

COBRAPRO: A MATLAB toolbox for Physics-based Battery Modeling and Co-simulation Parameter Optimization

Sara Ha ¹ and Simona Onori ²

¹*Mechanical Engineering, Stanford University, 440 Escondido Mall,
Stanford 94305, CA, USA*

²*Energy Science and Engineering, Stanford University, 367 Panama
Mall, Stanford 94305, CA, USA*

01 March 2024

Summary

COBRAPRO (**Co**-simulation **B**attery Modeling for **A**ccelerated **P**arameter **O**ptimization) is a physics-based battery modeling software with the capability to perform closed-loop parameter optimization using experimental data. COBRAPRO is based on the Doyle-Fuller-Newman (DFN) model (Doyle et al., 1993), which is the most widely-accepted high-fidelity model that considers the lithium-ion transport and charge conservation in the liquid electrolyte and solid electrodes, and kinetics at the solid and liquid interface during lithium intercalation and deintercalation. Such physics-based models have found applications in battery design (Dai & Srinivasan, 2016), (Couto et al., 2023) and advanced battery management systems to ensure reliable and safe operation of electric vehicles (Kolluri et al., 2020). The DFN model is characterized by several physical parameters, such as geometric, stoichiometric, concentration, transport, and kinetic parameters, which are often unknown and need to be determined to accurately predict battery response under various usage scenarios. Direct measurements through cell tear-down experiments are a viable but labor-intensive process (Ecker et al., 2015), (Schmalstieg et al., 2018), (Chen et al., 2020). Furthermore, parameters obtained through experimental characterization may necessitate further calibration to ensure suitability for use in the DFN model (Chen et al., 2020), since the model is a simplified representation of a real battery, assuming perfectly spherical particles, neglecting electrode heterogeneity, and considering internal dynamics in only one dimension. With COBRAPRO, users can noninvasively identify parameters for any given battery using readily available current-voltage data from a battery cyclers. COBRAPRO

optimizes the DFN parameters by minimizing the error between the simulated and experimentally observed data through an embedded parameter optimization routine.

Statement of need

Even though parameter calibration is required to accurately predict the dynamical behavior of actual batteries, current DFN modeling tools lack the capability to perform parameter identification (Sulzer et al., 2021) (Torchio et al., 2016) (Smith & Bazant, 2017) (Berliner et al., 2021).

COMSOL Multiphysics© (COMSOL AB, Stockholm, Sweden, 2023) is a commercially available finite element modeling software commonly used to simulate the DFN model. Although COMSOL lacks a built-in parameter identification feature, it was demonstrated that COMSOL’s *LiveLink™ for MATLAB*© can be used to establish communication between COMSOL and MATLAB for parameter optimization (Pozzato & Onori, 2023). This framework allows users to leverage the versatile suite of optimizers in MATLAB while running COMSOL to generate the model output. However, the expensive licensing fee and proprietary nature of COMSOL create barriers to public access, limiting collaboration and code reproducibility.

In contrast, several open-source DFN model simulation tools have emerged, such as PyBaMM (Sulzer et al., 2021), LIONSIMBA (Torchio et al., 2016), PETLION (Berliner et al., 2021), DEARLIBS (Lee & Onori, 2021), fastDFN (Scott Moura, 2016), and MPET (Smith & Bazant, 2017). Among these packages, DEARLIBS is the only software equipped for closed-loop parameter identification using experimental data. Other packages resort to literature-derived parameter values and lack the ability to predict real battery behavior. Taking inspiration from DEARLIBS, COBRAPRO aims to address three primary challenges in the DFN model:

Challenge 1. Computational complexity

- **Issue:** The DFN model is also known as the pseudo-two-dimensional (P2D) model due to the coupling of the cell thickness (x-direction) and radial particle (r-direction) dimensions. This coupling of dimensions contributes to the model’s computational complexity.
- **Solution:** COBRAPRO leverages a fast solver to significantly improve the model computation speed compared to DEARLIBS. For 10 discretized points in each domain of the cell (positive and negative electrodes, separator, and positive and negative active material particles) at 1C discharge, COBRAPRO solves the DFN model in 0.708 seconds, while DEARLIBS takes 2.54 minutes, which is a two orders of magnitude improvement (~257 times). Under identical simulation conditions, LIONSIMBA and PyBaMM computed the model in 1.13 seconds and 0.237 seconds, re-

spectively, demonstrating comparable performance to COBRAPRO. For larger discretization points, COBRAPRO exhibits up to three orders of magnitude improvement in computation speed compared to DEARLIBS.

Challenge 2. Consistent initial conditions

- **Issue:** The partial differential equations (PDEs) governing the DFN model are discretized in x- and r-directions to form a system of ordinary differential equations (ODEs) and algebraic equations (AEs), referred to as differential-algebraic equations (DAEs). To solve the DAE system, the correct initial conditions of the AEs are required, which are typically not known *a priori* for the DFN model. Inconsistent initial conditions result in either a failure to start the simulation or the model diverging towards an incorrect solution (Methkar et al., 2011).
- **Solution:** The single step approach (Lawder et al., 2015), a robust initialization method, is implemented in COBRAPRO that automatically determines the initial conditions and seamlessly simulates the DFN model.

Challenge 3. Unknown model parameters

- **Issue:** As highlighted earlier, battery parameters are frequently unknown, and even if obtained through experimental characterization, parameter calibration is essential to accurately simulate the DFN model.
- **Solution:** A co-simulation parameter optimization framework is implemented in COBRAPRO. The particle swarm optimization (PSO), a gradient-free population-based algorithm, is employed due to its suitability for nonlinear models like the DFN model. The PSO aims to optimize parameters by minimizing the objective function defined as the error between the experimental and simulated voltage and state of charge (SOC) curves. COBRAPRO employs PSO using MATLAB’s Parallel Computing Toolbox to accelerate PSO convergence through multicore processing.

Core Capabilities

- **Parameter identification routine:** PSO optimizes parameters using experimental current-voltage data
- **DFN model implementation:** PDEs are discretized with finite volume method (FVM), and the DAE system is solved with SUNDIALS IDA
- **Solid particle radial discretization options:**
 - FVM and 3rd order Hermite interpolation used to calculate particle surface concentration, accounting for sharp concentration gradients near the particle surface (Xu et al., 2023)
 - Finite difference method (FDM)
- **DAE initialization options:**
 - Single-step approach (Lawder et al., 2015)

- SUNDIALS IDACalcIC
- **Simulating battery cycling:**
 - Constant current (CC) profiles
 - Hybrid pulse power characterization (HPPC) profiles
 - Dynamic current profiles
- **Parameter identifiability analysis:**
 - Local sensitivity analysis (LSA): Perturbs parameters around their nominal values and evaluates their sensitivity with respect to voltage and SOC
 - Correlation analysis: Calculates linear correlation between two parameters
 - Utilizes user defined sensitivity and correlation index thresholds to determine the set of identifiable parameters

Example: Case Study on LG 21700-M50T Cells

As a demonstration of COBRAPRO, we conduct a case study aimed at parameterizing a fresh LG 21700-M50T cell using the C/20 capacity test, HPPC, and driving cycle data (Pozzato et al., 2022). In this example, we break down the identification problem by systematically grouping parameters in each identification step, as shown in Figure 1. This multi-step approach is proposed to improve the identifiability of parameters instead of identifying all the unknown parameters simultaneously (Arunachalam & Onori, 2019).

