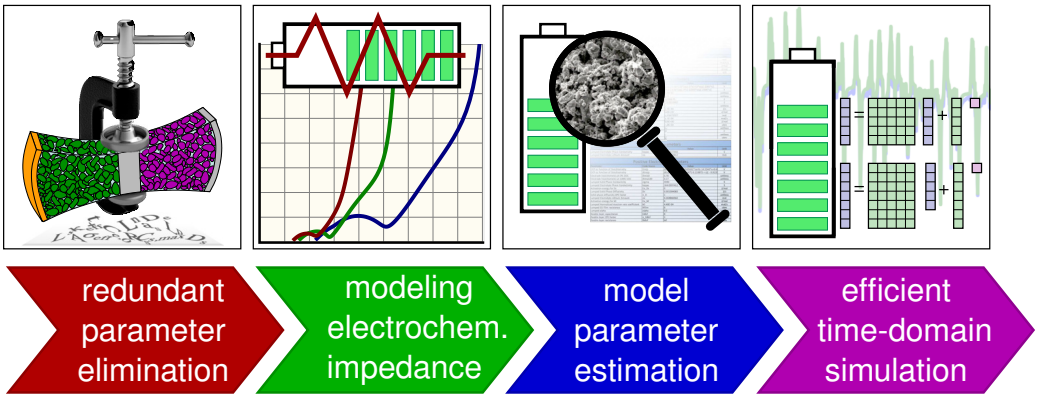
EFFICIENT TIME-DOMAIN SIMULATION

4.1: Introduction and context

- We have now eliminated redundant model parameters, created impedance models, and identified model parameter values.



- We now study creating reduced-order models (ROMs) for efficient time-domain computation of cell internal variables (and voltage).
- Must convert impedance models to state-space models of the form:

$$\boldsymbol{x}[k+1] = \boldsymbol{A}\boldsymbol{x}[k] + \boldsymbol{B}\boldsymbol{u}[k]$$

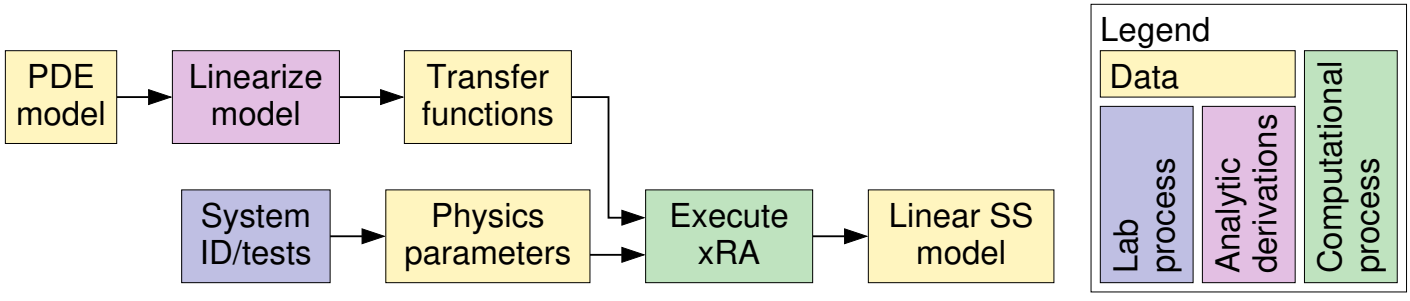
$$\boldsymbol{y}[k] = \boldsymbol{C}\boldsymbol{x}[k] + \boldsymbol{D}\boldsymbol{u}[k],$$

where the state vector of the system is represented by $\boldsymbol{x}[k] \in \mathbb{R}^{n\times 1}$, the input vector by $\boldsymbol{u}[k] \in \mathbb{R}^{m\times 1}$ and the output vector by $\boldsymbol{y}[k] \in \mathbb{R}^{q\times 1}$.

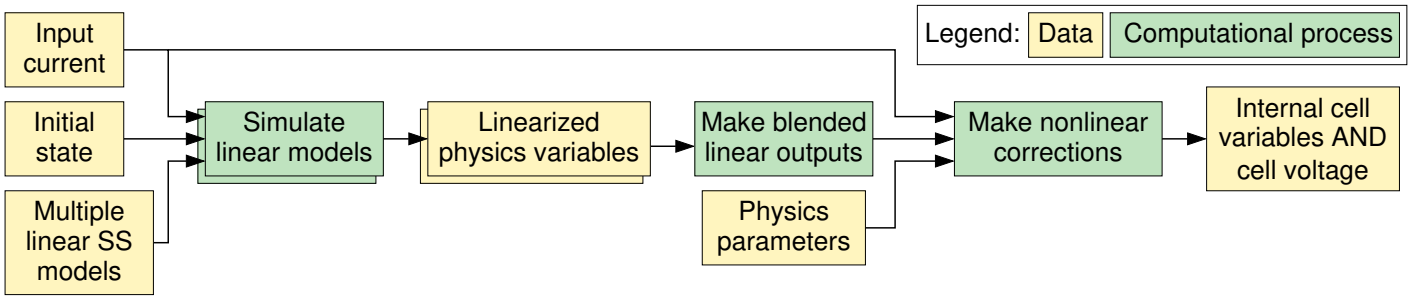
- Note, model matrices have dimensions:[1] $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times m}$, $\boldsymbol{C} \in \mathbb{R}^{q\times n}$, and $\boldsymbol{D} \in \mathbb{R}^{q\times m}$.

[1] By the end of this chapter, we augment the system with an integrator state, which has the overall effect of replacing n with $n+1$ in these dimensions.

- When we apply this type of model to describe lithium-ion batteries, $\boldsymbol{u}[k] = i_{\text{app}}[k]$ corresponds to the electrical current applied to the cell.
- $\boldsymbol{y}[k]$ corresponds to the set of electrochemical variables of interest.
- The state $\boldsymbol{x}[k]$ will be selected by the model-order reduction techniques that we will describe.
- In ECE5710, we studied one approach to creating ROMs via the "discrete-time realization algorithm" (DRA).
- The DRA has some limitations that have been overcome by other approaches (collectively, "xRAs"):
  - For accurate results, DRA needs long unit-pulse responses, which require a lot of memory to store and a lot of CPU to process.
  - Requires that TFs be stable or stabilized.
- More recent methods promise to reduce or remove these limitations.
- In this chapter, we look at the hybrid realization algorithm (HRA):
  - Converts $\boldsymbol{G}(s)$ to discrete-time frequency response $\boldsymbol{G}(e^{j\omega T})$, and
  - Then uses a subspace system-identification method to compute a high- but finite-order discrete-time ROM.
  - Finally, a standard balanced model-order reduction method is used to arrive at a final low-order discrete-time ROM.
- The HRA uses a very similar ROM-generation approach to the DRA:



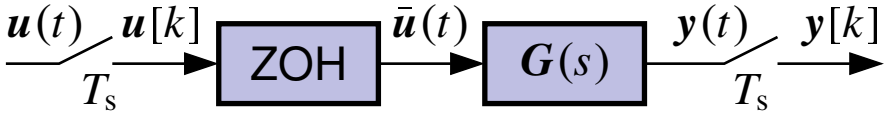
- The “xRA” block in the diagram can be either DRA or HRA.
- The models are then simulated to produce time-domain results:



- In this chapter, we first look at how to convert a frequency response from continuous-time to discrete-time.
- Then, we look at the HRA method and some frequency-response matches between the ideal impedance models and the ROMs.
- We then show how to apply this to simulate a cell in the time domain near a setpoint and over a wide operational range.
- We apply the ideas of this chapter to show how to simulate constant-voltage and constant-power profiles.
- Finally, we see how to simulate battery packs in the time domain.

4.2: Convert continuous-time to discrete-time frequency response

- Continuous-time transfer functions may be converted readily to continuous-time frequency responses via $\boldsymbol{G}(j\omega) = \boldsymbol{G}(s)|_{s=j\omega}$.
- It is less clear how to convert continuous-time frequency responses into discrete-time frequency responses needed by some xRAs.
- If transfer function is rational-polynomial in Laplace variable "s", then established methods may be used. Otherwise, need a new approach.
- The figure below shows components of a sampled-data system that are relevant to our discussion.



  - Computer produces discrete-time signal $\boldsymbol{u}[k]$,[2] which we assume originated as continuous-time $\boldsymbol{u}(t)$, sampled every T_s seconds.
  - $\boldsymbol{u}[k]$ is passed through digital-to-analog converter incorporating a zero-order hold (ZOH) circuit to produce continuous-time $\bar{\boldsymbol{u}}(t)$.
  - $\bar{\boldsymbol{u}}(t)$ is input to continuous-time system having transfer function $\boldsymbol{G}(s)$, producing continuous-time output signal $\boldsymbol{y}(t)$.
  - $\boldsymbol{y}(t)$ is sampled every T_s seconds to produce discrete-time $\boldsymbol{y}[k]$.
- We can combine continuous-time elements $\boldsymbol{G}(s)$ and ZOH using the known transfer function $(1-\exp(-sT_\mathrm{s}))/s$ of a ZOH:

$$\bar{\boldsymbol{G}}(s) = \frac{1-e^{-sT_\mathrm{s}}}{s}\boldsymbol{G}(s), \quad \text{so} \quad \bar{\boldsymbol{G}}(j\omega) = \frac{1-e^{-j\omega T_\mathrm{s}}}{j\omega}\boldsymbol{G}(j\omega). \tag{4.1}$$

- The sampled-data system in the figure comprises both discrete-time and continuous-time elements.

[2] Square brackets denote discrete-time signal and k is (integer) discrete-time sampling index; parenthesis denote a continuous-time signal. Therefore, $\boldsymbol{u}[k] = \boldsymbol{u}(kT_\mathrm{s})$.

  - It is customary to use z- or discrete-time Fourier transform to analyze discrete-time systems and to use Laplace- or continuous-time Fourier transforms to analyze continuous-time systems.
  - To analyze hybrid sampled-data systems using a common framework, we can use the "starred Laplace transform".[3]
- To do so, suppose that $\boldsymbol{u}[k] = \boldsymbol{u}(kT_\mathrm{s})$.
- A continuous-time representation of *sampled* signal is $\boldsymbol{u}(t)\delta_{T_\mathrm{s}}(t)$, where $\delta_{T_\mathrm{s}}(t)$ is the Dirac comb function having period T_s.
- Equivalent Laplace-domain representation of sampled signal is then:

$$\boldsymbol{U}^*(s) = \mathcal{L}\left\{\boldsymbol{u}(t)\delta_{T_\mathrm{s}}(t)\right\} = \sum_{k=0}^{\infty} \boldsymbol{u}[k]e^{-skT_\mathrm{s}},$$

- With this notation, we can write $\boldsymbol{Y}(s) = \bar{\boldsymbol{G}}(s)\boldsymbol{U}^*(s)$ and can also find that $\boldsymbol{Y}^*(s) = \bar{\boldsymbol{G}}^*(s)\boldsymbol{U}^*(s)$.
- Starred and standard Laplace transforms are related as:

$$\bar{\boldsymbol{G}}^*(s) = \frac{1}{T_\mathrm{s}} \sum_{k=-\infty}^{\infty} \bar{\boldsymbol{G}}\left(s + j\frac{2\pi}{T_\mathrm{s}}k\right) + \frac{\bar{\boldsymbol{g}}(0)}{2}. \tag{4.2}$$

  - The summand for $k = 0$ corresponds to the original frequency spectrum, which is copied into $\bar{\boldsymbol{G}}^*(s)$.
  - The summands for $k \neq 0$ comprise aliases of the original frequency spectrum that are introduced by the sampling operation.
- While the summation limits go from $-\infty$ to ∞ in principle, good approximations to $\bar{\boldsymbol{G}}^*(s)$ are achieved using much fewer terms since we don't expect much aliasing if we have sampled rapidly enough.

[3] E.I. Jury, "Analysis and synthesis of sampled-data control systems", *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 73(4), 332–346, 1954.

- The value for $\bar{\boldsymbol{g}}(0)$ can be found via the initial-value theorem as:

$$\bar{\boldsymbol{g}}(0) = \lim_{s\to\infty} s\bar{\boldsymbol{G}}(s) = \lim_{s\to\infty}(1-e^{-sT_\mathrm{s}})\boldsymbol{G}(s) = \lim_{\omega\to\infty}\boldsymbol{G}(j\omega).$$

  - This is the value of the "$\boldsymbol{D}$" matrix for both the continuous-time and discrete-time state-space models.

- Note that it can be subtracted from $\bar{\boldsymbol{G}}(j\omega)$ before converting from continuous-time to discrete-time, and then added back in later.

- Therefore, can assume $\bar{\boldsymbol{g}}(0) = 0$ in Eq. (4.2) without loss of generality.

- The starred Laplace transform can also be related to z-transform as $\boldsymbol{Y}(z) = \boldsymbol{Y}^*(s)\big|_{s=\ln(z)/T_\mathrm{s}}$. Therefore, we have

$$\boldsymbol{Y}(z) = \underbrace{\left[\frac{1}{T_\mathrm{s}}\sum_{k=-\infty}^{\infty}\bar{\boldsymbol{G}}\left(\frac{\ln(z)}{T_\mathrm{s}} + j\frac{2\pi}{T_\mathrm{s}}k\right)\right]}_{\boldsymbol{G}(z)}\boldsymbol{U}(z).$$

- The discrete-time frequency response that we desire to compute is $\boldsymbol{G}(e^{j\omega T_\mathrm{s}}) = \boldsymbol{G}(z)|_{z=\exp(j\omega T_\mathrm{s})}$. Therefore,

$$\boldsymbol{G}(e^{j\omega T_\mathrm{s}}) = \frac{1}{T_\mathrm{s}}\sum_{k=-\infty}^{\infty}\bar{\boldsymbol{G}}\left(j\omega + j\frac{2\pi}{T_\mathrm{s}}k\right). \tag{4.3}$$

- To be clear, $\boldsymbol{G}(e^{j\omega T_\mathrm{s}})$ on LHS is discrete-time frequency response of sampled-data system that we wish to approximate; $\bar{\boldsymbol{G}}(j\omega)$ on RHS is known continuous-time frequency response calculated by Eq. (4.1).

- Result is valid for frequencies up to Nyquist rate: $-\pi/T_\mathrm{s} < \omega < \pi/T_\mathrm{s}$.

Illustrating the frequency-response conversion method

- We need a method to implement Eq. (4.3). To summarize, we must:

1. Compute $\bar{\boldsymbol{G}}(s)$ from $\boldsymbol{G}(s)$ using Eq. (4.1a).
2. Compute $\boldsymbol{D} = \bar{\boldsymbol{g}}(0) = \lim_{s\to\infty} s\bar{\boldsymbol{G}}(s)$ and replace $\bar{\boldsymbol{G}}(s) := \bar{\boldsymbol{G}}(s) - \boldsymbol{D}$.
3. Find continuous-time frequency response $\bar{\boldsymbol{G}}(j\omega)$ using Eq. (4.1b).
4. For some $k_{\max}$, approximate discrete-time approximate frequency response $\widehat{\boldsymbol{G}}(e^{j\omega T_\mathrm{s}})$ at N output frequency points using Eq. (4.3),

$$\widehat{\boldsymbol{G}}(e^{j\omega T_\mathrm{s}}) = \frac{1}{T_\mathrm{s}} \sum_{k=-k_{\max}}^{k_{\max}} \bar{\boldsymbol{G}}(j\omega + j2\pi k/T_\mathrm{s}).$$

5. Add back high-frequency gain, replacing $\widehat{\boldsymbol{G}}(e^{j\omega T_\mathrm{s}}) := \widehat{\boldsymbol{G}}(e^{j\omega T_\mathrm{s}}) + \boldsymbol{D}$.

