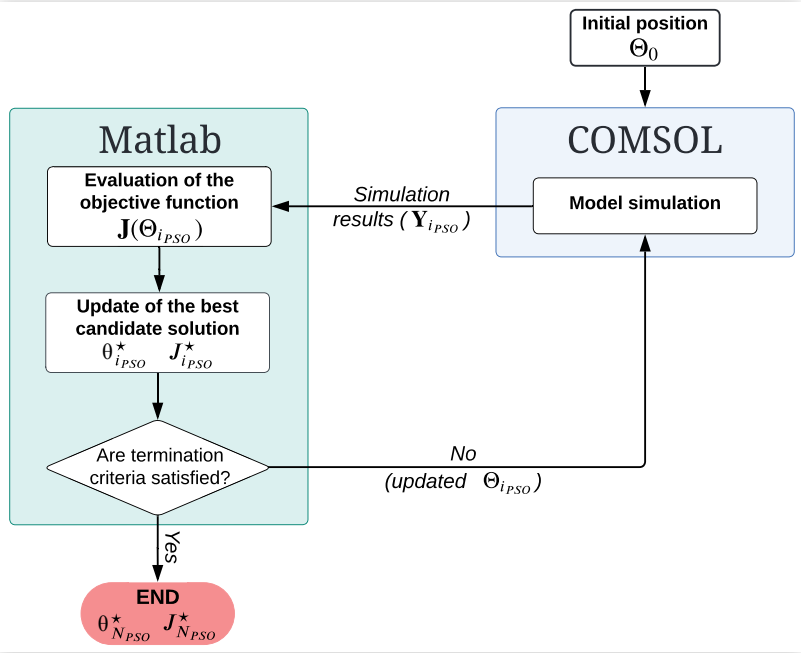


TABLE 1 List of Matlab variables used in Listing 1.

Matlab variable	Meaning
initial_position	Θ_0
lower_bound	LB
upper_bound	UB
n_vars	N_{par}
swarm_size	N_{swarm}
self_weight	w_{self}
social_weight	w_{social}
max_iter	*#iter*
tol	*tol*
x_opt	$\theta^*_{N_{PSO}}$
J_opt	$J^*_{N_{PSO}}$

communication links between Matlab and COMSOL. The PSO algorithm is implemented in Matlab and uses COMSOL to perform simulations for the different candidate solutions in $\Theta_{i_{PSO}}$ and build the vectors $\mathbf{J}(\Theta_{i_{PSO}})$. In Matlab, the PSO algorithm is initialized in the `main.m` script shown in Listing 1, where the initial position of the swarm and PSO options are defined according to Table 1. PSO is run with the following code:

```
[x_opt,J_opt] = particleswarm(                [g]
   @fit_comsol_model,n_vars,lower_bound,...
   upper_bound,options);
```

where `x_opt` and `J_opt` are $\theta^*_{N_{PSO}}$ and $J^*_{N_{PSO}}$, respectively, and `particleswarm` is the Matlab implementation of the PSO algorithm.

In Listing 2, the function `fit_comsol_model.m` simulates the COMSOL model for each $\theta^j_{i_{PSO}}$ and computes the value of the objective function. In the script, $\theta^j_{i_{PSO}}$ is indicated by `x`

LISTING 1 `main.m`

```
% Initial position                                              1
initial_position = ...                                          2
  [theta_1_1       theta_1_2       ...  theta_1_N              3
   theta_2_1       theta_2_2       ...  theta_2_N              4
   ...                                                          5
   theta_Nswarm_1  theta_Nswarm_2  ...  theta_Nswarm_N];        6
                                                                7
% Lower and upper bounds                                       8
lower_bound = [lb_theta_1  lb_theta_2 ...                       9
               lb_theta_N];                                     10
upper_bound = [ub_theta_1  ub_theta_2 ...                       11
               ub_theta_N];                                     12
                                                                13
% PSO options                                                  14
options = optimoptions('particleswarm',...                      15
              'SwarmSize',swarm_size,...                        16
              'MaxStallIterations',max_iter,...                 17
              'FunctionTolerance',tol,...                       18
              'InitialSwarmMatrix',initial_position,...         19
              'SocialAdjustmentWeight',social_weight,...        20
              'SelfAdjustmentWeight',self_weight);              21
n_vars = length(initial_position);                              22
                                                                23
% Run PSO                                                      24
[x_opt,J_opt] = particleswarm(@fit_comsol_model,n_vars,         25
     lower_bound,upper_bound,options);
```

LISTING 2 `fit_comsol_model.m`

```
function J = fit_comsol_model(x)                                1
% Parameters to be identified                                   2
theta_1 = x(1);                                                 3
theta_2 = x(2);                                                 4
...                                                             5
theta_N = x(n_vars);                                            6
                                                                7
% Loading of the COMSOL model                                  8
[b]                                                             9
                                                                10
% Time vector setting                                          11
[c]                                                             12
[d]                                                             13
                                                                14
% Parameter vector setting                                     15
model.param.set('theta_1' ,[num2str(theta_1) '[-]']);           16
model.param.set('theta_2' ,[num2str(theta_2) '[-]']);           17
...                                                             18
model.param.set('theta_N' ,[num2str(theta_N) '[-]']);           19
                                                                20
% Model run                                                    21
[e]                                                             22
                                                                23
% Export results                                               24
[f]                                                             25
                                                                26
% Objective function calculation                               27
J = f(y_exp,y_sim)                                              28
end                                                             29
```

and code snippets from Section 3.2 (labeled with letters) are reused to load the model `[b]`, set the time vector `[c][d]`, run the model `[e]`, and export the results `[f]`. Relying on the method `set`, lines 16 to 19 of Listing 2 modify the values of the parameters in the COMSOL model before running the simulation. Ultimately, line 28 computes the objective function. Matlab variables for Listing 2 are summarized in Table 2.