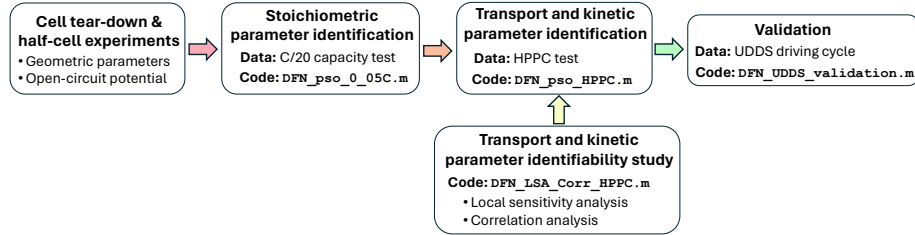


Figure 1: Case study: Parameter identification procedure on LG 21700-M50T cells.

First, the geometric parameters and open-circuit potential functions are extracted from measurements conducted in cell tear-down and half-cell experiments on LG 21700-M50 cells, as reported by (Chen et al., 2020). Next, the C/20 capacity test data is used to identify the stoichiometric parameters in the example code `DFN_pso_0_05C.m`. We then conduct a parameter identifiability study, comprising of LSA and correlation analysis, to pinpoint parameters with high sensitivity to HPPC voltage and SOC while maintaining low correlation with other parameters (`DFN_LSA_Corr_HPPC.mat`). Next, we calibrate the identifiable electrolyte transport and kinetic parameters using HPPC data in the example code `DFN_pso_HPPC.m`. Finally,

validation of the identified parameters is carried out on the urban dynamometer driving schedule (UDDS) data in the `DFN_UDDS_validation.m` code. The `DFN_pso_0_05C.m` and `DFN_pso_HPPC.m` files are located in the `Examples/Parameter_Identification_Routines` directory and `DFN_UDDS_validation.m` is located in `Examples/Parameter_Identification_Results`.

C/20 Capacity Test Identification

In `DFN_pso_0_05C.m`, the `User Input` section is used to define the parameter names, the upper and lower parameter bounds for the PSO, experimental data, etc. A preview of the `User Input` section is provided here.

First, load the `Parameters_LG_INR21700_M50.m` function, which outputs a `param` structure containing the nominal DFN parameters for a LG INR21700-M50 cell (Chen et al., 2020) and the DFN simulation settings, e.g., discretization method, DAE initialization method, constant or variable current type, and etc.:

```
%% User Input
% Load nominal parameters
param = Parameters_LG_INR21700_M50;
```

Enter a mat file name to save PSO results. The mat file will contain an updated `param` structure with the identified parameters from the PSO:

```
% Enter mat file name where your PSO results will be stored
file_name = 'identified_parameters_0_05C';
```

Define the names of the parameters you want to identify. Here, we identify the stoichiometric parameters θ_p^{100} (`theta100_p`), θ_n^{100} (`theta100_n`), θ_p^0 (`theta0_p`), and θ_n^0 (`theta0_n`):

```
% Enter names of parameters to identify (make sure names match the
% parameter names in "param" structure containing the nominal parameters)
param_CC = {'theta100_p', 'theta100_n', 'theta0_p', 'theta0_n'};
```

Define the lower and upper bounds of the parameters defined in `param_CC`:

```
% Enter lower and upper bounds of parameters to identify
% theta100_p
lower_bounds.theta100_p = 0.22;
upper_bounds.theta100_p = 0.34;
% theta100_n
lower_bounds.theta100_n = 0.7;
upper_bounds.theta100_n = 1;
% theta0_p
lower_bounds.theta0_p = 0.7;
upper_bounds.theta0_p = 1;
% theta0_n
lower_bounds.theta0_n = 0.015;
upper_bounds.theta0_n = 0.04;
```

Load your time, current, and voltage experimental data. In this example, load the C/20 capacity test data:

```
% Load Experimental Data
%-----
%   t: Should be a vector consisting of your time experiment data      [s] (Mx1)
%   I: Should be a vector consisting of your current experiment data    [A] (Mx1)
%   V: Should be a vector consisting of your voltage experiemntal data [V] (Mx1)
%   -> where M is the total number of data points in your experiment
%-----
% C/20 capacity test conducted on LG INR21700 M50T cells
load('data_INR21700_M50T/capacity_test_data_W8_Diag1.mat')
% Assign your data variables to t, I, and V
t = t_data;
I = I_data;
V = V_data;
```