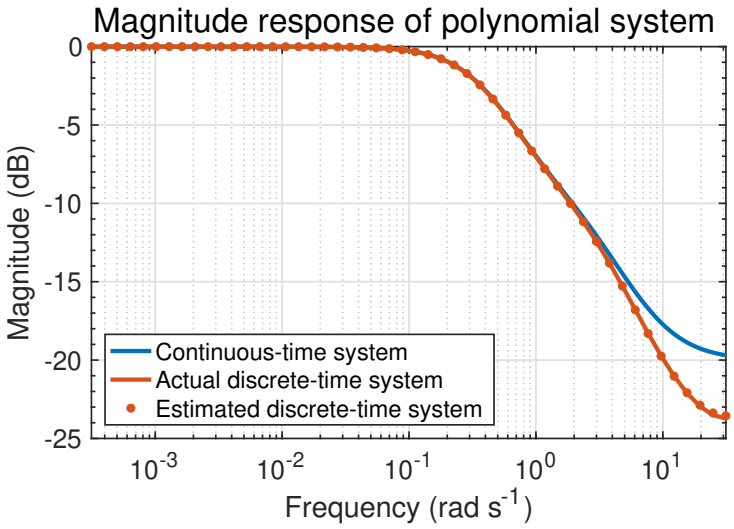
- New method can convert completely general continuous-time to discrete-time frequency responses, unlike conventional methods.
- But, we choose to demonstrate it first on a rational-polynomial TF, to validate its solution against one obtained from standard methods.
- In particular, we choose (for sampling period $T_\mathrm{s} = 0.1$ s):

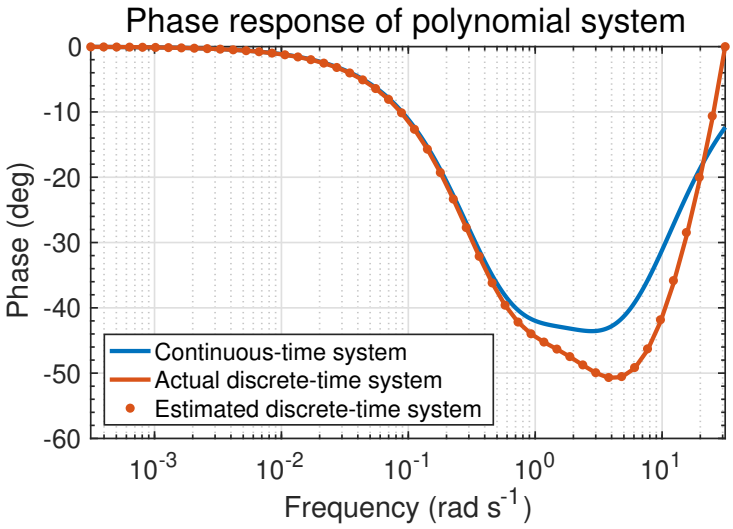
$$G(s) = \frac{0.1s^2 + s + 1}{s^2 + 3s + 1}, \quad \text{so} \quad G(j\omega) = \frac{1 + j\omega - 0.1\omega^2}{1 + 3j\omega - \omega^2}.$$

- For this simple case, we can compute the exact equivalent discrete-time system using MATLAB's `c2d.m` function as:

$$G(e^{j\omega T_\mathrm{s}}) = \frac{0.1e^{j0.2\omega} - 0.1088e^{j0.1\omega} + 0.0174}{e^{j0.2\omega} - 1.732e^{j0.1\omega} + 0.7408}.$$

- We choose $k_{\max} = 5$ and discrete-time frequency vector to comprise one point at $\omega = 0$ and 50 additional points spaced evenly on a logarithmic scale between 10^{-5} and 10^{0} times the Nyquist rate.
- The figures below compare magnitude and phase responses of the proposed method and exact result.

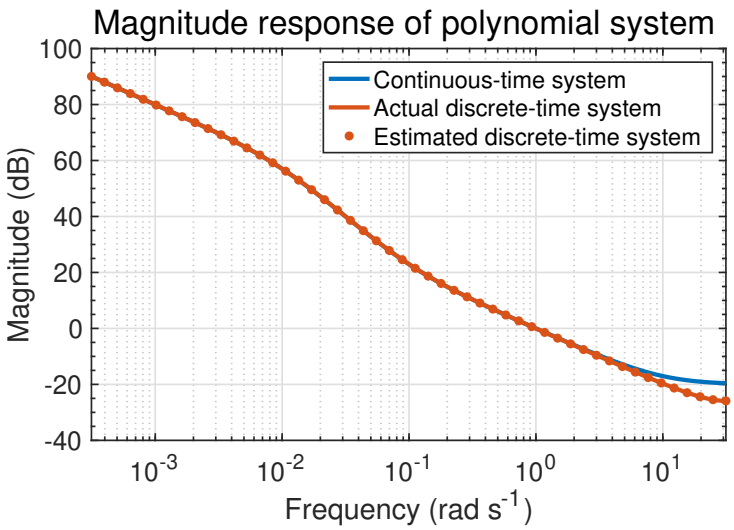


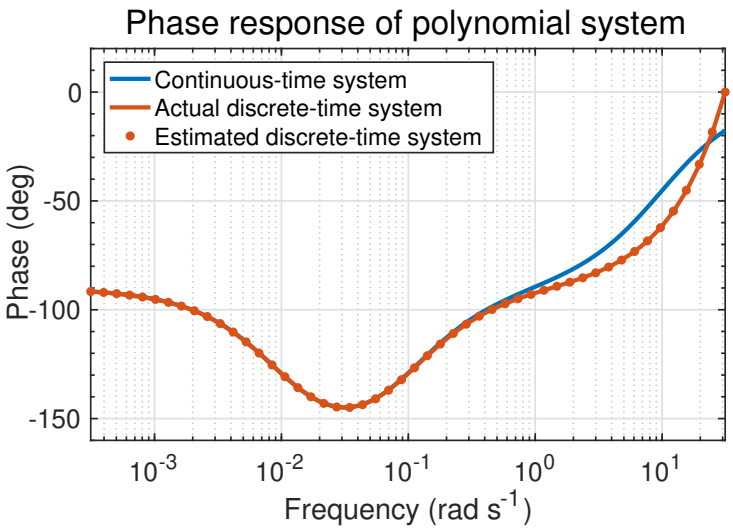


- A visually perfect match has been achieved between actual and estimated discrete-time system frequency responses, as hoped, showing that this method produces accurate results.
- We also note that the discrete-time and continuous-time frequency responses for this system are different from each other, as expected.
- Method also works for unstable systems. Consider ($T_s = 0.1$ s):

$$G(s) = \frac{s^2 + 10s + 1}{10s^2 + 0.1s}.$$

- The figures below compare magnitude and phase responses of the proposed method and exact result.





4.3: The hybrid realization algorithm (HRA)

- The HRA determines $\boldsymbol{A}$ from the discrete-time frequency responses.
- The derivation begins by taking z-transforms of state-space model:

$$z\boldsymbol{X}(z) = \boldsymbol{A}\boldsymbol{X}(z) + \boldsymbol{B}\boldsymbol{U}(z) \qquad [\text{assume } x(0) = 0]$$

$$\boldsymbol{Y}(z) = \boldsymbol{C}\boldsymbol{X}(z) + \boldsymbol{D}\boldsymbol{U}(z).$$

- Multiplying the output equation by z produces:

$$z\boldsymbol{Y}(z) = \boldsymbol{C}(z\boldsymbol{X}(z)) + \boldsymbol{D}(z\boldsymbol{U}(z)) = \boldsymbol{C}\boldsymbol{A}\boldsymbol{X}(z) + \boldsymbol{C}\boldsymbol{B}\boldsymbol{U}(z) + \boldsymbol{D}(z\boldsymbol{U}(z)).$$

- Repeating this process $i-1$ times (where i is termed the oversizing parameter) and combining the i modified output equations gives:

$$\begin{bmatrix} \boldsymbol{Y}(z) \\ z\boldsymbol{Y}(z) \\ \vdots \\ z^{i-1}\boldsymbol{Y}(z) \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{C}\boldsymbol{A} \\ \vdots \\ \boldsymbol{C}\boldsymbol{A}^{i-1} \end{bmatrix}}_{\mathcal{O}_i} \boldsymbol{X}(z) + \underbrace{\begin{bmatrix} \boldsymbol{D} & & & \\ \boldsymbol{C}\boldsymbol{B} & \boldsymbol{D} & & \\ \vdots & \ddots & \ddots & \\ \boldsymbol{C}\boldsymbol{A}^{i-2}\boldsymbol{B} & \cdots & \boldsymbol{C}\boldsymbol{B} & \boldsymbol{D} \end{bmatrix}}_{\mathcal{T}_i} \begin{bmatrix} \boldsymbol{U}(z) \\ z\boldsymbol{U}(z) \\ \vdots \\ z^{i-1}\boldsymbol{U}(z) \end{bmatrix},$$

  where $\mathcal{T}_i$ is block Toeplitz and $\mathcal{O}_i$ is extended observability matrix.

- Suppose $\boldsymbol{u}_j[k] = \mathbf{e}_j \exp(j\omega k T_\mathrm{s})$: sinusoid of freq. ω applied to jth input; $\mathbf{e}_j$ is basis vector j ($\mathbf{e}_1 = [\ 1,\ 0,\ \ldots,\ 0\ ]^T$, $\mathbf{e}_2 = [\ 0,\ 1,\ \ldots,\ 0\ ]^T$, ...).
  - Then, $\boldsymbol{U}_j(z) = \mathbf{e}_j z/(z - \exp(j\omega T_\mathrm{s}))$.
- Steady-state output is input–output frequency response for frequency ω and input j, which we write as $\boldsymbol{G}_j(e^{j\omega T_\mathrm{s}})$, multiplying the input:

$$\boldsymbol{Y}(z) = \boldsymbol{G}_j(e^{j\omega T_\mathrm{s}})\boldsymbol{U}_j(z).$$

- Similarly, steady-state version of state vector can be related to the input–state frequency response for frequency ω and input j:

$$\boldsymbol{X}(z) = \boldsymbol{X}_j(e^{j\omega T_\mathrm{s}})\boldsymbol{U}_j(z).$$

- Steady-state state vector is input–state frequency response $\boldsymbol{X}_j(e^{j\omega T_\mathrm{s}})$ multiplying the input. Overall, we can write:

$$\left[\begin{array}{c} \boldsymbol{G}_j(e^{j\omega T_\mathrm{s}})\boldsymbol{U}_j(z) \\ \hdashline z\boldsymbol{G}_j(e^{j\omega T_\mathrm{s}})\boldsymbol{U}_j(z) \\ \hdashline \vdots \\ \hdashline z^{i-1}\boldsymbol{G}_j(e^{j\omega T_\mathrm{s}})\boldsymbol{U}_j(z) \end{array}\right] = \boldsymbol{\mathcal{O}}_i \boldsymbol{X}_j(e^{j\omega T_\mathrm{s}})\boldsymbol{U}_j(z) + \boldsymbol{\mathcal{T}}_i \left[\begin{array}{c} \boldsymbol{U}_j(z) \\ \hdashline z\boldsymbol{U}_j(z) \\ \hdashline \vdots \\ \hdashline z^{i-1}\boldsymbol{U}_j(z) \end{array}\right].$$

- Notice that $z/(z-\exp(j\omega T_\mathrm{s}))$ is common to all terms, so cancels.
- Also note that we can write similar equations for j corresponding to all inputs, and when we do so we can stack the results horizontally.
- Defining $\boldsymbol{G}(e^{j\omega T_\mathrm{s}}) = \left[\begin{array}{c:c:c} \boldsymbol{G}_1(e^{j\omega T_\mathrm{s}}) & \cdots & \boldsymbol{G}_m(e^{j\omega T_\mathrm{s}}) \end{array}\right]$ and evaluating at $z = \exp(j\omega T_\mathrm{s})$ gives

$$\underbrace{\left[\begin{array}{c} \boldsymbol{G}(e^{j\omega T_\mathrm{s}}) \\ \hdashline e^{j\omega T_\mathrm{s}}\boldsymbol{G}(e^{j\omega T_\mathrm{s}}) \\ \hdashline \vdots \\ \hdashline e^{j\omega(i-1)T_\mathrm{s}}\boldsymbol{G}(e^{j\omega T_\mathrm{s}}) \end{array}\right]}_{\boldsymbol{\mathcal{G}}^\mathrm{c}(\omega)} = \boldsymbol{\mathcal{O}}_i \boldsymbol{\mathcal{X}}^\mathrm{c}(\omega) + \boldsymbol{\mathcal{T}}_i \underbrace{\left[\begin{array}{c} \boldsymbol{I}_m \\ \hdashline e^{j\omega T_\mathrm{s}}\boldsymbol{I}_m \\ \hdashline \vdots \\ \hdashline e^{j\omega(i-1)T_\mathrm{s}}\boldsymbol{I}_m \end{array}\right]}_{\boldsymbol{\mathcal{U}}^\mathrm{c}(\omega)},$$

  where $\boldsymbol{\mathcal{X}}^\mathrm{c}(\omega) = \left[\begin{array}{c:c:c:c} \boldsymbol{X}_1(e^{j\omega T_\mathrm{s}}) & \boldsymbol{X}_2(e^{j\omega T_\mathrm{s}}) & \cdots & \boldsymbol{X}_m(e^{j\omega T_\mathrm{s}}) \end{array}\right]$.
  - Superscript "c" denotes complex-valued matrices.
- We can further evaluate the terms of this equation at multiple frequency points $\omega_1 \ldots \omega_N$, giving:

$$\underbrace{\left[\begin{array}{c:c:c:c} \boldsymbol{\mathcal{G}}^\mathrm{c}(\omega_1) & \cdots & \boldsymbol{\mathcal{G}}^\mathrm{c}(\omega_N) \end{array}\right]}_{\boldsymbol{\mathcal{G}}^\mathrm{c}} = \boldsymbol{\mathcal{O}}_i \underbrace{\left[\begin{array}{c:c:c:c} \boldsymbol{\mathcal{X}}^\mathrm{c}(\omega_1) & \cdots & \boldsymbol{\mathcal{X}}^\mathrm{c}(\omega_N) \end{array}\right]}_{\boldsymbol{\mathcal{X}}^\mathrm{c}} + \boldsymbol{\mathcal{T}}_i \underbrace{\left[\begin{array}{c:c:c:c} \boldsymbol{\mathcal{U}}^\mathrm{c}(\omega_1) & \cdots & \boldsymbol{\mathcal{U}}^\mathrm{c}(\omega_N) \end{array}\right]}_{\boldsymbol{\mathcal{U}}^\mathrm{c}}.$$

- A final step yields matrices containing only real values:

$$\underbrace{\left[\text{real}(\mathcal{G}^{\text{c}}) \;\middle|\; \text{imag}(\mathcal{G}^{\text{c}})\right]}_{\mathcal{G}^{\text{r}}} = \mathcal{O}_i \underbrace{\left[\text{real}(\mathcal{X}^{\text{c}}) \;\middle|\; \text{imag}(\mathcal{X}^{\text{c}})\right]}_{\mathcal{X}^{\text{r}}} + \mathcal{T}_i \underbrace{\left[\text{real}(\mathcal{U}^{\text{c}}) \;\middle|\; \text{imag}(\mathcal{U}^{\text{c}})\right]}_{\mathcal{U}^{\text{r}}}. \tag{4.4}$$

  - We use values computed via method from Topic 4.2 to populate $\mathcal{G}^{\text{r}}$.
  - $\mathcal{U}^{\text{r}}$ can be computed from the chosen frequency vector $\{\omega_1 \ldots \omega_N\}$.
  - $\mathcal{X}^{\text{r}}$ is unknown but we will not require knowledge of its values.