4. Case Studies

The identification routine presented in Figure 4 is applied in two case studies: DFN battery model and GPF model, both developed in COMSOL Multiphysics 5.6. Matlab R2020b is used to run the PSO for parameter identification, with settings for the two case studies shown in Appendix A (Table A.1).

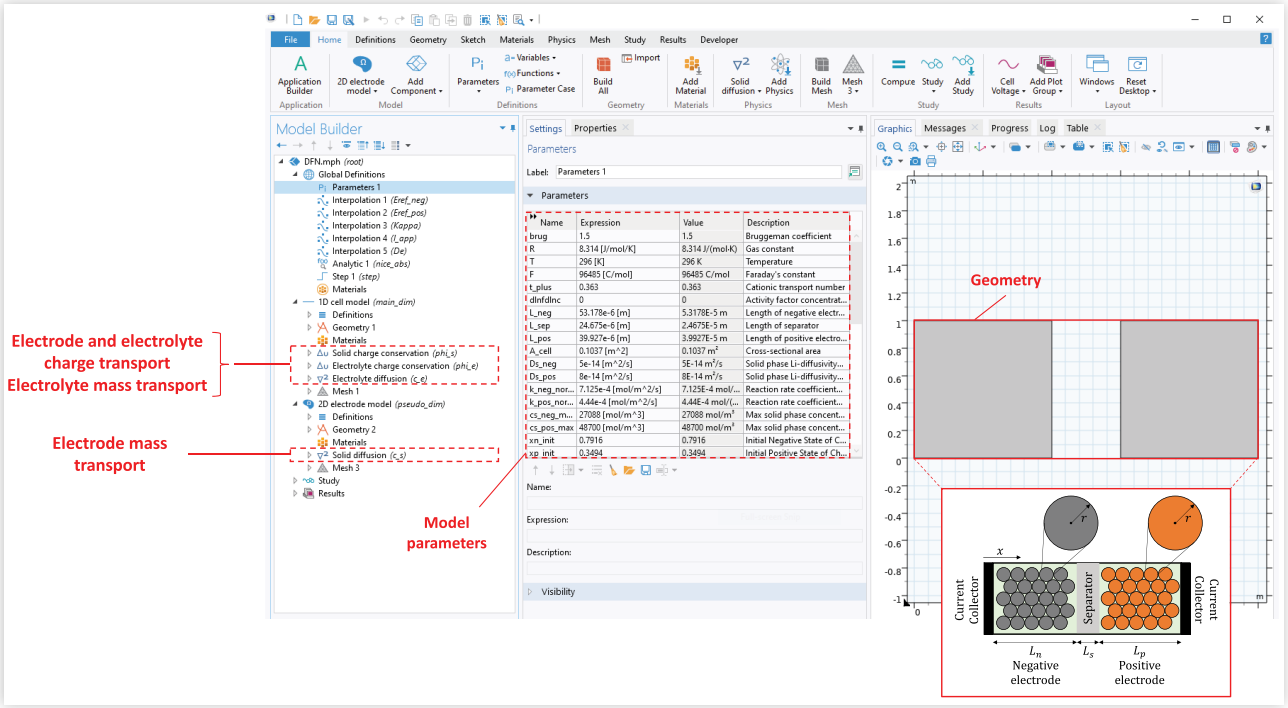
4.1. LIB Cell

The identification framework is tested on a DFN battery electrochemical model [14]. This pseudo-two-dimensional model relies on mass and charge transport PDEs to describe lithium-ion motion in the electrolyte and its intercalation/deintercalation in the electrodes. Electrodes are treated as spherical particles of a homogeneous medium in which mass transport

TABLE 2 List of Matlab variables used in Listing 2.

Matlab variable	Meaning
x	$\theta^j_{i_{PSO}}$
J	$J(\theta)$
y_exp	$\mathcal{Y}$
y_sim	$Y^j_{i_{PSO}}$

FIGURE 5 DFN battery model: COMSOL implementation. Mass and charge transport equations are highlighted and a pictorial representation of the battery is shown in the bottom right corner. Both positive and negative electrodes are modeled as a conglomerate of spherical particles.



is resolved along the radius of the particle (r). In the electrolyte, one-dimensional transport of mass and charge, along the x coordinate, is assumed. The governing equations of the DFN model are listed in Appendix B (Table B.1) and the COMSOL implementation, following [20], is shown in Figure 5. Mass and charge transport equations are highlighted, and a pictorial representation of the battery (with electrodes composed of spherical particles) is shown.