Once the all user inputs have been defined, run the DFN_pso_0_05C.m code to start the PSO. Once the PSO is finished, the code prints the identified parameter values, and the HPPC voltage and SOC objective function values to the Command Window:

```
Displaying identified values...
-----
theta100_p:
Identified value: 0.26475
0.22(lower) | 0.27(initial) | 0.34(upper)
-----
theta100_n:
Identified value: 0.77842
0.7(lower) | 0.9014(initial) | 1(upper)
-----
theta0_p:
Identified value: 0.89385
0.7(lower) | 0.9084(initial) | 1(upper)
-----
theta0_n:
Identified value: 0.029818
0.015(lower) | 0.0279(initial) | 0.04(upper)

Displaying objective function values...
-----
J_V =0.0033403 [-]
J_V =11.8445 [mV]
J_SOCp =0.030231 [%]
J_SOCn =0.019037 [%]
J_tot =0.003833 [-]
```

The code also plots the simulation results generated from the identified parameters and the experimental data, as shown in [Figure 2](#) and [Figure 3](#).

Run `Examples/Parameter_Identification_Results/DFN_pso_0_05C_identification.m` to view the C/20 identification results shown here.

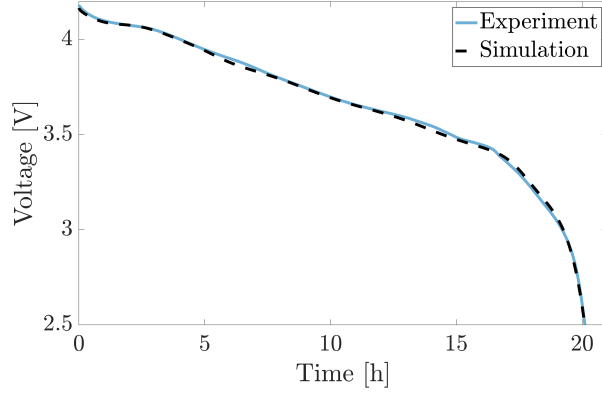


Figure 2: C/20 capacity test voltage identification results.

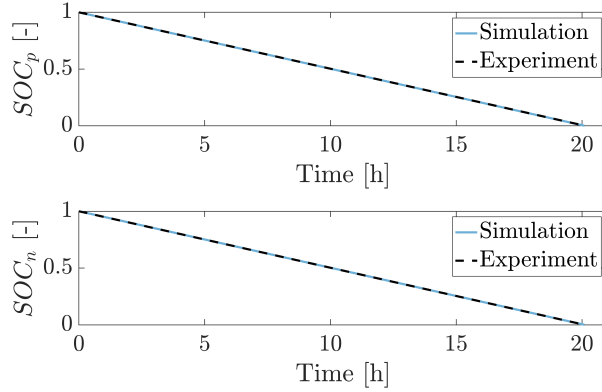


Figure 3: C/20 capacity test positive and negative electrode SOC identification results.

HPPC Identification

The `DFN_pso_HPPC.m` file's `User Input` section is similar to the one described in `DFN_pso_0_05C.m`. First, load your `param` structure, which contains the nominal DFN parameters and any previously identified parameter values. In this example, we load the `identified_parameters_0_05C.mat` file, which contains stoichiometric parameter identification results:

```

%% User Input
% Load nominal parameters and identified stoichiometric parameters
% from C/20 capacity test data
load('identified_parameters_0_05C.mat','param')

```

When defining the names of the HPPC parameters to identify, users can manually type the parameters (Option 1) or load the parameter identifiability results generated from `DFN_LSA_Corr_HPPC.m` (Option 2).