- We then perform the LQ decomposition:

$$\underbrace{\left[\begin{array}{c} \mathcal{U}^{\text{r}} \\ \hdashline \mathcal{G}^{\text{r}} \end{array}\right]}_{\text{data matrix}} = \boldsymbol{L}\,\boldsymbol{Q} = \left[\begin{array}{c|c} \boldsymbol{L}_{11} & \boldsymbol{0} \\ \hdashline \boldsymbol{L}_{21} & \boldsymbol{L}_{22} \end{array}\right] \left[\begin{array}{c} \boldsymbol{Q}_1 \\ \hdashline \boldsymbol{Q}_2 \end{array}\right],$$

  where $\boldsymbol{L}$ has the same dimension as "data matrix", and where $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are orthonormal ($\boldsymbol{Q}_1\boldsymbol{Q}_1^T = \boldsymbol{I}$, $\boldsymbol{Q}_2\boldsymbol{Q}_2^T = \boldsymbol{I}$, $\boldsymbol{Q}_1\boldsymbol{Q}_2^T = \boldsymbol{0}$, $\boldsymbol{Q}_2\boldsymbol{Q}_1^T = \boldsymbol{0}$):

$$\boldsymbol{U}^{\text{r}} = \boldsymbol{L}_{11}\boldsymbol{Q}_1$$

$$\mathcal{G}^{\text{r}} = \boldsymbol{L}_{21}\boldsymbol{Q}_1 + \boldsymbol{L}_{22}\boldsymbol{Q}_2.$$

- Multiplying the second equation by $\boldsymbol{Q}_2^T$ on the right gives $\mathcal{G}^{\text{r}}\boldsymbol{Q}_2^T = \boldsymbol{L}_{22}$.
- Multiplying Eq. (4.4) on right by $\boldsymbol{Q}_2^T$ and substituting this result gives:

$$\mathcal{G}^{\text{r}}\boldsymbol{Q}_2^T = \boldsymbol{L}_{22} = \mathcal{O}_i\mathcal{X}^{\text{r}}\boldsymbol{Q}_2^T.$$

- The net effect of the LQ decomposition has been to project Eq. (4.4) onto a subspace orthogonal to the input $\mathcal{U}^{\text{r}}$, thus removing the effect of $\mathcal{U}^{\text{r}}$ from Eq. (4.4) and removing the need to know $\mathcal{T}_i$.
- Resulting equation states that $\boldsymbol{L}_{22}$ has the same column space as $\mathcal{O}_i$.

- To recover $\mathcal{O}_i$ (up to an unknown but unimportant similarity transformation) from $\boldsymbol{L}_{22}$, we perform the SVD:

$$\boldsymbol{L}_{22} = \left[\begin{array}{c|c} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{array}\right] \left[\begin{array}{c|c} \boldsymbol{\Sigma}_1 & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{\Sigma}_2 \end{array}\right] \left[\begin{array}{c} \boldsymbol{V}_1^T \\ \hline \boldsymbol{V}_2^T \end{array}\right].$$

- $\boldsymbol{L}_{22}$ can be well approximated using n_1 most significant singular values, those stored in $\boldsymbol{\Sigma}_1$ ($\boldsymbol{\Sigma}$ is partitioned so that $\boldsymbol{\Sigma}_2 \approx \boldsymbol{0}$).
- We estimate the extended observability matrix $\widehat{\mathcal{O}}_i = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1^{1/2}$.
- The shift property of the extended observability matrix allows computing $\boldsymbol{A}$ as the least-squares solution to:

$$\boldsymbol{A} = (\widehat{\mathcal{O}}_i^{\downarrow})^{\dagger} \widehat{\mathcal{O}}_i^{\uparrow},$$

  where $\dagger$ is the matrix pseudo-inverse operation and the $\uparrow$ and $\downarrow$ symbols denote shifting the indicated matrix up or down (deleting top or bottom block row), in this case resulting in

$$\widehat{\mathcal{O}}_i^{\downarrow} \approx \mathcal{O}_i^{\downarrow} = \left[\begin{array}{c|c|c|c|c} \boldsymbol{C}^T & (\boldsymbol{CA})^T & (\boldsymbol{CA}^2)^T & \cdots & (\boldsymbol{CA}^{i-2})^T \end{array}\right]^T,$$

$$\widehat{\mathcal{O}}_i^{\uparrow} \approx \mathcal{O}_i^{\uparrow} = \left[\begin{array}{c|c|c|c|c} (\boldsymbol{CA})^T & (\boldsymbol{CA}^2)^T & (\boldsymbol{CA}^3)^T & \cdots & (\boldsymbol{CA}^{i-1})^T \end{array}\right]^T.$$

- We find that the HRA is sensitive to model order n and produces much better models with large n than with small n.
- So, we begin with order $n_1 > n$ to obtain an initial state-space model that closely approximates the infinite-order frequency response.
- Then, we perform balanced model-order reduction (e.g., using `balred.m`) to obtain final state-space ROM of order n.
- We expect $\boldsymbol{A}$ to have real, stable poles, which is not guaranteed by the method as presented so far. So, we replace unstable poles in $\boldsymbol{A}$ by their reciprocals and complex poles by their magnitudes.

Summarizing the HRA

- To summarize the steps of the HRA:

1. Compute $\boldsymbol{G}(e^{j\omega T_\mathrm{s}})$ from $\boldsymbol{G}(j\omega)$ using the method of Topic 4.2.
2. Extend the frequency responses to form:

$$\boldsymbol{\mathcal{G}}^\mathrm{c}(\omega) = \left[\begin{array}{c} \boldsymbol{G}(e^{j\omega T_\mathrm{s}}) \\ \hdashline e^{j\omega T_\mathrm{s}}\boldsymbol{G}(e^{j\omega T_\mathrm{s}}) \\ \hdashline \vdots \\ \hdashline e^{j\omega(i-1)T_\mathrm{s}}\boldsymbol{G}(e^{j\omega T_\mathrm{s}}) \end{array}\right] \quad \text{and} \quad \boldsymbol{\mathcal{U}}^\mathrm{c}(\omega) = \left[\begin{array}{c} \boldsymbol{I}_m \\ \hdashline e^{j\omega T_\mathrm{s}}\boldsymbol{I}_m \\ \hdashline \vdots \\ \hdashline e^{j\omega(i-1)T_\mathrm{s}}\boldsymbol{I}_m \end{array}\right].$$

3. Form the complex data matrices:

$$\boldsymbol{\mathcal{G}}^\mathrm{c} = \left[\begin{array}{c|c|c} \boldsymbol{\mathcal{G}}^\mathrm{c}(\omega_1) & \cdots & \boldsymbol{\mathcal{G}}^\mathrm{c}(\omega_N) \end{array}\right] \text{ and } \boldsymbol{\mathcal{U}}^\mathrm{c} = \left[\begin{array}{c|c|c} \boldsymbol{\mathcal{U}}^\mathrm{c}(\omega_1) & \cdots & \boldsymbol{\mathcal{U}}^\mathrm{c}(\omega_N) \end{array}\right].$$

4. Form the data real data matrices:

$$\boldsymbol{\mathcal{G}}^\mathrm{r} = \left[\begin{array}{c|c} \mathrm{real}(\boldsymbol{\mathcal{G}}^\mathrm{c}) & \mathrm{imag}(\boldsymbol{\mathcal{G}}^\mathrm{c}) \end{array}\right] \quad \text{and} \quad \boldsymbol{\mathcal{U}}^\mathrm{r} = \left[\begin{array}{c|c} \mathrm{real}(\boldsymbol{\mathcal{U}}^\mathrm{c}) & \mathrm{imag}(\boldsymbol{\mathcal{U}}^\mathrm{c}) \end{array}\right].$$

5. Find $\boldsymbol{L}_{22}$ via the LQ decomposition:

$$\left[\begin{array}{c} \boldsymbol{\mathcal{U}}^\mathrm{r} \\ \hdashline \boldsymbol{\mathcal{G}}^\mathrm{r} \end{array}\right] = \boldsymbol{L}\,\boldsymbol{Q} = \left[\begin{array}{c|c} \boldsymbol{L}_{11} & \boldsymbol{0} \\ \hdashline \boldsymbol{L}_{21} & \boldsymbol{L}_{22} \end{array}\right]\left[\begin{array}{c} \boldsymbol{Q}_1 \\ \hdashline \boldsymbol{Q}_2 \end{array}\right].$$

6. Compute the SVD of $\boldsymbol{L}_{22}$ to give $\boldsymbol{U}_1$ and $\boldsymbol{\Sigma}_1$. Model order $n_1 > n$ is selected such that we accurately approximate the actual system.
7. Compute $\widehat{\mathcal{O}}_i = \boldsymbol{U}_1\boldsymbol{\Sigma}_1^{1/2}$.
8. Compute $\boldsymbol{A} = (\widehat{\mathcal{O}}_i^{\downarrow})^\dagger\widehat{\mathcal{O}}_i^{\uparrow}$.
9. Diagonalize $\boldsymbol{A}$ and replace unstable poles by their reciprocals and complex poles by their magnitudes.
10. Find $\boldsymbol{B}$ and $\boldsymbol{C}$ as described later ($\boldsymbol{D}$ is found analytically).
11. Perform balanced model-order reduction to obtain a discrete-time state-space model having desired order n.

4.4: Final form of A, B, C, and D

Solving for the state-space B matrix

- State-space models have multiple degrees of freedom for choosing some parameter values, after which the remaining values will be fixed.
- For simplicity of a final implementation, we choose to assign $\boldsymbol{B} = \mathbf{1}_{n\times 1}$ without loss of generality, since the $\boldsymbol{C}$ matrix will be computed to scale outputs properly for this choice of the discrete-time $\boldsymbol{B}$ matrix.

Solving for the state-space C matrix

- When finding $\boldsymbol{C}$, we desire to minimize the squares of errors between the actual and approximate frequency responses while forcing the dc gain of the approximated system to match that of the actual system.
- So, we form constrained-optimization scalar cost function to minimize:

$$J = \sum_{k=1}^{N} \|\boldsymbol{G}(\omega_k) - \boldsymbol{H}(\omega_k)\|_2^2 + \boldsymbol{\lambda}^T \left(\boldsymbol{G}(0) - \boldsymbol{H}(0)\right),$$

  where $\boldsymbol{G}$ is actual frequency response and $\boldsymbol{H}$ is ROM approximation.
- For notational simplicity, we remove the $\boldsymbol{D}$ term (known analytically) and write $\boldsymbol{G}(\omega_k) = \boldsymbol{G}(e^{j\omega_k T_s}) - \boldsymbol{D}$ and $\boldsymbol{M}(\omega_k) = (e^{j\omega_k T_s}\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}$.
- Then, $\boldsymbol{H}(\omega_k) = \boldsymbol{C}\boldsymbol{M}(\omega_k)$ and,

$$\begin{aligned} J &= \left\{ \sum_{k=1}^{N} \left[\boldsymbol{G}(\omega_k) - \boldsymbol{H}(\omega_k)\right]^* \left[\boldsymbol{G}(\omega_k) - \boldsymbol{H}(\omega_k)\right] \right\} + \boldsymbol{\lambda}^T \left(\boldsymbol{G}(0) - \boldsymbol{H}(0)\right) \\ &= \left\{ \sum_{k=1}^{N} \mathrm{Tr}\left[\boldsymbol{G}(\omega_k)\boldsymbol{G}^*(\omega_k)\right] - \mathrm{Tr}\left[\boldsymbol{G}(\omega_k)\boldsymbol{M}^*(\omega_k)\boldsymbol{C}^T\right] - \mathrm{Tr}\left[\boldsymbol{C}\boldsymbol{M}(\omega_k)\boldsymbol{G}^*(\omega_k)\right] \right. \\ &\quad \left. + \mathrm{Tr}\left[\boldsymbol{C}\boldsymbol{M}(\omega_k)\boldsymbol{M}^*(\omega_k)\boldsymbol{C}^T\right] \right\} + \mathrm{Tr}\left[\left(\boldsymbol{G}(0) - \boldsymbol{C}\boldsymbol{M}(0)\right)\boldsymbol{\lambda}^T\right]. \end{aligned}$$

- Taking derivatives of J with respect to $\boldsymbol{\lambda}$ and $\boldsymbol{C}$ gives:

$$\frac{\partial J}{\partial \boldsymbol{\lambda}} = \boldsymbol{G}(0) - \boldsymbol{C}\boldsymbol{M}(0) = 0$$

$$\frac{\partial J}{\partial \boldsymbol{C}} = \underbrace{-2\sum_{k=1}^{N} \mathbb{R}\left[\boldsymbol{G}(\omega_k)\boldsymbol{M}^*(\omega_k)\right]}_{\boldsymbol{M}_1} + \underbrace{2\boldsymbol{C}\sum_{k=1}^{N} \mathbb{R}\left[\boldsymbol{M}(\omega_k)\boldsymbol{M}^*(\omega_k)\right]}_{\boldsymbol{M}_2} - \boldsymbol{\lambda}\boldsymbol{M}^T(0) = 0.$$

- Transposing and combining into one matrix equation, we have

$$\left[\begin{array}{c|c} 2\boldsymbol{M}_2^T & -\boldsymbol{M}(0) \\ \hline \boldsymbol{M}^T(0) & \boldsymbol{0}^T \end{array}\right] \left[\begin{array}{c} \boldsymbol{C}^T \\ \hline \boldsymbol{\lambda}^T \end{array}\right] = \left[\begin{array}{c} 2\boldsymbol{M}_1^T \\ \hline \boldsymbol{G}^T(0) \end{array}\right].$$

- We solve this for $\boldsymbol{C}$ (and $\boldsymbol{\lambda}$, which is not used) using least squares.

Solving for the state-space $\boldsymbol{D}$ matrix

- The final model unknown is $\boldsymbol{D}$, which describes the instantaneous change in output $\boldsymbol{y}[k]$ when the system is excited with input $\boldsymbol{u}[k]$.
- For a PBM, the $\boldsymbol{D}$ matrix can be found in closed-form since the electrochemical transfer functions are analytical.
- That is, we simply compute:

$$\boldsymbol{D} = \left[\lim_{s\to\infty} \boldsymbol{G}_1(s),\ \lim_{s\to\infty} \boldsymbol{G}_2(s),\ \ldots,\ \lim_{s\to\infty} \boldsymbol{G}_q(s)\right]^T.$$

Handling integration dynamics

- Remember that the DRA from ECE5710 could not directly handle transfer functions having integration dynamics.
- In principle, the HRA *should* be able to do so; even so we encounter better numerics if we follow the approach taken in ECE5710:
  - Compute the integrator residue as $\mathbf{res}^* = \lim_{s\to 0} s\boldsymbol{G}(s)$,

- Remove the integrator by defining $[\boldsymbol{G}(s)]^* = \boldsymbol{G}(s) - \mathbf{res}^*/s$.

- We then invoke the DRA or HRA to find a discrete-time state-space model of order n for integrator-removed $[\boldsymbol{G}(s)]^*$, where:

$$\boldsymbol{A} = \text{diag}\,(a_1, a_2, \ldots, a_n)\,,$$

and $|a_i| < 1$ for stability, $a_i \in \mathbb{R}$ for a nonoscillatory system and $a_1 < a_2 < \ldots < a_n$.