The identification framework presented in Section 3.3 is used to identify the following parameter vector:

$$\theta = \left[L_n \;\; L_s \;\; L_p \;\; A_{cell} \;\; D_{s,n} \;\; D_{s,p} \;\; k_{0,n} \;\; k_{0,p} \;\; R_c \;\; x_{n,init} \;\; x_{p,init} \;\; c_{s,n,\max} \;\; c_{s,p,\max} \;\; \eta_{s,n} \;\; \eta_{s,p} \;\; \eta_{e,n} \;\; \eta_{e,s} \;\; \eta_{e,p} \right] \quad \text{Eq. (11)}$$

with L_n, L_s, and L_p the thicknesses of the negative electrode (n), separator (s), and positive electrode (p) as shown in Figure 5, A_{cell} the cross-sectional area, $D_{s,n}$ and $D_{s,p}$ the solid-phase diffusion coefficients, $k_{0,n}$ and $k_{0,p}$ the reaction rate constants, $x_{n,init}$ and $x_{p,init}$ the initial stoichiometric coefficients, $c_{s,n,max}$ and $c_{s,p,max}$ the maximum solid-phase lithium concentrations, $\eta_{s,n}$ and $\eta_{s,p}$ the active material volume fractions, and $\eta_{e,n}$, $\eta_{e,s}$, and $\eta_{e,p}$ the electrolyte volume fractions.

The parameter vector in Equation 11 is identified minimizing the following objective function:

$$J(\theta) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{V}(i) - V(\theta;i)\right)^2} \quad \text{Eq. (12)}$$

where N is the number of data samples and $\mathcal{V}$ and V are the experimental and simulated voltage profiles, respectively. In this paper, model parameters are identified for a Sony 2.1Ah US18650VTC4 NMC cylindrical cell discharged at 1C constant current at 23°C. Properties of this cell, such as cut-off voltages and operating temperatures, can be found in [21].

In Table 3, the lower and upper bounds, initial position, and identified parameter vector θ^* are listed. To show the goodness of the identification results, a comparison between experimental and simulated data is proposed in Figure 6. The one-shot identification of 18 parameters could lead to overfitting experimental data, and as proposed in [22], a more robust approach splitting the identification between geometrical, stoichiometric, and transport parameters could be used. In this paper, to show the potentialities of the proposed framework, we stick to the one-shot identification. Additional details on the battery model and governing equations can be found in [22], where the proposed identification framework is used to identify parameters of both DFN and full homogenized macroscale battery models.

4.2. Gasoline Particulate Filter

The identification framework is tested on a second case study, i.e., the GPF. The GPF is a filtration device used in gasoline direct-injection engines to trap particulates generated during combustion and prevent their release into the atmosphere.

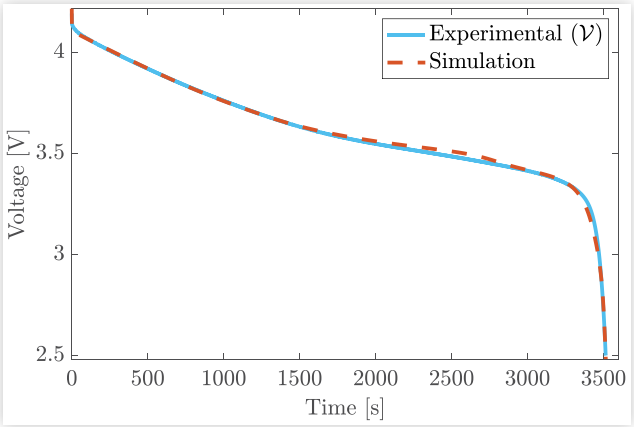
In [15], a two-dimensional model for a clean[4] and uncoated GPF accounting for mass, energy, and momentum

[4] A clean filter has no trapped particulate matter.

TABLE 3 DFN identification results.

Parameter	Lower bound	Upper bound	Initial position	Identified vector θ^*	Unit
L_n	45×10^{-6}	55×10^{-6}	50×10^{-6}	45×10^{-6}	[m]
L_s	20×10^{-6}	32×10^{-6}	26×10^{-6}	32×10^{-6}	[m]
L_p	35×10^{-6}	45×10^{-6}	40×10^{-6}	44.7×10^{-6}	[m]
A_{cell}	0.1006	0.1120	0.1063	0.1083	[m^2]
$D_{s,n}$	1×10^{-14}	6×10^{-14}	3.5×10^{-14}	2.3×10^{-14}	[m^2/s]
$D_{s,p}$	2×10^{-14}	10×10^{-14}	6×10^{-14}	9.9×10^{-14}	[m^2/s]
$k_{0,n}$	2×10^{-4}	14×10^{-4}	8×10^{-4}	2×10^{-4}	[$Am^{2.5}/mol^{1.5}$]
$k_{0,p}$	1×10^{-4}	8×10^{-4}	4.5×10^{-4}	1×10^{-4}	[$Am^{2.5}/mol^{1.5}$]
R_c	0.0240	0.0360	0.0300	0.0359	[Ω]
$x_{n,init}$	0.7500	0.8000	0.7750	0.7759	[-]
$x_{p,init}$	0.3100	0.3600	0.3350	0.3392	[-]
$c_{s,n,max}$	26,000	31,500	28,750	31,318	[mol/m^3]
$c_{s,p,max}$	45,000	50,000	47,500	49,089	[mol/m^3]
$\eta_{s,n}$	0.5400	0.6600	0.6000	0.6315	[-]
$\eta_{s,p}$	0.5000	0.6000	0.5500	0.5097	[-]
$\eta_{e,n}$	0.2800	0.3600	0.3200	0.2803	[-]
$\eta_{e,s}$	0.3500	0.4500	0.4000	0.4494	[-]
$\eta_{e,p}$	0.2800	0.3600	0.3200	0.2822	[-]
$J^* = 0.014$ [V]					