In Option 1, all the unknown transport and kinetic parameters are identified, consisting of the reaction rate constants in the electrodes k_p (**kp**) and k_n (**kn**), electrolyte diffusivity D_e (**De**), electrolyte conductivity κ (**Kappa**), transference number t_+ (**t1_constant**), initial electrolyte concentration (**c0**), and solid phase diffusivities $D_{s,p}$ (**Dsp**) and $D_{s,n}$ (**Dsn**):

```

%-----
% Option 1: Enter names of parameters to identify (make sure names match the
% parameter names in "param" structure containing nominal parameters)
%-----
param_HPPC = {'Dsp' 'Dsn' 't1_constant' 'kp' 'kn' 'c0' 'De' 'Kappa'};

```

Option 2 uses identifiability analysis results from `DFN_LSA_Corr_HPPC.m`, which produces two sets of parameters: **LSA_identifiable** and **corr_identifiable**. The former includes parameters that have sensitivities higher than the user-defined threshold (**beta_LSA**). The latter consists of parameters with high sensitivity and correlation coefficients lower than the specified correlation threshold (**beta_corr**).

In this example, identification of the **LSA_identifiable** parameter set is investigated:

```

%-----
% Option 2: Load identifiable parameters from identifiability analysis
% conducted in "Examples/Local_Sensitivity_Analysis/DFN_LSA_Corr_HPPC.m"
%-----
% 'LSA_identifiable' -> parameters with sensitivity higher than beta_LSA
% 'corr_identifiable' -> parameters determined through corr. analysis with
%                       a corr. threshold of beta_corr
%-----
load('DFN_identification_results/HPPC_identifiable_params.mat',...
     'LSA_identifiable','corr_identifiable')
param_HPPC = LSA_identifiable;

```

Enter the MAT file name to save an updated param structure containing the PSO results:

```

% Enter mat file name where your PSO results will be stored
file_name = 'identified_parameters_HPPC_noCorr';

```

Define the upper and lower bounds for each parameter in **param_HPPC**:

```

% Enter lower and upper bounds of parameters to identify
% Dsp
pct = 0.2; % perturbation coeff
lower_bounds.Dsp = 10^(log10(param.Dsp)*(1+pct));
upper_bounds.Dsp = 10^(log10(param.Dsp)*(1-pct));
...

```

Load the time, current, and voltage vectors generated from the HPPC data:

```

% Load Experimental Data
% HPPC test conducted on LG INR21700 M50T cells
load('data_INR21700_M50T/HPPC_data_W8_Diag1.mat')

```

Once all user inputs has been defined, run the code to start the PSO. Once the PSO is completed, the identified parameter and objective function values are printed to the Command Window:

Displaying identified values...

```

-----
Dsp:
Identified value: 5.5423e-15
5.278e-18(lower) | 4e-15(initial) | 3.0314e-12(upper)
-----
kn:
Identified value: 8.6471e-09
2.987e-15(lower) | 6.7159e-12(initial) | 1.51e-08(upper)
-----
c0:
Identified value: 1166.3688
500(lower) | 1000(initial) | 1500(upper)
-----
Dsn:
Identified value: 2.1618e-14
6.6407e-17(lower) | 3.3e-14(initial) | 1.6399e-11(upper)

```

Displaying objective function values...

```

-----
J_V =0.0038222 [-]
J_V =13.4785 [mV]
J_SOCp =0.13299 [%]
J_SOCn =0.17276 [%]
J_tot =0.0068797 [-]

```

Similar to DFN_pso_0_05C.m, the simulation results generated from the identified parameters are plotted against the experimental data, as shown in [Figure 4](#) and [Figure 5](#).

Run Examples/Parameter_Identification_Results/DFN_pso_HPPC_identification.m to view the HPPC identification results shown here.