- Finally, we augment the identified model with the integration dynamics that were removed previously. That is, we write:

$$\mathfrak{A} = \left[\begin{array}{c|c} \boldsymbol{A} & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right], \quad \mathfrak{B} = \left[\begin{array}{c} \boldsymbol{B} \\ \hline 1 \end{array}\right], \quad \mathfrak{C} = \left[\begin{array}{c|c} \boldsymbol{C} & \mathbf{res}^* \end{array}\right],$$

and $\mathfrak{D} = \boldsymbol{D}$ is unchanged. The augmented model is of order $n+1$.[4]

- Using this augmented system, we can write:

$$\underbrace{\left[\begin{array}{c} \boldsymbol{x}[k+1] \\ \hline x_0[k+1] \end{array}\right]}_{\mathfrak{x}[k+1]} = \underbrace{\left[\begin{array}{c|c} \boldsymbol{A} & \mathbf{0} \\ \hline \mathbf{0}^T & 1 \end{array}\right]}_{\mathfrak{A}} \underbrace{\left[\begin{array}{c} \boldsymbol{x}[k] \\ \hline x_0[k] \end{array}\right]}_{\mathfrak{x}[k]} + \underbrace{\left[\begin{array}{c} \boldsymbol{B} \\ \hline 1 \end{array}\right]}_{\mathfrak{B}} u[k]$$

$$\boldsymbol{y}[k] = \underbrace{\left[\begin{array}{c|c} \boldsymbol{C} & \mathbf{res}^* \end{array}\right]}_{\mathfrak{C}} \underbrace{\left[\begin{array}{c} \boldsymbol{x}[k] \\ \hline x_0[k] \end{array}\right]}_{\mathfrak{x}[k]} + \mathfrak{D}u[k].$$

or

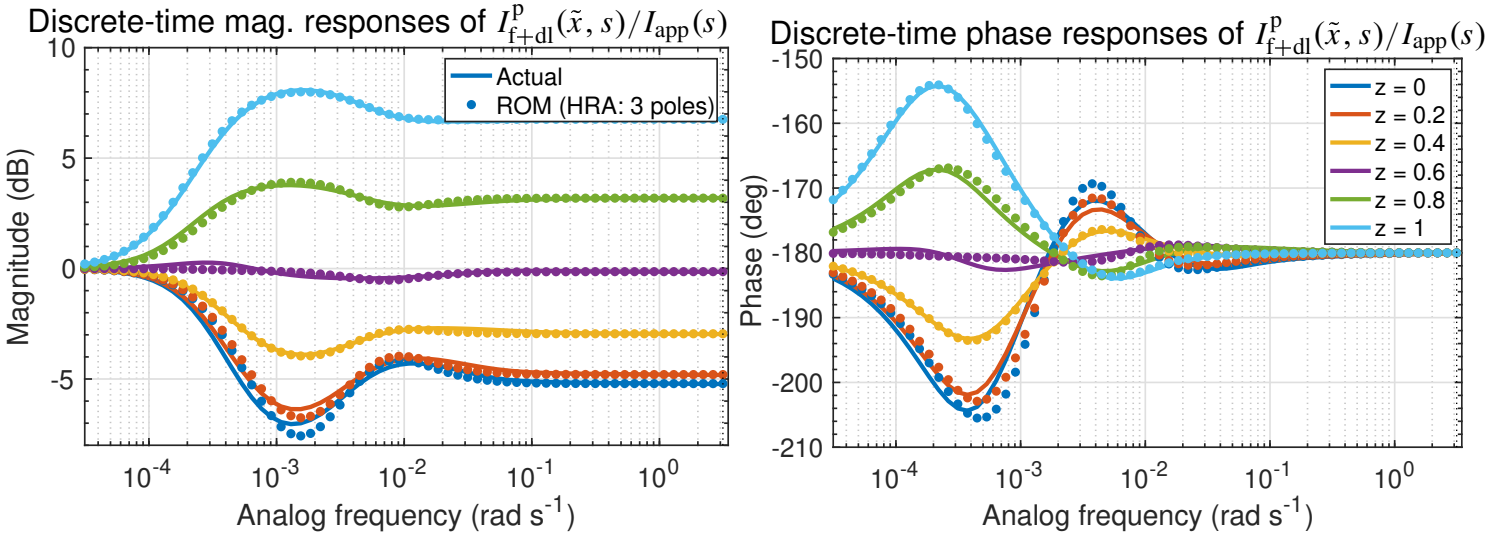
$$\mathfrak{x}[k+1] = \mathfrak{A}\mathfrak{x}[k] + \mathfrak{B}u[k]$$

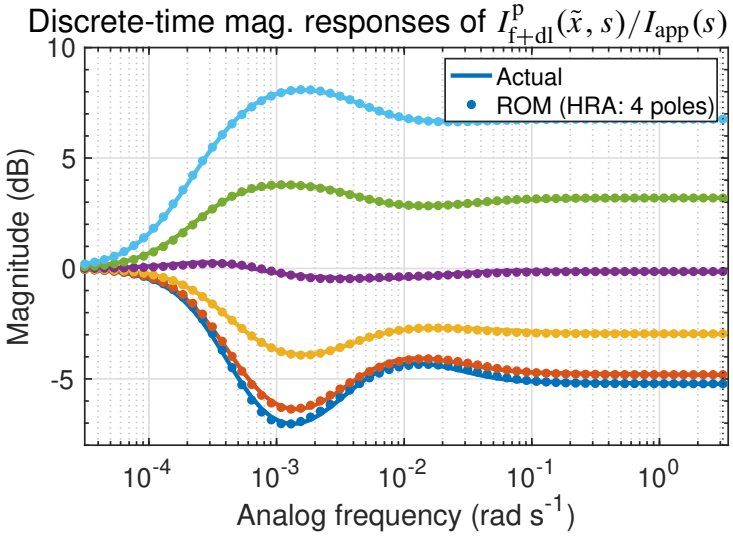
$$\boldsymbol{y}[k] = \mathfrak{C}\mathfrak{x}[k] + \mathfrak{D}u[k].$$

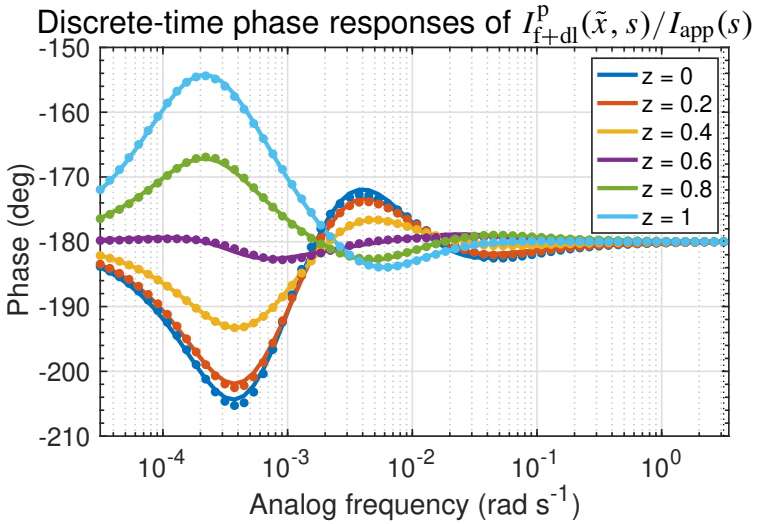
[4] Removing integration dynamics and then augmenting the final model with the integration dynamics is not a problem since integrator residues are given in App. 2.e.

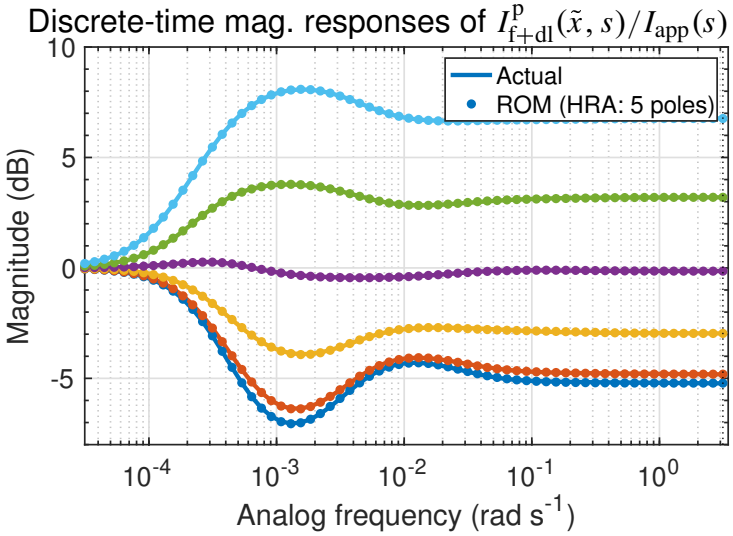
Sample HRA results

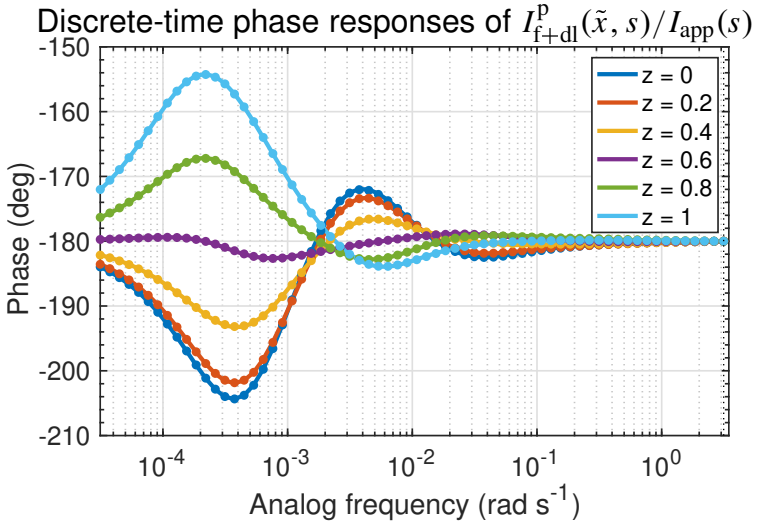
- We briefly examine some summary HRA-method results.
  - We seek three discrete-time ROMs having 3. . . 5 states to match $I^{\mathrm{p}}_{\mathrm{f+dl}}(z,s)/I_{\mathrm{app}}(s)$ over $(z = 0 : 0.2 : 1)$ in the electrode.
  - The sampling period was chosen as $T_{\mathrm{s}} = 1$.
- The method was tuned by hand; better tuning might be possible, but these results are representative of expected performance.
  - The oversizing parameter was chosen to be $i = 6$.
  - The method used $n_1 = 40$ initial poles.
  - The discrete-time frequency vector for building $\boldsymbol{G}(z)$ was chosen to comprise one point at $\omega = 0$ and 79 additional points spaced evenly on a logarithmic scale between 10^{-5} and 10^{0} times the Nyquist rate.
  - The value of $k_{\max} = 5$ was chosen to estimate the discrete-time frequency response from the continuous-time infinite-order TFs.
- Actual and HRA-ROM discrete-time frequency-responses are:











- The HRA average run time was 7.5 ms (much, much, much faster than the DRA).

4.5: Simulating a (single) cell in the time domain, near a setpoint

- Here, we wish to use a single xRA-produced model to simulate cell performance near a specific SOC and temperature setpoint.
- The xRAs produce state-space matrices to compute linearized variables $\boldsymbol{y}[k]$ as (where $\boldsymbol{x}[0] = 0$):

$$\mathfrak{X}[k+1] = \mathfrak{A}\mathfrak{X}[k] + \mathfrak{B}i_{\text{app}}[k]$$

$$\boldsymbol{y}[k] = \mathfrak{C}\mathfrak{X}[k] + \mathfrak{D}i_{\text{app}}[k].$$

- This can be implemented very simply in a "`for`" loop in MATLAB.
- Suppose that we have the model $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, and $\mathfrak{D}$ matrices, as well as a profile of current versus time stored in "`ik`".
- Then, we can simulate the state update and output update:

```
x = zeros(n+1,1); % initialize x[0] to the correct size
for k = 0:length(ik)-1, % MATLAB indexing: ik(1) = iapp[0]
  y = C*x + D*ik(k+1);  % The linearized outputs at this step, y[k]
  x = A*x + B*ik(k+1);  % The state for next time step, x[k+1]
end
```

- Of course, we will want to do something with output "`y`", which has linearized predictions of the internal variables chosen to simulate.
- To compute accurate predictions of the nonlinear variables we desire to model, we will need to apply nonlinear corrections.
- Also, to predict cell voltage, we will need to combine variables in a nonlinear way. Here, we see how to do both.

Update cell state of charge $z[k]$

- First, we look at how to update electrode and cell states of charge.

- If the cell has no double layer, then electrode state of charge (i.e., average solid stoichiometry) can be computed as

$$\theta^{\mathrm{r}}_{\mathrm{s,avg}}[k] = \theta^{\mathrm{r}}_{\mathrm{s},0} + \tilde{\theta}^{\mathrm{res0,r}}_{\mathrm{ss}} x_0[k], \tag{4.5}$$

  where $\theta^{\mathrm{r}}_{\mathrm{s},0}$ is the initial stoichiometry, $x_0[k]$ represents the integrator state, and $\tilde{\theta}^{\mathrm{res0,r}}_{\mathrm{ss}}$ is the integrator residue for the $\widetilde{\Theta}^{\mathrm{r}}_{\mathrm{s}}(z,s)/I_{\mathrm{app}}(s)$ TF.
  - Initial stoichiometry is determined from cell SOC as:

$$\theta^{\mathrm{r}}_{\mathrm{s},0} = \theta^{\mathrm{r}}_{0} + z_0\left(\theta^{\mathrm{r}}_{100} - \theta^{\mathrm{r}}_{0}\right). \tag{4.6}$$

  - Integrator residues are the following constants:

$$\tilde{\theta}^{\mathrm{res0,n}}_{\mathrm{ss}} = \frac{-\left|\theta^{\mathrm{n}}_{100} - \theta^{\mathrm{n}}_{0}\right|}{3600Q} \quad \text{and} \quad \tilde{\theta}^{\mathrm{res0,p}}_{\mathrm{ss}} = \frac{\left|\theta^{\mathrm{p}}_{100} - \theta^{\mathrm{p}}_{0}\right|}{3600Q}.$$