FIGURE 6 Comparison between 1C experimental and simulated voltage profiles for a Sony 2.1Ah US18650VTC4 NMC cylindrical cell tested at 23°C.



transport is developed. The model considers 47 single channels, each composed of one inlet channel, one outlet channel, walls, and plugs, where transport PDEs are resolved in space (x and r coordinates) and time. Figure 7 shows the COMSOL implementation of the GPF model with highlights on mass, energy, and momentum balance equations, model parameters, geometry, and single channel. A summary of the model equations, governing the transport dynamics inside the filter, is shown in Appendix B (Table B.2). Geometrical properties of the GPF, available in the COMSOL model (at the link specified at the end of the paper) and in [15], were provided by the industrial partner of the project.

The identification framework described in Section 3.3 is used to identify the following parameter vector:

$$\theta = \left[\mathcal{A} \;\; \mathcal{B} \;\; h_{ext}\right] \qquad \text{Eq. (13)}$$

$\mathcal{A}$ and $\mathcal{B}$ control the shape of the inlet exhaust gas velocity and temperature profiles, which are defined as follows:

$$\begin{aligned} &\textit{Velocity profile}: \\ &\mathbf{u}(r,t)\big|_{x=0} = u_x^{inlet}\,\mathbf{i} + u_r^{inlet}\,\mathbf{j} \\ &\begin{cases} u_x^{inlet} = \mathcal{A}\;\; v_{inlet}(t)\left[1-\left(\dfrac{D/2-r}{D/2}\right)^2\right] \\ u_r^{inlet} = 0 \end{cases} \end{aligned} \qquad \text{Eq. (14)}$$

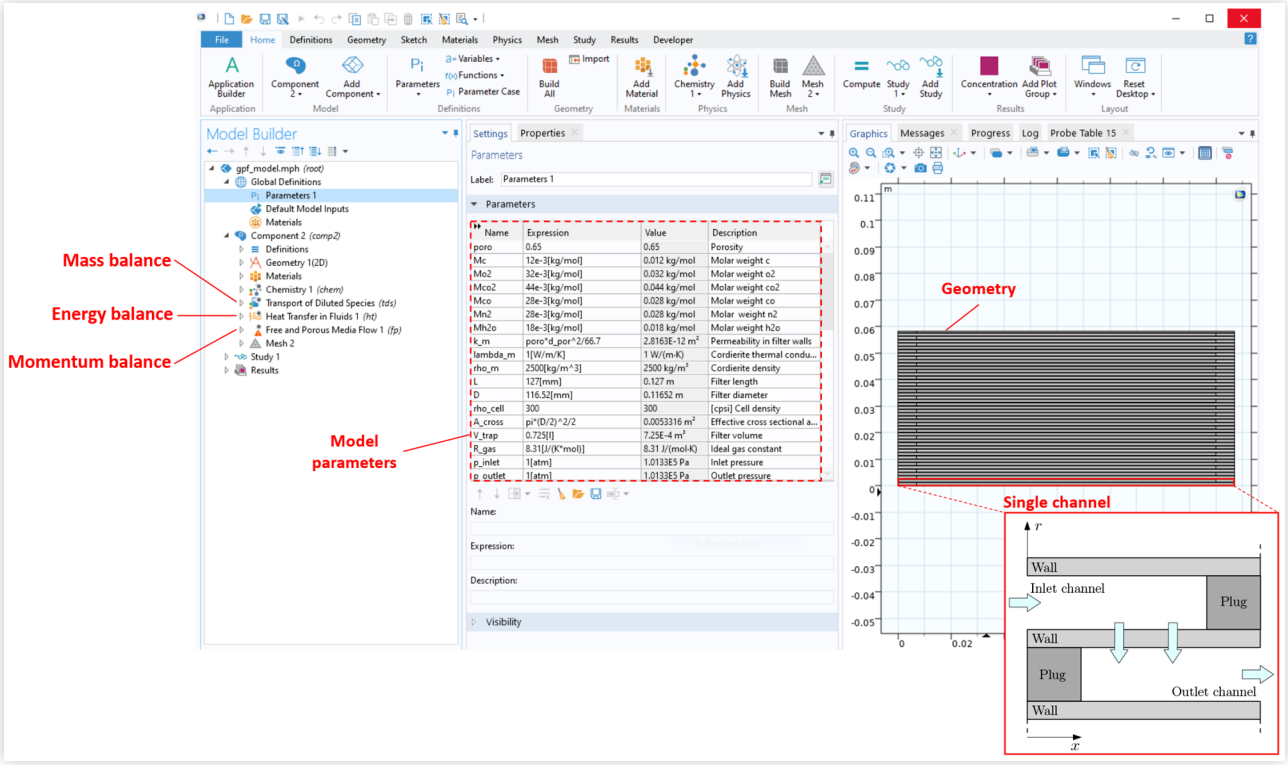
$$\begin{aligned} &\textit{Temperature profile}: \\ &T(r,t)\big|_{x=0} = \mathcal{B}\;\; \mathcal{T}_{inlet}(t) + \\ &+ \left(1-\mathcal{B}\right)\;\mathcal{T}_{inlet}(t)\left[1-\left(\dfrac{D/2-r}{D/2}\right)^2\right] \end{aligned} \qquad \text{Eq. (15)}$$

with D the diameter of the filter, v_{inlet} the inlet exhaust gas velocity, u_x^{inlet} and u_r^{inlet} the x and r components of the inlet velocity profile, and $\mathcal{T}_{inlet}$ the inlet exhaust gas temperature. The parameter h_{ext} in Equation 13 is the convective heat transfer coefficient between the filter and the external environment, entering the following boundary condition:

$$-\mathbf{n}\cdot\left(k\nabla T\right) = h_{ext}\left(T_{ext} - T\right) \qquad \text{Eq. (16)}$$

where k is the exhaust gas thermal conductivity and T_{ext} the ambient temperature.