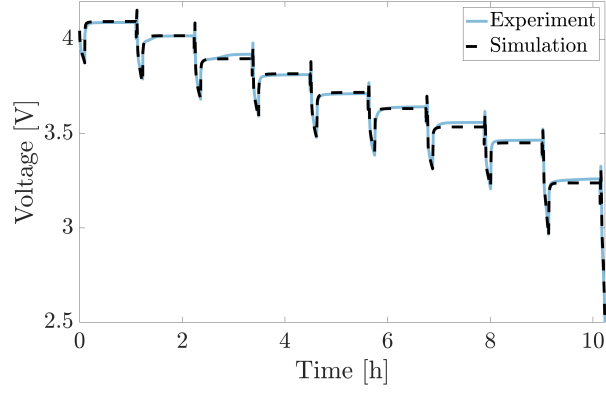


Figure 4: HPPC voltage identification results.

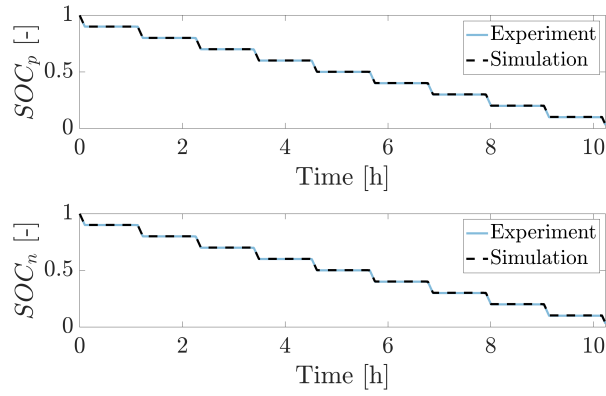


Figure 5: HPPC positive and negative electrode SOC identification results.

UDDS Driving Cycle Validation

In the code `Examples/Parameter_Identification_Results/DFN_pso_UDDS_validation.m`, the identified parameters from the C/20 capacity test and HPPC data are validated using the UDDS driving cycle. The model is simulated under the UDDS profile and compared against the experimental UDDS data.

In the **User Input** section, load the parameter values identified from C/20 and HPPC data:

```
%% User Input  
% Load identification results  
load('identified_parameters_HPPC_noCorr.mat', 'param')
```

and load the experimental UDDS data:

```
% Load Experimental Data  
% HPPC test conducted on LG INR21700 M50T cells  
load('data_INR21700_M50T/UDDS_W8_cyc1.mat')
```

The objective function is printed to the Command Window:

Displaying objective function values...

```
-----  
J_V =0.0039168 [-]  
J_V =14.3911 [mV]  
J_SOCp =0.032573 [%]  
J_SOCn =0.015161 [%]  
J_tot =0.0043941 [-]
```

The simulation results and experimental data are plotted as shown in [Figure 6](#) and [Figure 7](#).

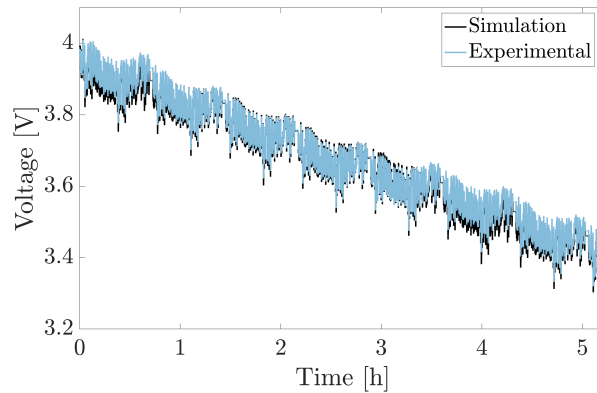


Figure 6: UDDS voltage identification results.