- If the cell has a double layer, integrator gains are a function of cell SOC; so, for most accurate results we must use a different approach.
- Recall the following two terms to condense notation,

$$\left[U^{\mathrm{r}}_{\mathrm{ocp}}\right]' = \left.\frac{\partial U^{\mathrm{r}}_{\mathrm{ocp}}(\theta_{\mathrm{s}})}{\partial \theta_{\mathrm{s}}}\right|_{\theta^{\mathrm{r}}_{\mathrm{s},0}}$$

$$\bar{C}^{\mathrm{r}}_{\mathrm{dl,eff}} = \left(\bar{C}^{\mathrm{r}}_{\mathrm{dl}}\right)^{2-n^{\mathrm{r}}_{\mathrm{dl}}} \left(\bar{\omega}^{\mathrm{r}}_{\mathrm{dl}}\right)^{n^{\mathrm{r}}_{\mathrm{dl}}-1}.$$

  where $\bar{C}^{\mathrm{r}}_{\mathrm{dl,eff}}$ is the CPE modified at low frequencies (cf. Topic 2.3).
- The integrator residues of $\widetilde{\Theta}^{\mathrm{r}}_{\mathrm{s}}(z,s)/I_{\mathrm{app}}(s)$ are:

$$\tilde{\theta}^{\mathrm{res0,n}}_{\mathrm{ss}} = \frac{-\left|\theta^{\mathrm{n}}_{100} - \theta^{\mathrm{n}}_{0}\right|}{3600Q - \bar{C}^{\mathrm{n}}_{\mathrm{dl,eff}}\left|\theta^{\mathrm{n}}_{100} - \theta^{\mathrm{n}}_{0}\right|\left[U^{\mathrm{n}}_{\mathrm{ocp}}\right]'}$$

$$\tilde{\theta}^{\mathrm{res0,p}}_{\mathrm{ss}} = \frac{\left|\theta^{\mathrm{p}}_{100} - \theta^{\mathrm{p}}_{0}\right|}{3600Q - \bar{C}^{\mathrm{p}}_{\mathrm{dl,eff}}\left|\theta^{\mathrm{p}}_{100} - \theta^{\mathrm{p}}_{0}\right|\left[U^{\mathrm{p}}_{\mathrm{ocp}}\right]'}.$$

- Since $\left[U^{\mathrm{r}}_{\mathrm{ocp}}\right]'$ is a function of electrode SOC, we need to update electrode SOC using time-varying gains.

```
thetaavgn = thetas0n; % initialize stoichiometry in negative
thetaavgp = thetas0p; % initialize stoichiometry in positive
dQn = abs(theta100n - theta0n);
dQp = abs(theta100p - theta0p);

% Start main simulation loop
for k = 0:length(ik)-1, % MATLAB indexing: ik(1) = ik[0]
  res0n = -dQn/(F*Q-Cdleffn*dQn*dUocpn(thetaavgn));
  res0p =  dQp/(F*Q-Cdleffp*dQp*dUocpp(thetaavgp));
  thetaavgn = thetaavgn + res0n*ik(k+1);
  thetaavgp = thetaavgp + res0p*ik(k+1);
end
```

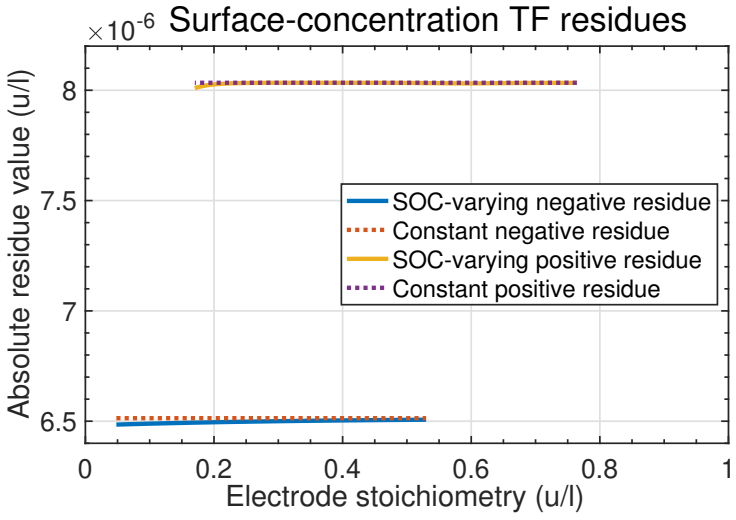
- Next, cell-level SOC can now be predicted as

$$z\,[k] = \frac{\theta^{\mathrm{n}}_{\mathrm{s,avg}}\,[k] - \theta^{\mathrm{n}}_{0}}{\theta^{\mathrm{n}}_{100} - \theta^{\mathrm{n}}_{0}} \qquad \text{or} \qquad z\,[k] = \frac{\theta^{\mathrm{p}}_{\mathrm{s,avg}}\,[k] - \theta^{\mathrm{p}}_{0}}{\theta^{\mathrm{p}}_{100} - \theta^{\mathrm{p}}_{0}}.$$

Or, maybe not

- The above code essentially replaces a single integrator with two—one for each electrode—making state estimation a challenge (Ch. 5).
  - Also, the two equations for cell-level SOC may give different answers; which $z[k]$ computation to use?

- How important is it to use the SOC-varying residue instead of a constant residue?
- Figure plots residue magnitudes for a modified Doyle cell with $\bar{C}^{\mathrm{n}}_{\mathrm{dl,eff}} = 200\,\mathrm{F}$ and $\bar{C}^{\mathrm{p}}_{\mathrm{dl,eff}} = 50\,\mathrm{F}$.



- Differences are noticeable, so SOC-varying residues will give better accuracy in open-loop predictions. But the differences are small:

  - $3600Q$ for many cells is a large number, frequently $100\,000$ or more;
  - OCP changes are on the order of 1 V for the full SOC range so $\left|[U^{\mathrm{r}}_{\mathrm{ocp}}]'\right|$ is on the order of 1;
  - $\left|\theta^{\mathrm{r}}_{100} - \theta^{\mathrm{r}}_{0}\right| < 1$ and $\bar{C}^{\mathrm{r}}_{\mathrm{dl,eff}}$ may be on the order of a few hundred farads (tends to be larger with larger-capacity cells);
  - Altogether, $\bar{C}^{\mathrm{r}}_{\mathrm{dl,eff}}\left|\theta^{\mathrm{r}}_{100} - \theta^{\mathrm{r}}_{0}\right|[U^{\mathrm{r}}_{\mathrm{ocp}}]'$ has magnitude on the order of a few hundred, at most;
  - The denominator term in $\tilde{\theta}^{\mathrm{res0,r}}_{\mathrm{ss}}$ is dominated by $3600Q$.

- When using voltage feedback to perform state estimation, filter adaptivity is sufficient to overcome constant-residue modeling errors.
  - So, in Ch. 5, we will use the simpler constant residues.

Nonlinear corrections

- Simulating the state-space model produces linear outputs $\boldsymbol{y}[k]$, whose elements correspond to linearized electrochemical variables.
- Many of these linearized predictions can be improved by computing "nonlinear corrections." cf. App. 4.b for a summary.

Cell voltage

- Cell voltage is computed by combining different variables:

$$\begin{aligned} v_{\mathrm{cell}}[k] = &\left(\eta^{\mathrm{p}}[3,k] - \eta^{\mathrm{n}}[0,k]\right) + \left(\phi^{\mathrm{p}}_{\mathrm{e}}[3,k] - \phi^{\mathrm{n}}_{\mathrm{e}}[0,k]\right) \\ &+ \left(U^{\mathrm{p}}_{\mathrm{ocp}}(\theta^{\mathrm{p}}_{\mathrm{ss}}[3,k]) - U^{\mathrm{n}}_{\mathrm{ocp}}(\theta^{\mathrm{n}}_{\mathrm{ss}}[0,k])\right) \\ &+ \left(\bar{R}^{\mathrm{p}}_{\mathrm{f}} i^{\mathrm{p}}_{\mathrm{f+dl}}[3,k] - \bar{R}^{\mathrm{n}}_{\mathrm{f}} i^{\mathrm{n}}_{\mathrm{f+dl}}[0,k]\right). \end{aligned}$$

- If charge-transfer coef. $\alpha = 0.5$, overpotentials η^{r} are computed as:

$$\eta^{\mathrm{r}}[\tilde{x},k] = \frac{2RT}{F}\mathrm{asinh}\left(\frac{i_{\mathrm{f}}^{\mathrm{r}}[\tilde{x},k]}{2i_0^{\mathrm{r}}}\right),$$

  where $i_0^{\mathrm{r}} = \bar{k}_0^{\mathrm{r}}\sqrt{\theta_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k]\left(1-\theta_{\mathrm{ss}}^{\mathrm{r}}[\tilde{x},k]\right)\theta_{\mathrm{ss}}^{\mathrm{r}}[\tilde{x},k]}$.

- The electrolyte-potential difference across the cell is:

$$\phi_{\mathrm{e}}^{\mathrm{p}}[3,k] - \phi_{\mathrm{e}}^{\mathrm{n}}[0,k] = \tilde{\phi}_{\mathrm{e}}^{\mathrm{p}}[3,k],$$

  where $\tilde{\phi}_{\mathrm{e}}^{\mathrm{p}}[3,k]$ can be a direct linear output from the ROM.

- With all previous definitions, we can finally write cell voltage as:

$$v_{\mathrm{cell}}[k] = \left(\eta^{\mathrm{p}}[3,k] - \eta^{\mathrm{n}}[0,k]\right) + \tilde{\phi}_{\mathrm{e}}^{\mathrm{p}}[3,k] + \left(U_{\mathrm{ocp}}^{\mathrm{p}}(\theta_{\mathrm{ss}}^{\mathrm{p}}[3,k]) - U_{\mathrm{ocp}}^{\mathrm{n}}(\theta_{\mathrm{ss}}^{\mathrm{n}}[0,k])\right) + \left(\bar{R}_{\mathrm{f}}^{\mathrm{p}} i_{\mathrm{f+dl}}^{\mathrm{p}}[3,k] - \bar{R}_{\mathrm{f}}^{\mathrm{n}} i_{\mathrm{f+dl}}^{\mathrm{n}}[0,k]\right), \tag{4.7}$$

  where $\theta_{\mathrm{ss}}^{\mathrm{p}}[3,k]$ and $\theta_{\mathrm{ss}}^{\mathrm{n}}[0,k]$ are evaluated in Eq. (4.10).

SUMMARY: The table below lists the equations that address nonlinear corrections for each variable; the last column identifies all variables required to compute the complete set of nonlinear corrections.

Variable	Definition	Correction	Needed for $v_{\mathrm{cell}}[k]$
$i_{\mathrm{f+dl}}^{\mathrm{r}}[\tilde{x},k]$	—	—	$i_{\mathrm{f+dl}}^{\mathrm{n}}[0,k]$, $i_{\mathrm{f+dl}}^{\mathrm{p}}[3,k]$
$i_{\mathrm{f}}^{\mathrm{r}}[\tilde{x},k]$	—	—	$i_{\mathrm{f}}^{\mathrm{n}}[0,k]$, $i_{\mathrm{f}}^{\mathrm{p}}[3,k]$
$i_{\mathrm{dl}}^{\mathrm{r}}[\tilde{x},k]$	—	—	—
$\theta_{\mathrm{ss}}^{\mathrm{r}}[\tilde{x},t]$	$\tilde{\theta}_{\mathrm{ss}}^{\mathrm{r}}[\tilde{x},k] + \theta_{\mathrm{s},0}$	Eq. (4.10)	$\tilde{\theta}_{\mathrm{ss}}^{\mathrm{n}}[0,k]$, $\tilde{\theta}_{\mathrm{ss}}^{\mathrm{p}}[3,k]$
$\phi_{\mathrm{s\text{-}e}}^{\mathrm{r}}[\tilde{x},k]$	$\tilde{\phi}_{\mathrm{s\text{-}e}}^{\mathrm{r}}[\tilde{x},k] + U_{\mathrm{ocp}}^{\mathrm{r}}(\theta_{\mathrm{s},0}^{\mathrm{r}})$	Eq. (4.11)	$[\tilde{\phi}_{\mathrm{s\text{-}e}}^{\mathrm{n}}[0,k]]^*$
$\phi_{\mathrm{s}}^{\mathrm{n}}[\tilde{x},k]$	$\tilde{\phi}_{\mathrm{s}}^{\mathrm{n}}[\tilde{x},k] + 0$	Eq. (4.12)	—
$\phi_{\mathrm{s}}^{\mathrm{p}}[\tilde{x},k]$	$\tilde{\phi}_{\mathrm{s}}^{\mathrm{p}}[\tilde{x},k] + v_{\mathrm{cell}}[k]$	Eq. (4.13)	—
$\phi_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k]$	$\tilde{\phi}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] + \phi_{\mathrm{e}}^{\mathrm{r}}[0,k]$	Eq. (4.14)	$\tilde{\phi}_{\mathrm{e}}^{\mathrm{p}}[3,k]$
$\theta_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k]$	$\tilde{\theta}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] + 1$	Eq. (4.15)	$\tilde{\theta}_{\mathrm{e}}^{\mathrm{n}}[0,k]$, $\tilde{\theta}_{\mathrm{e}}^{\mathrm{p}}[3,k]$
$v_{\mathrm{cell}}[k]$	$\phi_{\mathrm{s}}^{\mathrm{p}}[3,k] - \phi_{\mathrm{s}}^{\mathrm{n}}[0,k]$	Eq. (4.7)	—

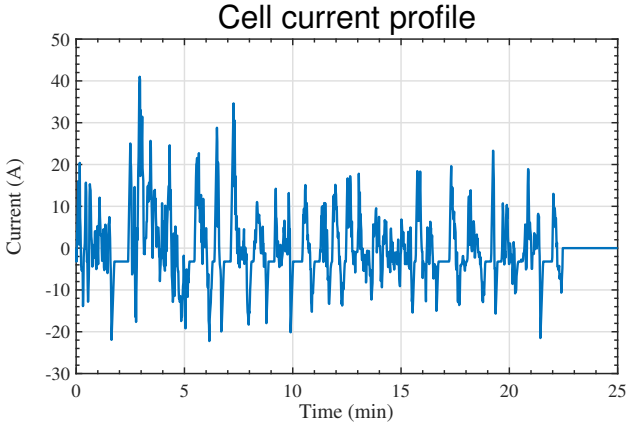
4.6: Simulation results near a ROM setpoint

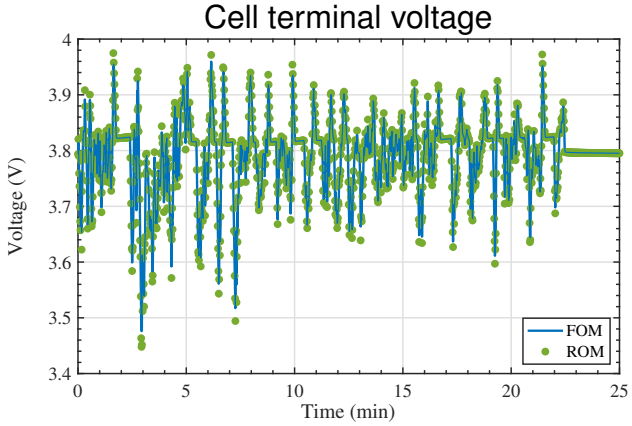
- This section presents results from a simulation to demonstrate the fidelity of the ROM with respect to the FOM.
- FOM was simulated in COMSOL; ROM had 8 dynamic states.
- The simulation implemented a single charge-neutral UDDS cycle, initialized at rest at 60 % SOC.