FIGURE 7 GPF: COMSOL implementation. Mass, energy, and momentum balance equations are highlighted in the model tree on the left. In the middle, model parameters are shown. CAD geometry and one single channel are depicted on the right.



The parameter vector θ in Equation 13 is identified minimizing the following objective function:

$$J(\theta)=\sum_{\iota}\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{T}_{\iota}(i)-T_{\iota}(\theta;i)\right)^{2}} \qquad \text{Eq. (17)}$$

where $\mathcal{T}_{\iota}$ and T_{ι} are the ι-th experimental and simulated temperatures, respectively. The objective function is the summation of the root mean squared errors computed at different locations inside the filter, namely, $\iota \in \{2,3,7,8,10\}$ (Figure 8). This allows to capture the spatial temperature gradient, leading to a robust identification of the parameters.

Identified parameters, together with the lower/upper bounds, initial position, and minimum of the objective functions, are summarized in Table 4. A comparison between experimental and simulated temperature profiles in the center location of the filter (#8) is shown in Figure 8. For additional details on the GPF modeling and selection of the numerical solver settings, readers are referred to [15, 23], respectively.

4.3. Numerical Solution

Identification problems are solved on a Dell Precision 7920 Tower equipped with an Intel Xeon Gold 6136 CPU at 3.00 GHz and 32.0 GB of RAM. The minimization of cost functions Equations 12 and 17 is shown in Figure 9, where each subplot depicts the number of model evaluations together with the objective function value. In red, the minimum value of the cost function, corresponding to the solution of the identification problem in Tables 3 and 4, is highlighted. Given the PSO settings in Table A.1, parameter vectors minimizing the objective functions are obtained in 56 h and 155 h for the battery and GPF case studies, respectively.

FIGURE 8 Comparison between identification results and experimental temperature data measured in the center location of the filter.

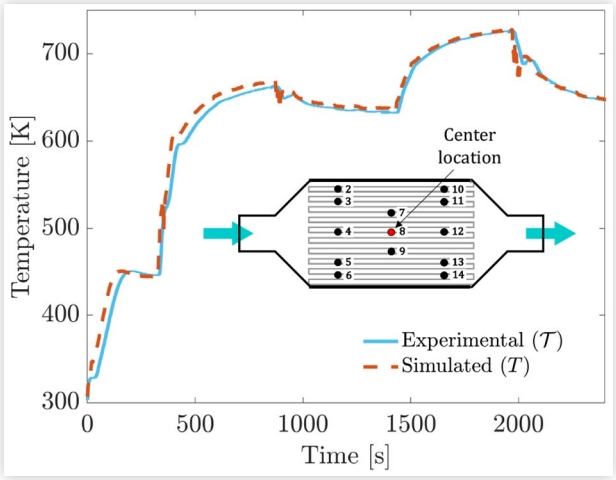
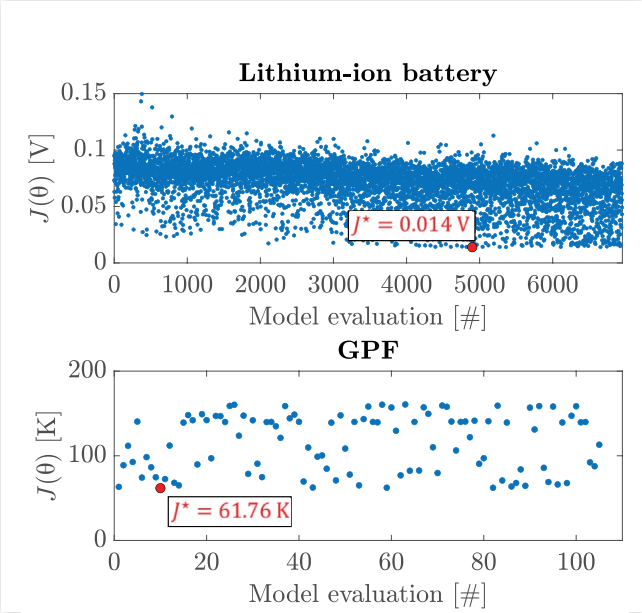


TABLE 4 GPF identification results.

Parameter	Lower bound	Upper bound	Initial position	Identified vector θ^*	Unit
$\mathcal{A}$	1.63	2.06	1.72	2.04	[–]
$\mathcal{B}$	0.82	1.00	0.91	0.91	[–]
h_{ext}	27.37	37.03	32.20	34.90	[W/(m² K)]
J^* = 61.76 [K]					

FIGURE 9 Objective function versus iterations number for the LIB and GPF case studies.