Visit COBRAPRO's [Github](#) to view all example codes:

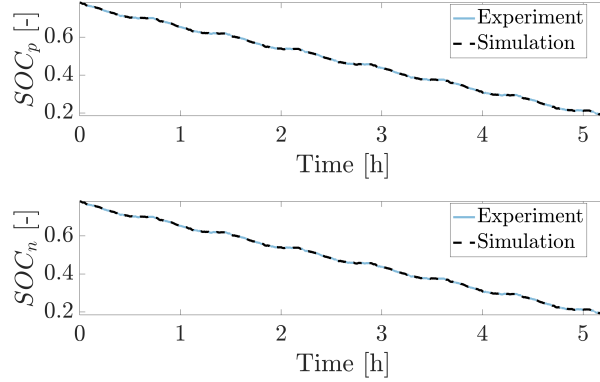


Figure 7: UDDS positive and negative electrode SOC identification results.

- **Examples/Parameter_Identification_Routines:** Parameter identification examples
 - `DFN_pso_0_05C.m`: Parameter identification using C/20 capacity test data
 - `DFN_pso_HPPC.m`: Parameter identification using HPPC data
- **Examples/Parameter_Identification_Results:** Load parameter identification results
 - `DFN_pso_0_05C_identification.m`: C/20 capacity test identification results
 - `DFN_pso_HPPC_identification.m`: HPPC identification results
 - `DFN_pso_UDDS_validation.m`: UDDS validation results
- **Examples/Cycling:** Simulating battery cycling examples
 - `cycle_CC.m`: CC cycling and model output visualization
 - `cycle_HPPC.m`: HPPC simulation and model output visualization
 - `cycle_UDDS.m`: UDDS simulation and model output visualization
- **Examples/Local_Sensitivity_Analysis:** LSA and correlation analysis examples
 - `DFN_LSA_Corr_CC.m`: LSA and correlation analysis on CC profile
 - `DFN_LSA_Corr_HPPC.m`: LSA and correlation analysis on HPPC profile

Acknowledgements

The authors thank the Bits and Watts Initiative within the Precourt Institute for Energy at Stanford University for its partial financial support. We thank Dr. Le Xu for all the insightful discussions that greatly contributed to the enhancement of COBRAPRO. We extend our thanks to Alexis Geslin, Joseph Lucero, and Maitri Uppaluri for testing COBRAPRO and providing valuable feedback.

References

- Arunachalam, H., & Onori, S. (2019). 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(8), A1380–A1392. <https://doi.org/10.1149/2.0051908jes>
- Berliner, M. D., Cogswell, D. A., Bazant, M. Z., & Braatz, R. D. (2021). Methods—PETLION: Open-Source Software for Millisecond-Scale Porous Electrode Theory-Based Lithium-Ion Battery Simulations. *Journal of The Electrochemical Society*, 168(9), 090504. <https://doi.org/10.1149/1945-7111/ac201c>
- Chen, C.-H., Brosa Planella, F., O'Regan, K., Gastol, D., Widanage, W. D., & Kendrick, E. (2020). Development of Experimental Techniques for Parameterization of Multi-scale Lithium-ion Battery Models. *Journal of The Electrochemical Society*, 167(8), 080534. <https://doi.org/10.1149/1945-7111/ab9050>
- COMSOL AB, Stockholm, Sweden. (2023). *COMSOL multiphysics* & copy; v. 6.2. www.comsol.com
- Couto, Luis. D., Charkhgard, M., Karaman, B., Job, N., & Kinnaert, M. (2023). Lithium-ion battery design optimization based on a dimensionless reduced-order electrochemical model. *Energy*, 263, 125966. <https://doi.org/10.1016/j.energy.2022.125966>
- Dai, Y., & Srinivasan, V. (2016). On Graded Electrode Porosity as a Design Tool for Improving the Energy Density of Batteries. *Journal of The Electrochemical Society*, 163(3), A406–A416. <https://doi.org/10.1149/2.0301603jes>
- Doyle, M., Fuller, T. F., & Newman, J. (1993). Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell. *Journal of The Electrochemical Society*, 140(6), 1526–1533. <https://doi.org/10.1149/1.2221597>
- Ecker, M., Tran, T. K. D., Dechent, P., Käbitz, S., Warnecke, A., & Sauer, D. U. (2015). Parameterization of a Physico-Chemical Model of a Lithium-Ion Battery: I. Determination of Parameters. *Journal of The Electrochemical Society*, 162(9), A1836–A1848. <https://doi.org/10.1149/2.0551509jes>
- Kolluri, S., Aduru, S. V., Pathak, M., Braatz, R. D., & Subramanian, V. R. (2020). Real-time Nonlinear Model Predictive Control (NMPC) Strategies using Physics-Based Models for Advanced Lithium-ion Battery Management System (BMS). *Journal of The Electrochemical Society*, 167(6), 063505. <https://doi.org/10.1149/1945-7111/ab7bd7>
- Lawder, M. T., Ramadesigan, V., Suthar, B., & Subramanian, V. R. (2015). Extending explicit and linearly implicit ODE solvers for index-1 DAEs. *Computers & Chemical Engineering*, 82, 283–292. <https://doi.org/10.1016/j.compchemeng.2015.07.002>
- Lee, S. B., & Onori, S. (2021). A Robust and Sleek Electrochemical Battery Model Implementation: A MATLAB® Framework. *Journal of The Electrochemical Society*, 168(9), 090527. <https://doi.org/10.1149/1945-7111/ac22c8>
- Methekar, R. N., Ramadesigan, V., Pirkle, J. C., & Subramanian, V. R.