RMS errors for each variable

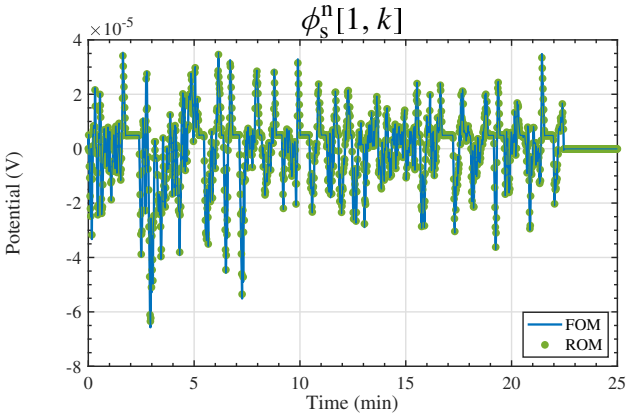
$\tilde{x}$	$i_{\mathrm{f+dl}}$ [A]	i_{f} [A]	i_{dl} [A]	θ_{ss} [u/l]	θ_{e} [u/l]	ϕ_{s} [mV]	ϕ_{e} [mV]	$\phi_{\mathrm{s\text{-}e}}$ [mV]
0	0.473	1.508	1.057	1.311×10^{-4}	7.502×10^{-4}	—	3.218	3.218
1	1.332	2.724	3.829	3.405×10^{-4}	3.373×10^{-4}	3.624×10^{-4}	3.992	3.992
2	4.377	3.417	5.741	2.830×10^{-3}	2.813×10^{-4}	7.194×10^{0}	4.021	3.394
3	0.735	1.339	0.621	3.665×10^{-4}	4.330×10^{-4}	7.148×10^{0}	5.701	1.665

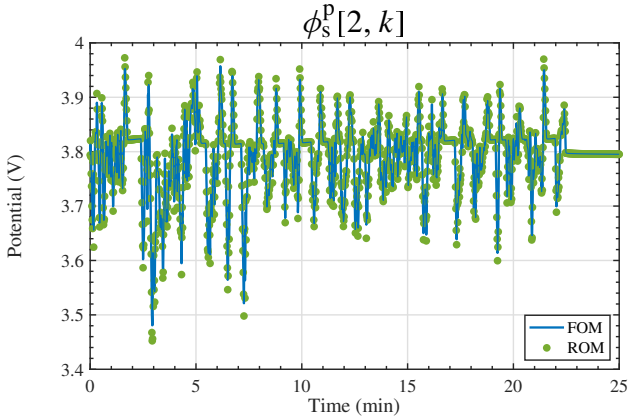
- The simulation overall inputs and outputs were:



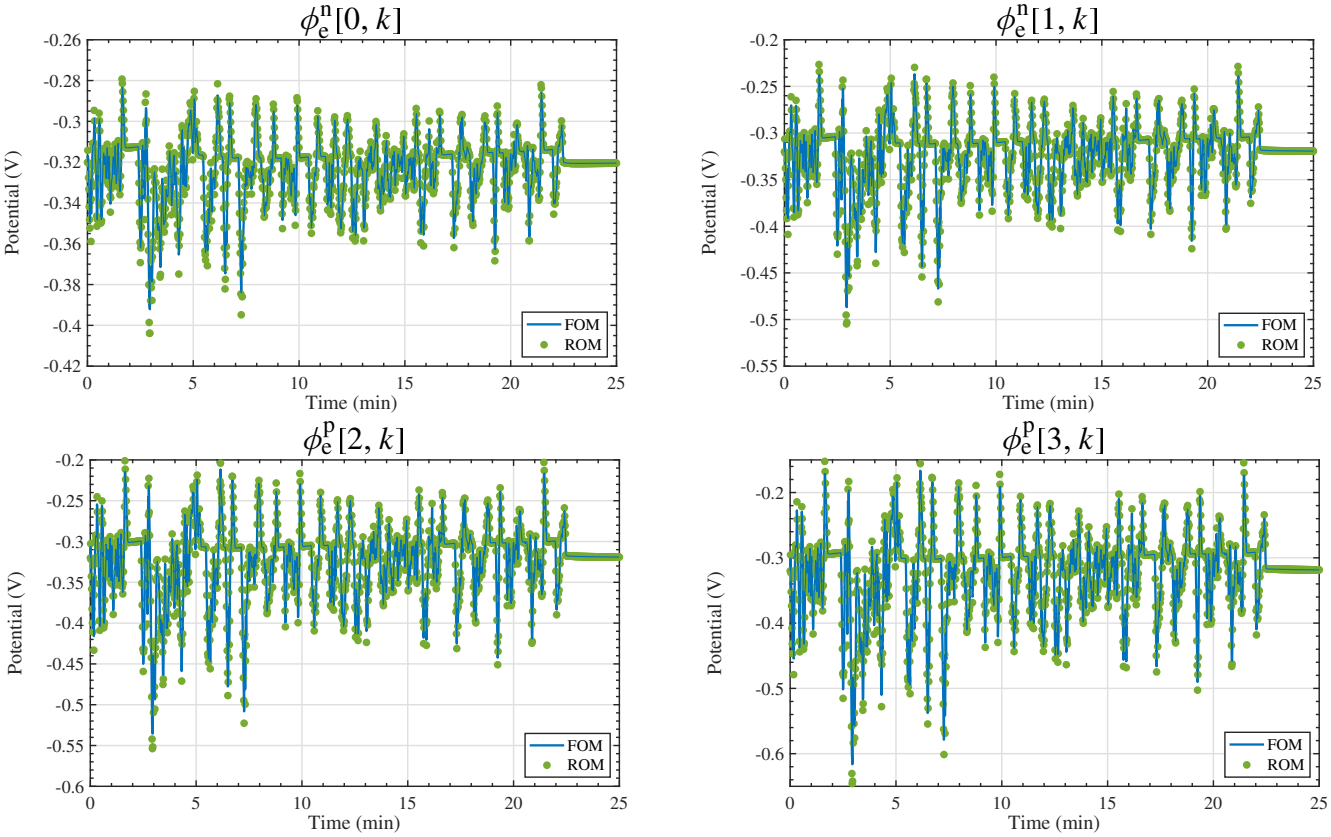


- Potential in the solid was (note: $\phi_{\mathrm{s}}^{\mathrm{n}}[0,k]=0$ and $\phi_{\mathrm{s}}^{\mathrm{p}}[3,k]=v_{\mathrm{cell}}[k]$):

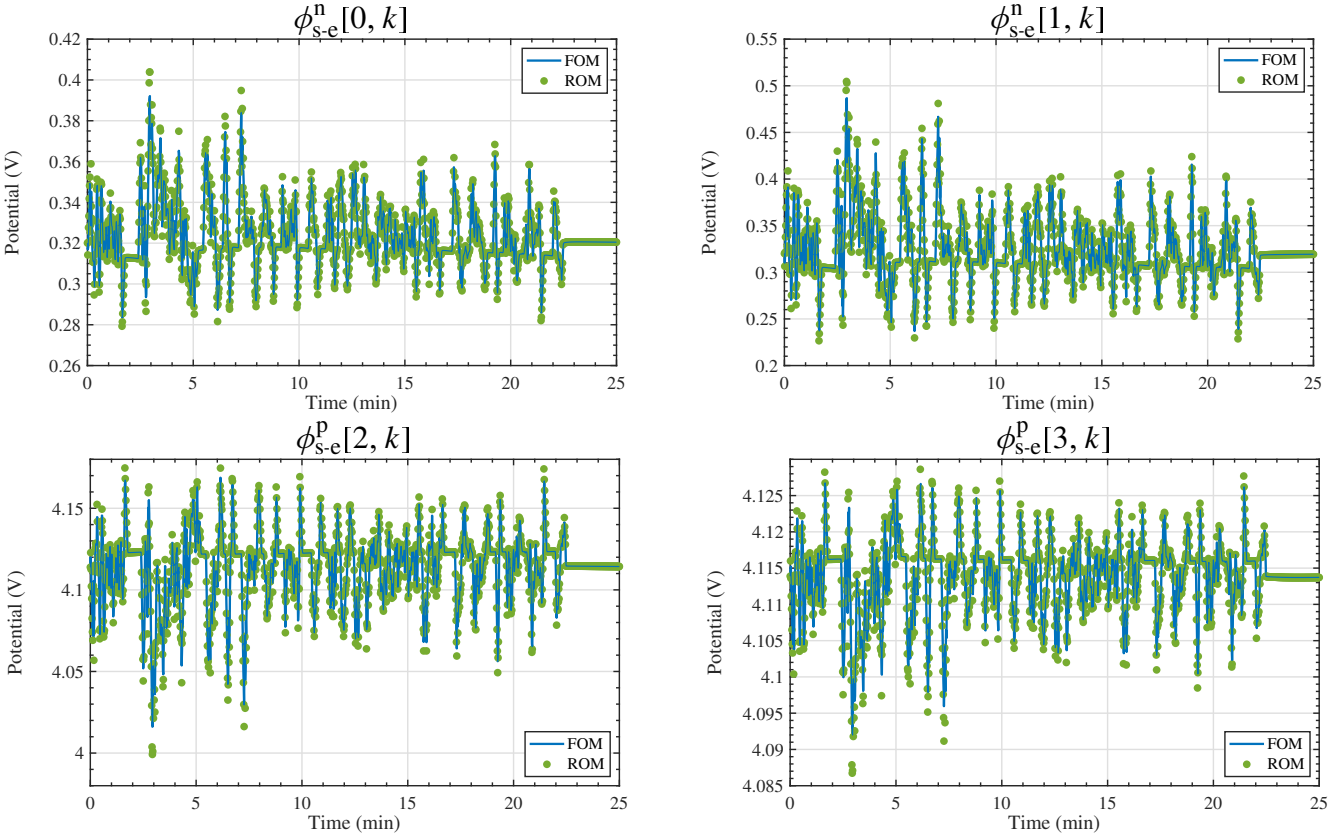




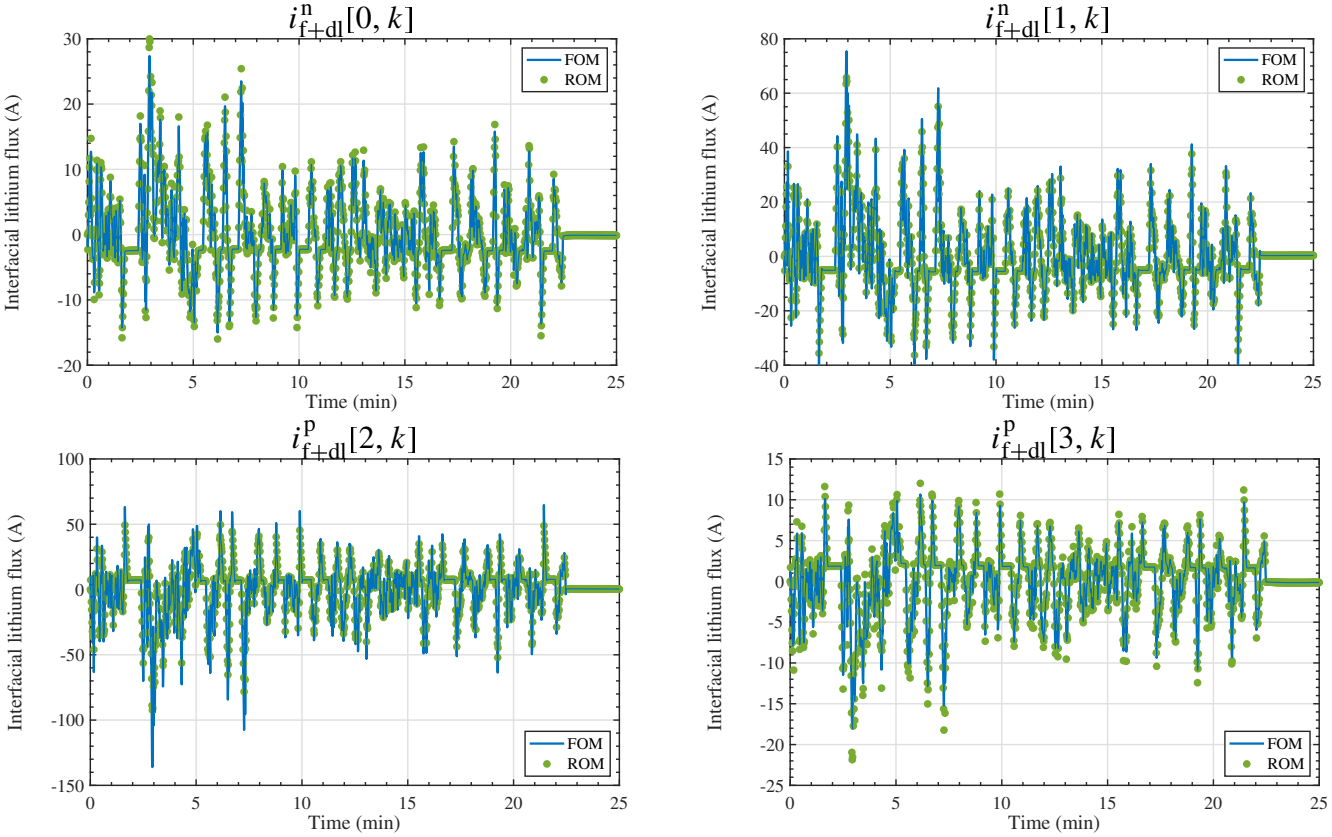
- Potential in the electrolyte was:



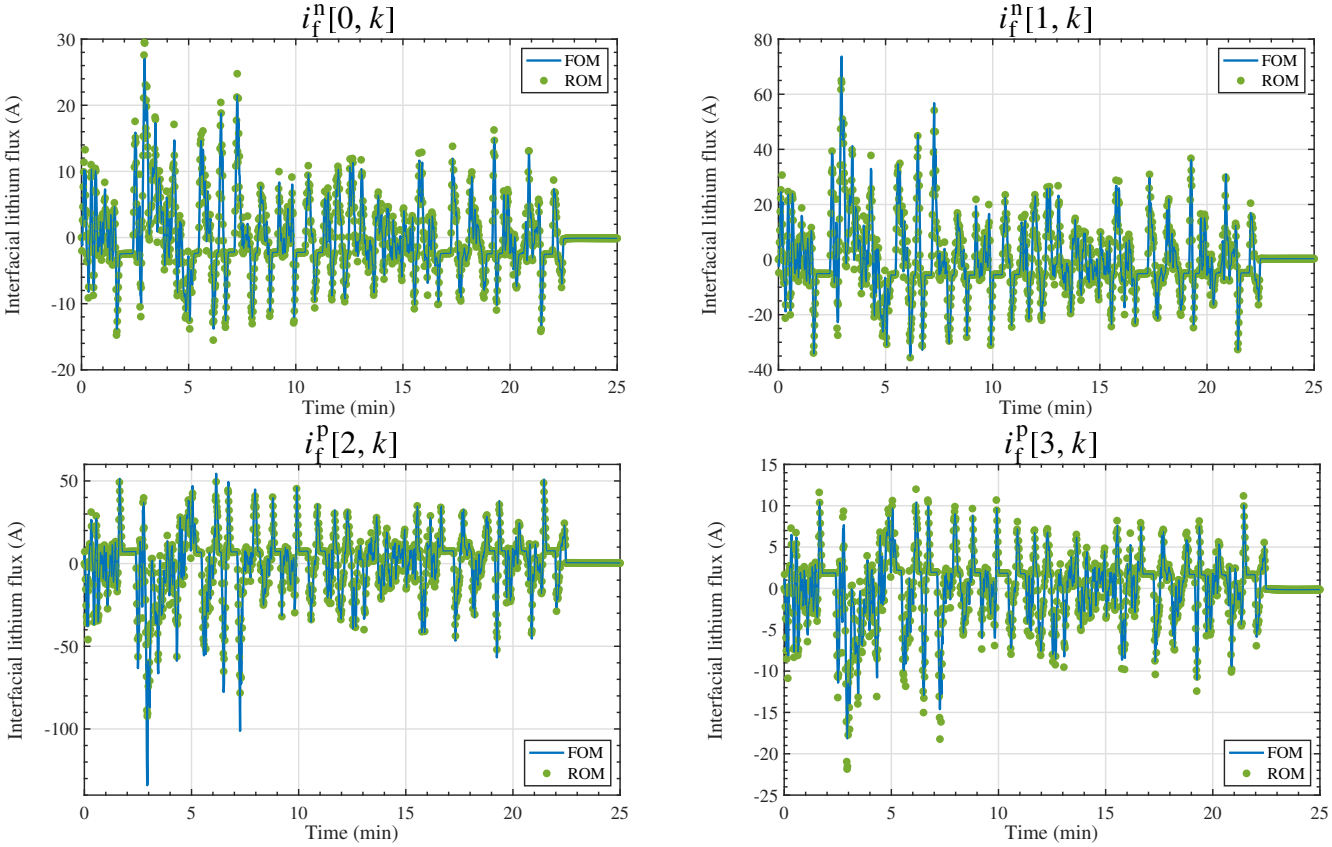
- Interphase potential difference was:



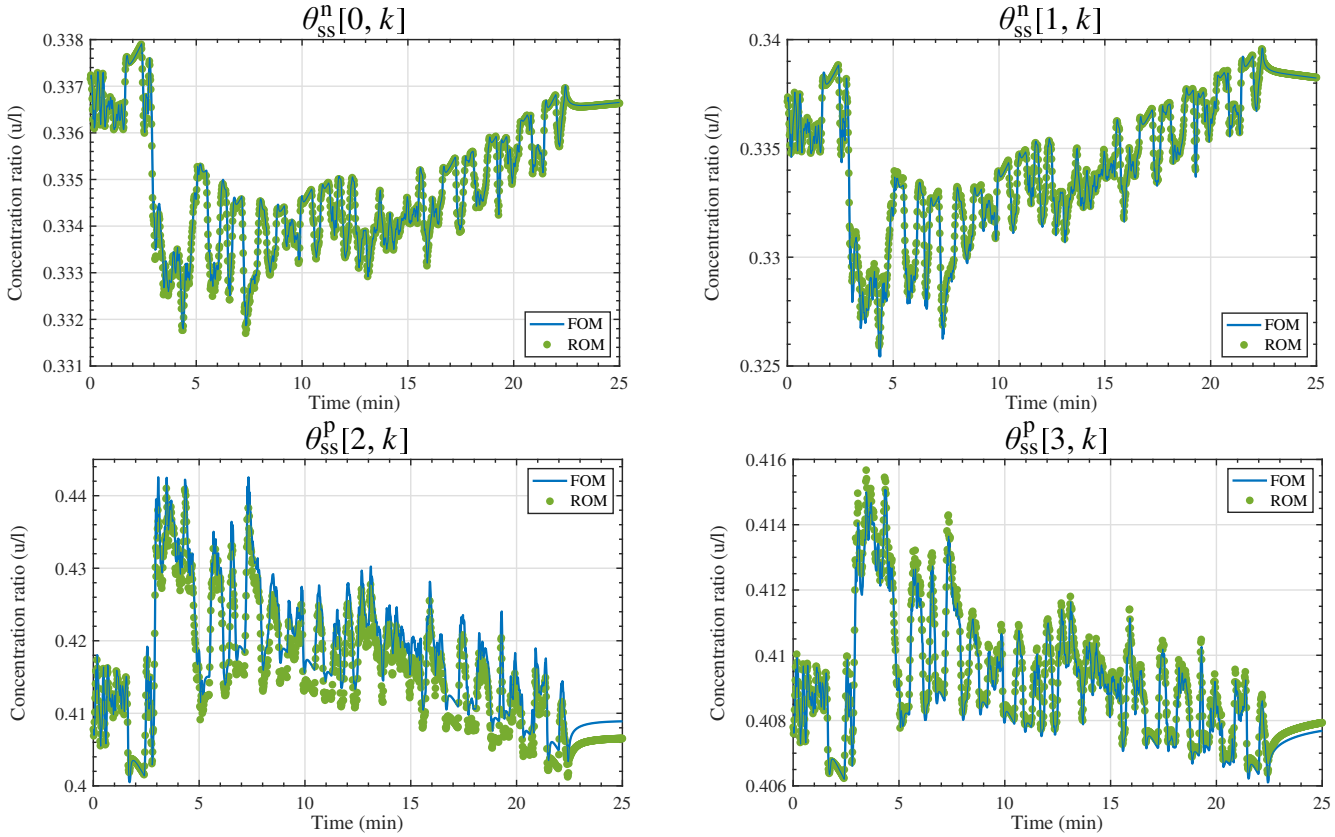
- Total interfacial lithium flux was:



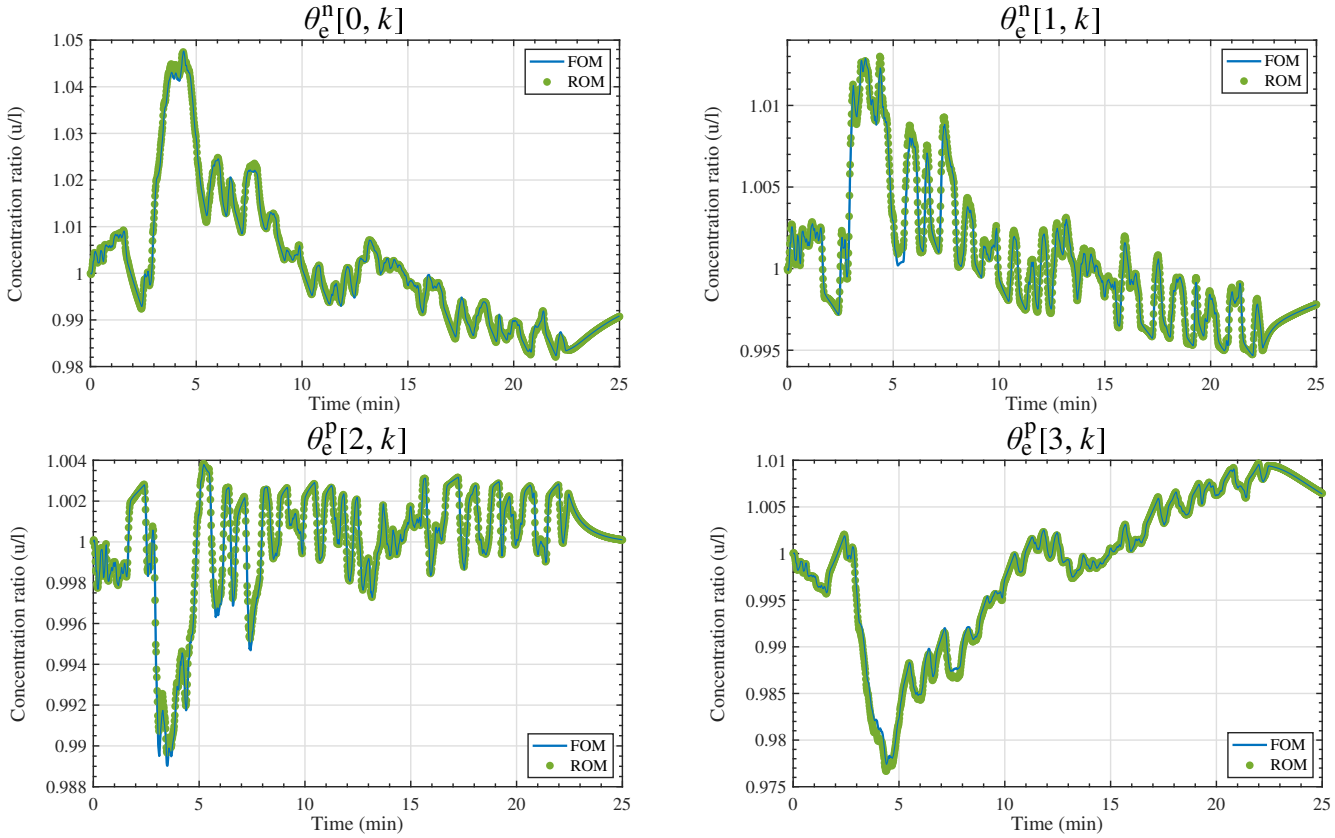
- Faradaic portion of interfacial lithium flux:



- Solid surface concentration was:

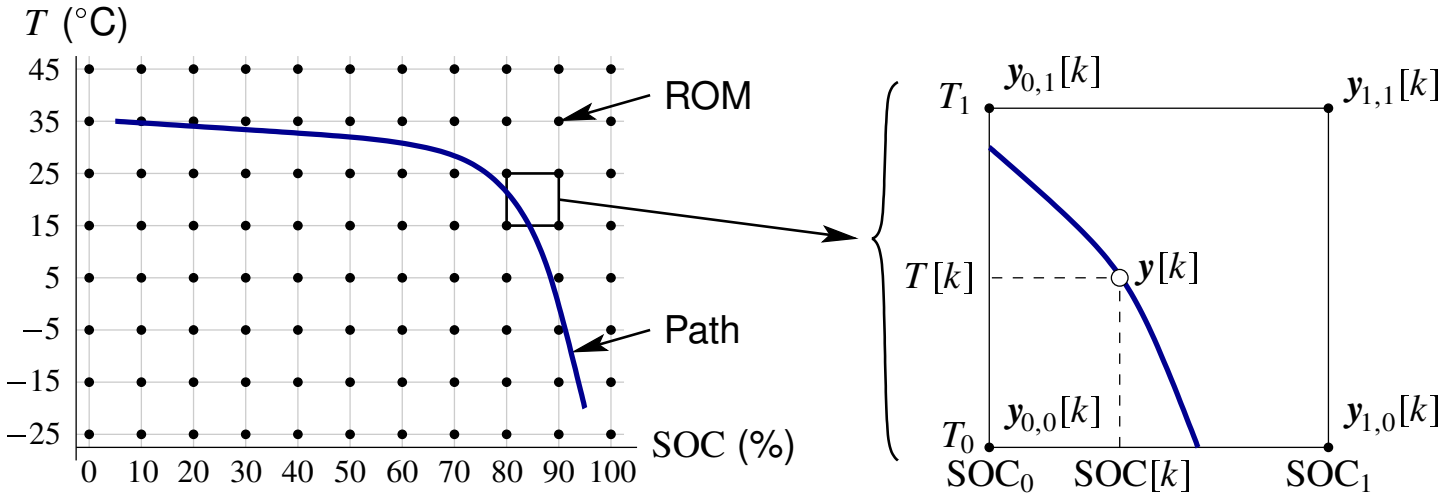


- Electrolyte concentration was:



4.7: Simulating over a wide operating range (output blending)

- The ROM we have seen so far applies to a specific operating setpoint.
- However, since cell dynamics vary with temperature and SOC, a single model is often not sufficient.
- To extend the cell model to apply to a wide range of operational setpoints, we use an output-blending approach.
- The basic idea is to precompute ROMs ($\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ matrices) for multiple SOC and temperature setpoints (shown as dots below).



- The SOC, temperature (z, T) trajectory of the cell as a function of time follows some path (drawn as a blue line) through the model space.
- To make a time-varying model that applies to every point on the path:
  - At every point in time we generate linear outputs $\boldsymbol{y}_{0,0}[k]$, $\boldsymbol{y}_{0,1}[k]$, $\boldsymbol{y}_{1,0}[k]$, and $\boldsymbol{y}_{1,1}[k]$ from the four "closest" ROMs to the present operating (z, T)... see zoom on the right of the figure.
  - We blend these four outputs, giving a best "average" linear output.
  - Then, we apply nonlinear corrections to this linear output.
- The real-time output-blending steps are described in the following:

Step 1

- Determine the present cell state of charge $z[k]$ and temperature $T[k]$.

Step 2

- Update state vectors of all pre-computed ROMs as:

$$\mathfrak{X}_j\,[k+1] = \mathfrak{A}_j \boldsymbol{x}_j\,[k] + \mathfrak{B}_j i_{\text{app}}\,[k]\,.$$

- Since each model's $\mathfrak{A}$ matrix is diagonal and $\mathfrak{B}_j = \mathbf{1}_{(n+1)\times 1}$, we note that :

$$\mathfrak{X}_j\,[k+1] = \text{diag}(\mathfrak{A}_j) \odot \mathfrak{X}_j\,[k] + \mathbf{1}_{(n+1)\times 1} i_{\text{app}}\,[k]\,,$$

  where "$\odot$" is a Hadamard (pointwise) product, or "$.\star$" in MATLAB.

- Then, all $\mathfrak{A}$-matrix diagonals can be combined in a single $(n+1)\times N$ matrix and all states can be combined in an $(n+1)\times N$ matrix as well.
  - In MATLAB, we first build "big $\mathfrak{A}$" and "big $\boldsymbol{x}$" matrices:

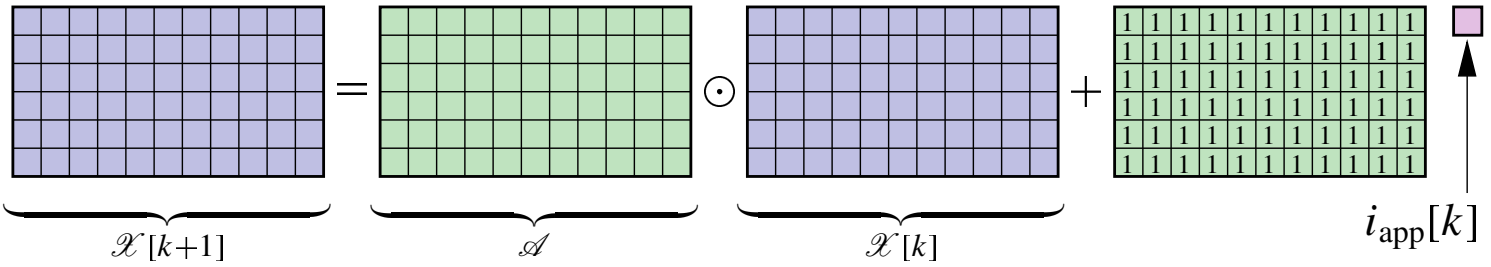
$$\mathscr{A} = \left[\begin{array}{c|c|c|c} \text{diag}(\mathfrak{A}_1) & \text{diag}(\mathfrak{A}_2) & \cdots & \text{diag}(\mathfrak{A}_N) \end{array}\right]$$

$$\mathscr{X}[k] = \left[\begin{array}{c|c|c|c} \mathfrak{X}_1[k] & \mathfrak{X}_2[k] & \cdots & \mathfrak{X}_N[k] \end{array}\right],$$

    where each column represents a model linearized around one setpoint.

- The state equations of all models are updated using a Hadamard product in MATLAB, mathematically written as:

$$\mathscr{X}[k+1] = \mathscr{A} \odot \mathscr{X}[k] + \mathbf{1}_{(n+1)\times N} i_{\text{app}}[k].$$



Step 3

- Compute output from only the four closest pre-computed ROMs as:

$$\boldsymbol{y}_j[k] = \mathfrak{C}_j \mathfrak{X}_j[k] + \mathfrak{D}_j i_{\text{app}}[k].$$

- The structure of matrices $\mathfrak{C}$ and $\mathfrak{D}$ of a $n+1$ states q output system are denoted as:

$$\mathfrak{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} & \text{res}_1^* \\ c_{21} & c_{22} & \cdots & c_{2n} & \text{res}_2^* \\ c_{31} & c_{32} & \cdots & c_{3n} & \text{res}_3^* \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{q1} & c_{q2} & \cdots & c_{qn} & \text{res}_q^* \end{bmatrix},$$

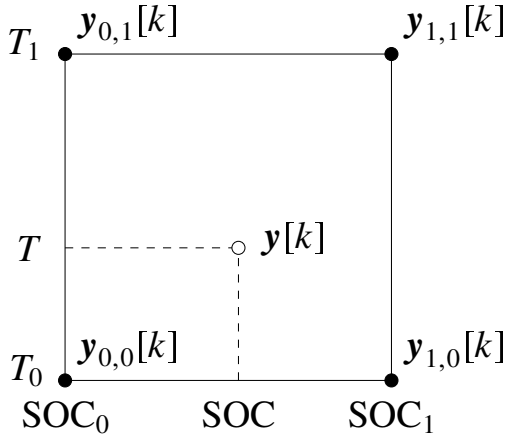
$$\mathfrak{D} = \left[\lim_{s\to\infty} \boldsymbol{G}_1(s), \ \lim_{s\to\infty} \boldsymbol{G}_2(s), \ \cdots \ \lim_{s\to\infty} \boldsymbol{G}_q(s) \right]^T.$$

- Since the $\mathfrak{C}$ matrices are not in diagonal forms, we are unable to substitute Hadamard products for efficiency, as done previously.