5. Conclusions

The paper provides guidelines to set up and run COMSOL and Matlab co-simulations with the aim of model parameter identification. As shown by the LIB and GPF case studies, the proposed approach is general and can be used for the identification of unknown model parameters in various scenarios and with customizable objective functions. The framework proposed in this paper is based on PSO; however, the routine could be modified to use other gradient-free optimization algorithms, such as GA (which uses similar principles to PSO). In the battery field, the proposed framework adds to the available software for DFN development and, specifically, identification. As a matter of fact, except for DEARLIBS [24], current publicly available tools (such as PyBaMM [25] and LIONSIMBA [26]) focus on forward DFN model simulation and do not include embedded identification routines.

Matlab co-simulation scripts, together with LIB and GPF COMSOL models, are publicly available. Interested readers can freely download and use these resources to develop identification routines for COMSOL models.

Acknowledgments

The authors gratefully acknowledge the support of Fiat Chrysler Automobiles (FCA) US LLC for granting permission to utilize experimental data from previous research collaboration. This work was supported by the National Science Foundation (NSF), USA through CAREER Award number CMMI-#1839050.

The DFN battery COMSOL model was developed by Harikesh Arunachalam [24].

Nomenclature

0 - initial position

***** - optimum

ϑ - model parameter

$\boldsymbol{\theta}$ - model parameter vector

$\boldsymbol{\Theta}$ - particles' position

i - index indicating the *i*-th parameter to be identified

i_{PSO} - PSO iteration

j - index indicating the *j*-th particle

k - index indicating the positive electrode, negative electrode, and separator (battery) or chemical species in the exhaust gas (GPF)

lb, ub - lower and upper bounds for one parameter

LB, UB - vector of the lower and upper bounds

J - objective function

J - vector of objective function evaluations

N - number of samples

N_{par} - number of parameters to be identified

N_{PSO} - number of PSO iterations

N_{swarm} - number of particles

t - time [s]

w_{self}, ***w_{social}*** - PSO weights

x, r - axial and radial coordinates [m]

Y - simulation results

$\mathcal{Y}$ - experimental data

Battery Case Study

$\eta_{e,k}$ - electrolyte volume fraction, $k = (n, s, p)$ [–]

$\eta_{s,k}$ - active material volume fraction, $k = (n, p)$ [–]

$\phi_{e,k}$ - electrolyte-phase potential, $k = (n, s, p)$ [V]

$\phi_{s,k}$ - solid-phase potential, $k = (n, p)$ [V]

A_{cell} - cell cross-sectional area [m²]

a_k - specific surface area, $k = (n, p)$ [m²/m³]

$c_{s,k}$ - solid-phase lithium concentration, $k = (n, p)$ [mol/m³]

$c_{s,surf,k}$ - solid-phase lithium concentration at the surface, $k = (n, p)$ [mol/m³]

$c_{s,k,max}$ - maximum solid-phase lithium concentration, $k = (n, p)$ [mol/m³]

$c_{e,k}$ - lithium electrolyte-phase concentration, $k = (n, s, p)$ [mol/m^3]

$D_{e,k}^{eff}$ - effective electrolyte-phase diffusion coefficient, $k = (n, s, p)$ [m^2/s]

$D_{s,k}$ - solid-phase diffusion coefficient, $k = (n, p)$ [m^2/s]

F - Faraday constant [C/mol]

I_{app} - applied current [A]

$J_{Li,k}$ - intercalation current density, $k = (n, s, p)$ [A/m^3]

$k_{0,k}$ - reaction rate constant, $k = (n, p)$ [Am$^{2.5}$/mol$^{1.5}$]

$K_{e,k}^{eff}$ - effective electrolyte conductivity, $k = (n, s, p)$ [S/m]

$K_{s,k}^{eff}$ - effective electrode conductivity, $k = (n, s, p)$ [S/m]

L_k - region thickness, $k = (n, s, p)$ [m]

n, s, p - negative electrode (n), separator (s), and positive electrode (p)

R - universal gas constant [J/(mol·K)]

R_c - contact resistance, $k = (n, s, p)$ [Ω]

T - temperature [K]

t_+ - transference number [–]

$U_{0,k}$ - open-circuit potential, $k = (n, p)$ [V]

V - simulated voltage profile [V]

$\mathcal{V}$ - experimental voltage profile [V]

$x_{k,init}$ - initial stoichiometric coefficient, $k = (n, p)$ [–]

GPF Case Study

p - exhaust gas pressure [Pa]

ε_P^w - wall porosity [–]

ι - index indicating the temperature location

κ_w - wall permeability [m^2]

μ - exhaust gas dynamic viscosity [Pa s]

ρ - exhaust gas density [kg/m^3]

ρ_{plug} - plug density [kg/m^3]

ρ_w - wall density [kg/m^3]

$[c_k]$ - k-th species concentration [mol/m^3]

C_p - exhaust gas specific heat capacity at constant pressure [J/(kg K)]

C_w - wall specific heat capacity [J/(kg K)]

D_k - k-th species diffusion coefficient [m^2/s]

eff - effective property

h_{ext} - external convective heat transfer coefficient [W/(m^2 K)]

$\mathbf{i}, \mathbf{j} \in \mathbb{R}^2$ - unit vectors

k_{gas} - exhaust gas thermal conductivity [W/(mK)]

k_{plug} - plug conductivity [W/(mK)]

k_w - wall thermal conductivity [W/(mK)]