- (2011). A perturbation approach for consistent initialization of index-1 explicit differential-algebraic equations arising from battery model simulations. *Computers & Chemical Engineering*, 35(11), 2227–2234. <https://doi.org/10.1016/j.compchemeng.2011.01.003>
- Pozzato, G., Allam, A., & Onori, S. (2022). Lithium-ion battery aging dataset based on electric vehicle real-driving profiles. *Data in Brief*, 41, 107995. <https://doi.org/10.1016/j.dib.2022.107995>
- Pozzato, G., & Onori, S. (2023). A General Matlab and COMSOL Co-simulation Framework for Model Parameter Optimization: Lithium-Ion Battery and Gasoline Particulate Filter Case Studies. *Automotive Technical Papers*, 2023-01-5047. <https://doi.org/10.4271/2023-01-5047>
- Schmalstieg, J., Rahe, C., Ecker, M., & Sauer, D. U. (2018). Full Cell Parameterization of a High-Power Lithium-Ion Battery for a Physico-Chemical Model: Part I. Physical and Electrochemical Parameters. *Journal of The Electrochemical Society*, 165(16), A3799–A3810. <https://doi.org/10.1149/2.0321816jes>
- Scott Moura. (2016). *fastDFN*. <https://github.com/scott-moura/fastDFN>
- Smith, R. B., & Bazant, M. Z. (2017). Multiphase Porous Electrode Theory. *Journal of The Electrochemical Society*, 164(11), E3291–E3310. <https://doi.org/10.1149/2.0171711jes>
- Sulzer, V., Marquis, S. G., Timms, R., Robinson, M., & Chapman, S. J. (2021). Python Battery Mathematical Modelling (PyBaMM). *Journal of Open Research Software*, 9(1), 14. <https://doi.org/10.5334/jors.309>
- Torchio, M., Magni, L., Gopaluni, R. B., Braatz, R. D., & Raimondo, D. M. (2016). 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(7), A1192–A1205. <https://doi.org/10.1149/2.0291607jes>
- Xu, L., Cooper, J., Allam, A., & Onori, S. (2023). Comparative Analysis of Numerical Methods for Lithium-Ion Battery Electrochemical Modeling. *Journal of The Electrochemical Society*, 170(12), 120525. <https://doi.org/10.1149/1945-7111/ad1293>