$\mathbf{n} \in \mathbb{R}^2$ - normal vector

T - simulated temperature [K]

$\mathcal{T}$ - measured temperature [K]

T_{ext} - room temperature [K]

$\mathbf{u}$ - exhaust gas velocity field [m/s]

$\mathbf{v}$ - exhaust gas Darcy velocity field [m/s]

v_{inlet} - inlet exhaust gas velocity [m/s]

w - wall property

References

1. Rao, R.V. and Savsani, V.J., *Mechanical Design Optimization Using Advanced Optimization Techniques* (London, UK: Springer, 2012).
2. Guzzella, L., Sciarretta, A. et al., *Vehicle Propulsion Systems*. Vol. 1 (Berlin, Germany: Springer, 2007).
3. Savaresi, S.M., Poussot-Vassal, C., Spelta, C., Sename, O. et al., *Semi-active Suspension Control Design for Vehicles* (Elsevier, 2010).
4. Dickinson, E.J., Ekström, H., and Fontes, E., "COMSOL Multiphysics®: Finite Element Software for Electrochemical Analysis. A Mini-Review," *Electrochemistry Communications* 40 (2014): 71-74.
5. Vencels, J., Birjukovs, M., Kataja, J., and Råback, P., "Microwave Heating of Water in a Rectangular Waveguide: Validating EOF-Library against COMSOL Multiphysics and Existing Numerical Studies," *Case Studies in Thermal Engineering* 15 (2019): 100530.
6. Obraztsov, N., Subbotin, D., Popov, V., Frolov, V. et al., "Modelling of Heating of Plasma-Chemical Reactor in COMSOL Multiphysics," *Journal of Physics: Conference Series* 1038 (2018): 012137.
7. Jegede, A.O., Gualtieri, C., Zeeman, G., and Bruning, H., "Three-Phase Simulation of the Hydraulic Characteristics of an Optimized Chinese Dome Digester Using COMSOL Multiphysics," *Renewable Energy* 157 (2020): 530-544.
8. Sun, X., Luo, H., and Soga, K., "A Coupled Thermal–Hydraulic–Mechanical–Chemical (THMC) Model for Methane Hydrate Bearing Sediments Using COMSOL Multiphysics," *Journal of Zhejiang University-Science A* 19, no. 8 (2018): 600-623.
9. Ljung, L., "System Identification," in *Signal Analysis and Prediction* (Springer, 1998), 163-173.
10. COMSOL Multiphysics, "Optimization Module: User's Guide," https://doc.comsol.com/.
11. Gurobi Optimization, https://www.gurobi.com/.
12. Conforti, M., Cornuéjols, G., Zambelli, G. et al., *Integer Programming*. Vol. 271 (Cham: Springer, 2014).
13. Das, S., Abraham, A., and Konar, A., "Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives," in *Advances of Computational Intelligence in Industrial Systems* (Berlin, Germany: Springer, 2008), 1-38.
14. Doyle, M., Fuller, T.F., and Newman, J., "Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell," *Journal of the Electrochemical Society* 140, no. 6 (1993): 1526.
15. Pozzato, G., Hoffman, M.A., and Onori, S., "Multi-channel Physics-Based Modeling and Experimental Validation of an Uncoated Gasoline Particulate Filter in Clean Operating

Conditions," in *2017 American Control Conference (ACC)*, Seattle, WA, 2017, IEEE, 5392-5397.

16. COMSOL Multiphysics, COMSOL Multiphysics User Guide (version 4.3 a), 2012.
17. COMSOL Multiphysics, "LiveLink™ for Matlab: User's Guide," https://doc.comsol.com/.
18. Allam, A. and Onori, S., "Online Capacity Estimation for Lithium-Ion Battery Cells via an Electrochemical Model-Based Adaptive Interconnected Observer," *IEEE Transactions on Control Systems Technology* 29, no. 4 (2021): 1636-1651.
19. Ebbesen, S., Kiwitz, P., and Guzzella, L., "A Generic Particle Swarm Optimization Matlab Function," in *2012 American Control Conference (ACC)*, Montreal, QC, Canada, 2012, IEEE, 1519-1524.
20. Plett, G.L., *Battery Management Systems, Volume I: Battery Modeling*. Vol. 1 (Norwood: Artech House, 2015).
21. Arunachalam, H., "A New Multiscale Modeling Framework for Lithium-Ion Battery Dynamics: Theory, Experiments, and Comparative Study with the Doyle-Fuller-Newman Model," PhD thesis, Clemson University, Clemson, SC, 2017.
22. Arunachalam, H. and Onori, S., "Full Homogenized Macroscale Model and Pseudo-2-Dimensional Model for Lithium-Ion Battery Dynamics: Comparative Analysis, Experimental Verification and Sensitivity Analysis," *Journal of the Electrochemical Society* 166, no. 8 (2019): A1380.
23. Levine, K., Pozzato, G., and Onori, S., "Modeling of Regeneration Dynamics in Gasoline Particulate Filters and Sensitivity Analysis of Numerical Solutions," SAE Technical Paper 2022-01-0556, 2022, doi:https://doi.org/10.4271/2022-01-0556.
24. Lee, S.B. and Onori, S., "A Robust and Sleek Electrochemical Battery Model Implementation: A Matlab Framework," *Journal of the Electrochemical Society* 168, no. 9 (2021): 090527.
25. PyBaMM, https://www.pybamm.org/.
26. Torchio, M., Magni, L., Gopaluni, R.B., Braatz, R.D. et al., "LIONSIMBA: A Matlab Framework Based on a Finite Volume Model Suitable for Li-Ion Battery Design, Simulation, and Control," *Journal of the Electrochemical Society* 163, no. 7 (2016): A1192.

Appendix A

Table A.1 shows the PSO settings for the battery DFN and GPF models, respectively. For the GPF, only 15 particles are used because the initial position is a good guess of the parameter vector. Among the parameters listed in Table A.1, `swarm_size` and `max_iter` are the most important to control the convergence of the algorithm.

TABLE A.1 PSO settings for the DFN battery model and GPF model.

Matlab variable	DFN	GPF
`n_vars`	18	3
`swarm_size`	400	15
`self_weight`	0.3	0.2
`social_weight`	3.6	3.7
`max_iter`	5	5
`tol`	0.5×10^{-6}	0.5×10^{-6}

Appendix B

TABLE B.1 DFN battery model governing equations.

Electrode mass transport equation—$k = (n, p)$

$$\frac{\partial c_{s,k}}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}\left(D_{s,k} r^2 \frac{\partial c_{s,k}}{\partial r}\right) \quad \text{Eq. (18)}$$

Electrolyte mass transport equation—$k = (n, s, p)$

$$\eta_{e,k}\frac{\partial c_{e,k}}{\partial t} = \frac{\partial}{\partial x}\left(D_{e,k}^{eff}\frac{\partial c_{e,k}}{\partial x}\right) + \frac{(1-t_+)}{F} J_{Li,k} \quad \text{Eq. (19)}$$

Electrode charge transport equation—$k = (n, p)$

$$K_{s,k}^{eff}\frac{\partial^2 \phi_{s,k}}{\partial x^2} = J_{Li,k} \quad \text{Eq. (20)}$$

Electrolyte charge transport equation—$k = (n, s, p)$

$$-K_{e,k}^{eff}\frac{\partial^2 \phi_{e,k}}{\partial x^2} - \frac{2K_{e,k}^{eff} RT(1-t_+)}{F}\frac{\partial^2 \ln c_{e,k}}{\partial x^2} = J_{Li,k} \quad \text{Eq. (21)}$$

Intercalation current density—$k = (n, p)$

$$J_{Li,k} = a_k k_{0,k} \cdot \sqrt{c_{s,surf,k} \cdot c_{e,k}} \cdot \sqrt{\left(c_{s,k,max} - c_{s,surf,k}\right)} \cdot 2\sinh\left[\frac{0.5F}{RT}\left(\phi_{s,k} - \phi_{e,k} - U_{0,k}\right)\right], \quad \text{Eq. (22)}$$

$$J_{Li,s} = 0$$

Output voltage equation

$$V = \phi_s|_{x = L_n + L_s + L_p} - \phi_s|_{x=0} - R_c I_{app} \quad \text{Eq. (23)}$$

TABLE B.2 GPF model governing equations.

Inlet/outlet channels

$$\frac{\partial[c_k]}{\partial t} = -\mathbf{u}\cdot\nabla[c_k] + \nabla\cdot\left(D_k\nabla[c_k]\right) \quad \text{Eq. (24)}$$

$$\rho C_p\frac{\partial T}{\partial t} + \rho C_p\left(\mathbf{u}\cdot\nabla T\right) = \nabla\cdot\left(k_{gas}\nabla T\right) \quad \text{Eq. (25)}$$

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} \quad \text{Eq. (26)}$$

Walls

$$\varepsilon_p^w\frac{\partial[c_k]}{\partial t} = -\mathbf{v}\cdot\nabla[c_k] + \nabla\cdot\left(D_{eff,k}^w\nabla[c_k]\right) \quad \text{Eq. (27)}$$

$$\left(\rho C_p\right)_{eff}^w\frac{\partial T}{\partial t} + \rho C_p\left(\mathbf{v}\cdot\nabla T\right) = \nabla\cdot\left(k_{eff}^w\nabla T\right)$$
$$\left(\rho C_p\right)_{eff}^w = \left(1-\varepsilon_p^w\right)\rho_w C_w + \varepsilon_p^w\rho C_p$$
$$k_{eff}^w = \left(1-\varepsilon_p^w\right)k_w + \varepsilon_p^w k_{gas} \quad \text{Eq. (28)}$$

$$\frac{\rho}{\varepsilon_p^w}\left(\frac{\partial \mathbf{v}}{\partial t} + \frac{\mathbf{v}}{\varepsilon_p^w}\cdot\nabla\mathbf{v}\right) = -\nabla p + \frac{\mu}{\varepsilon_p^w}\nabla^2\mathbf{v} - \frac{\mu}{\kappa_w}\mathbf{v} \quad \text{Eq. (29)}$$

Plugs

$$\rho_{plug}C_w\frac{\partial T}{\partial t} = \nabla\cdot\left(k_{plug}\nabla T\right) \quad \text{Eq. (30)}$$

Web Resources

Matlab scripts for co-simulation and COMSOL models are available at the following link: https://data.mendeley.com/datasets/298yzrnw35/2

ISSN 0148-7191