Step 4

- The next step is to blend the linear outputs only for the four models linearized at setpoints closest to the present (z, T) operating point.

- In the illustration, cell's present SOC is marked "SOC", and SOC_0 and SOC_1 are the SOC setpoints of the closest precomputed models to the present SOC that bracket the present SOC: $\text{SOC}_0 \le \text{SOC} \le \text{SOC}_1$.



- The cell's present temperature is marked on the figure as "T" and T_0 and T_1 are the temperature setpoints of the closest precomputed models to T that bracket T: $T_0 \leq T \leq T_1$.
- Define:

$$\theta = \frac{\text{SOC} - \text{SOC}_0}{\text{SOC}_1 - \text{SOC}_0} \quad \text{and} \quad \psi = \frac{T - T_0}{T_1 - T_0};$$

then if $\bar{\theta} = 1 - \theta$ and $\bar{\psi} = 1 - \psi$,

$$\begin{aligned} \boldsymbol{y}[k] &= \bar{\psi}(\bar{\theta}\boldsymbol{y}_{0,0}[k] + \theta\boldsymbol{y}_{1,0}[k]) + \psi(\bar{\theta}\boldsymbol{y}_{0,1}[k] + \theta\boldsymbol{y}_{1,1}[k]) \\ &= \gamma_{0,0}\boldsymbol{y}_{0,0}[k] + \gamma_{0,1}\boldsymbol{y}_{0,1}[k] + \gamma_{1,0}\boldsymbol{y}_{1,0}[k] + \gamma_{1,1}\boldsymbol{y}_{1,1}[k] \qquad (4.8) \\ &= \sum_{j=1}^{N} \gamma_j \boldsymbol{y}_j[k]. \qquad (4.9) \end{aligned}$$

- Notice that although we must update the state vector of all models every time step, we are not required to compute the output vector of any model that is not being used for the present calculation of $\boldsymbol{y}[k]$.
- That is, in the set of weights $\{\gamma_j\}$ for $1 \leq j \leq N$ in Eq. (4.9), only the four closest models have nonzero weights, denoted as $\gamma_{0,0}$, $\gamma_{0,1}$, $\gamma_{1,0}$, and $\gamma_{1,1}$ in Eq. (4.8), reducing required computation.

Step 5

- Apply the nonlinear corrections from Topic 4.5 to $\boldsymbol{y}[k]$ to produce nonlinear outputs at this point in time.

Step 6

- Repeat Steps 1 through 5 every iteration k.

4.8: Simulation results over wide operating range

- This section presents simulation results to demonstrate the fidelity of the ROM with respect to the FOM (FOM simulated in COMSOL).
- ROMs generated at SOCs in 0:0.02:1, each having 8 dynamic states.

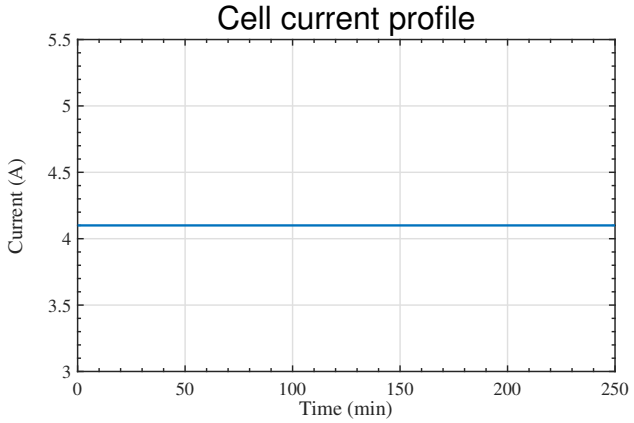
Constant-current profile

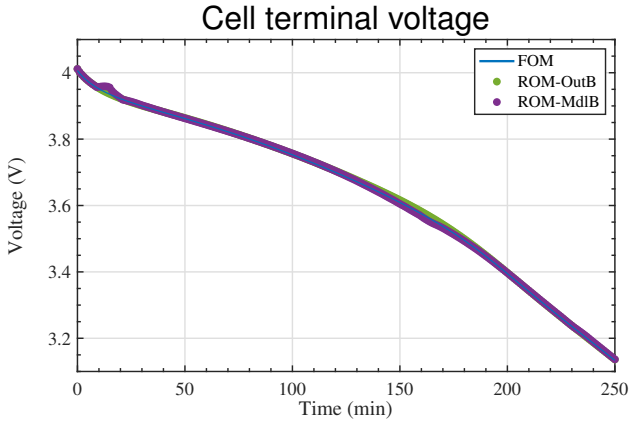
- First, consider C/5 CC discharge, initialized at rest at 95 % SOC.

RMS errors for each variable (output blending)

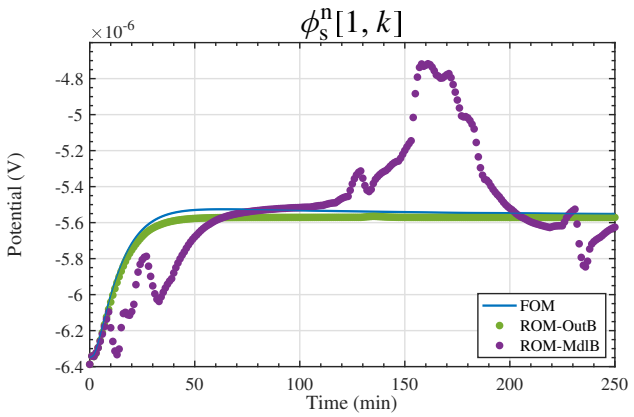
$\tilde{x}$	$i_{\mathrm{f+dl}}$ [A]	i_{f} [A]	i_{dl} [A]	θ_{ss} [u/l]	θ_{e} [u/l]	ϕ_{s} [mV]	ϕ_{e} [mV]	$\phi_{\mathrm{s\text{-}e}}$ [mV]
0	4.918×10^{-2}	4.918×10^{-2}	—	7.930×10^{-4}	2.075×10^{-3}	—	1.263	1.263
1	9.932×10^{-2}	9.932×10^{-2}	—	6.856×10^{-4}	1.864×10^{-3}	3.347×10^{-5}	1.065	1.065
2	2.640×10^{0}	2.640×10^{0}	—	3.718×10^{-2}	1.905×10^{-3}	3.835×10^{0}	1.049	3.653
3	1.236×10^{0}	1.236×10^{0}	—	1.469×10^{-2}	5.984×10^{-3}	3.797×10^{0}	3.450	1.189

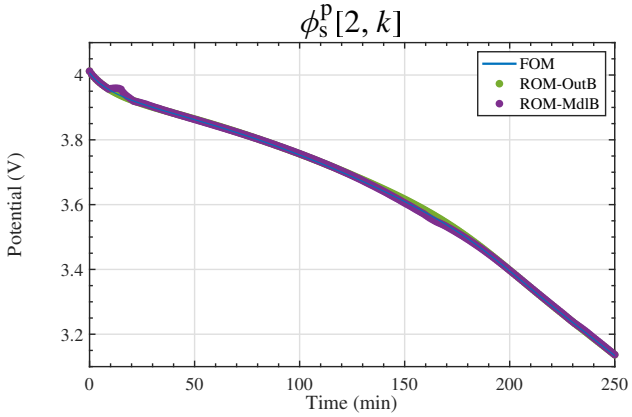
- The simulation overall inputs and outputs were:



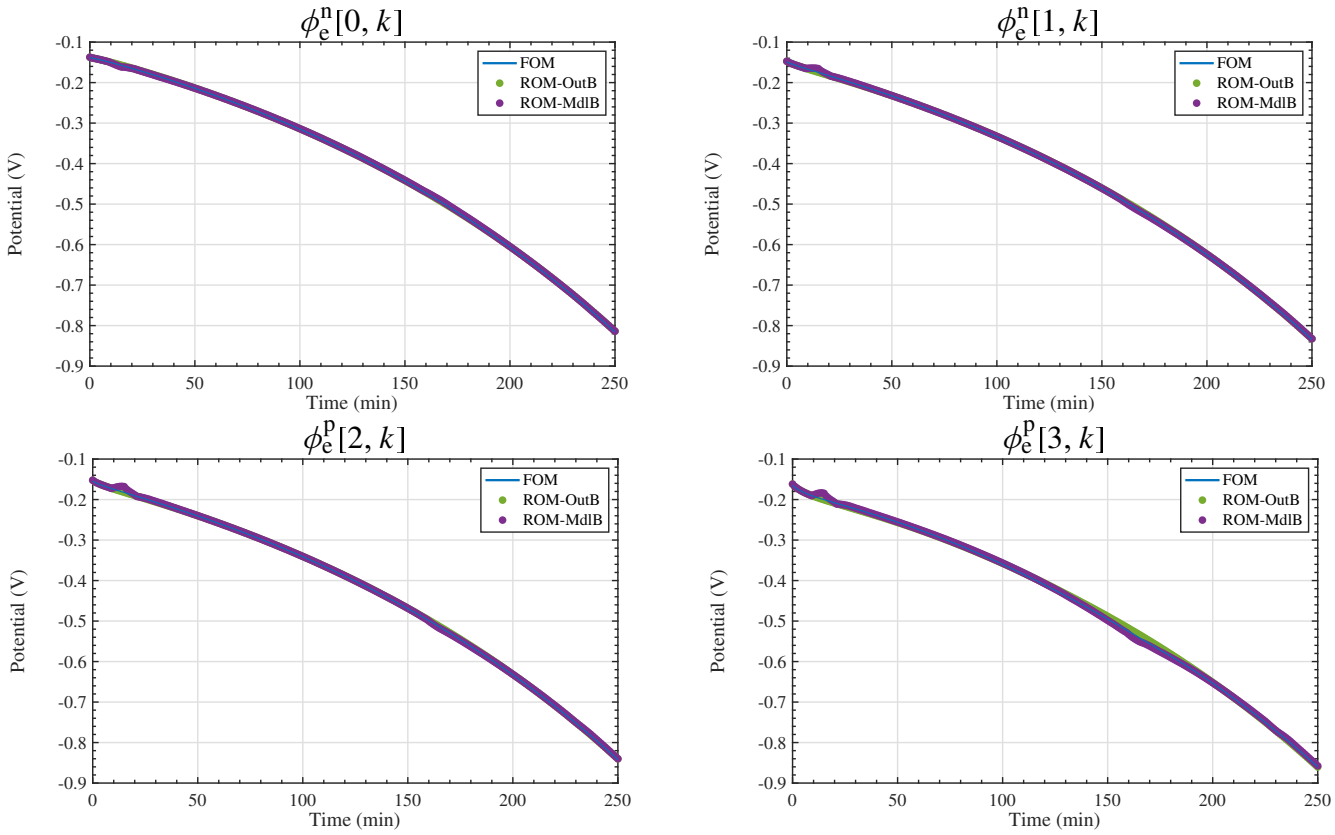


- Potential in the solid was (note: $\phi_{\mathrm{s}}^{\mathrm{n}}[0,k]=0$ and $\phi_{\mathrm{s}}^{\mathrm{p}}[3,k]=v_{\mathrm{cell}}[k]$):

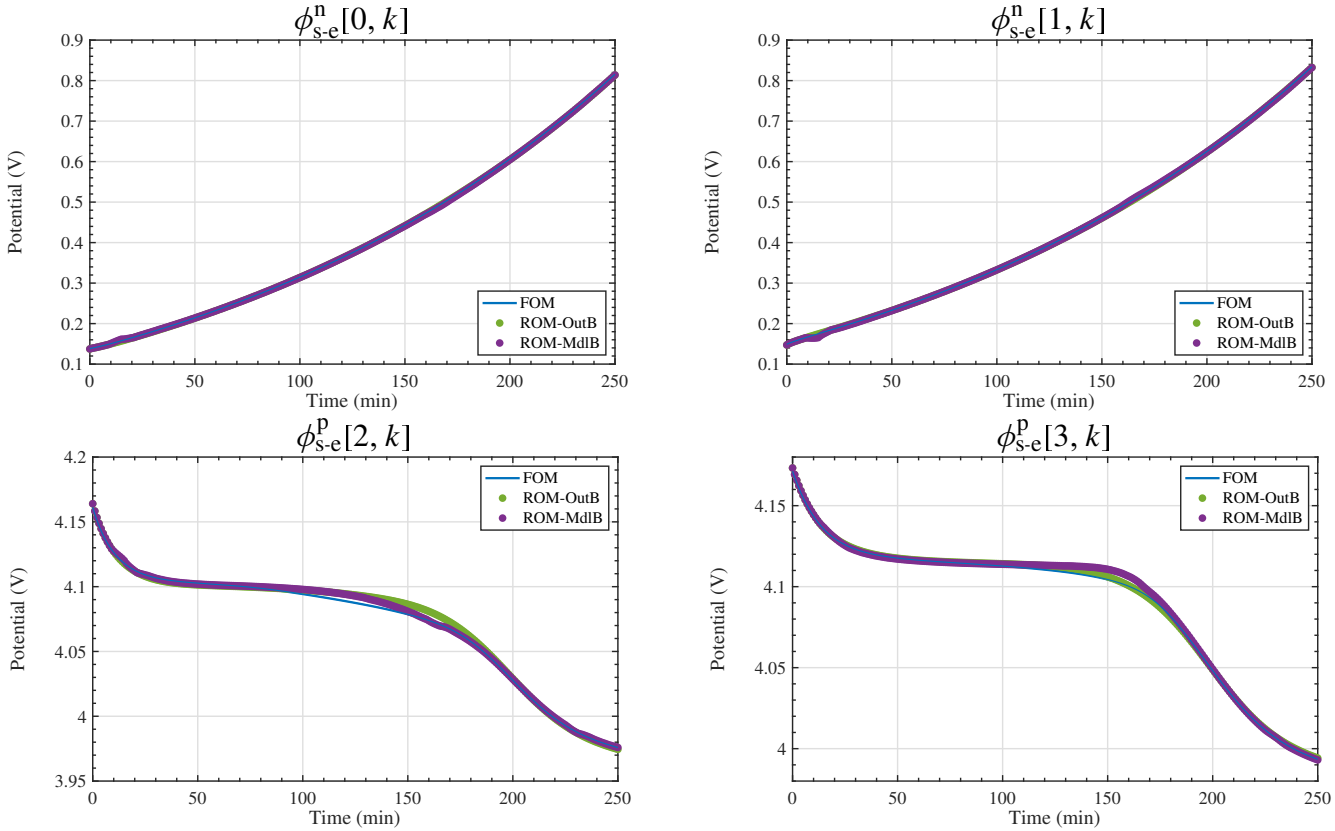




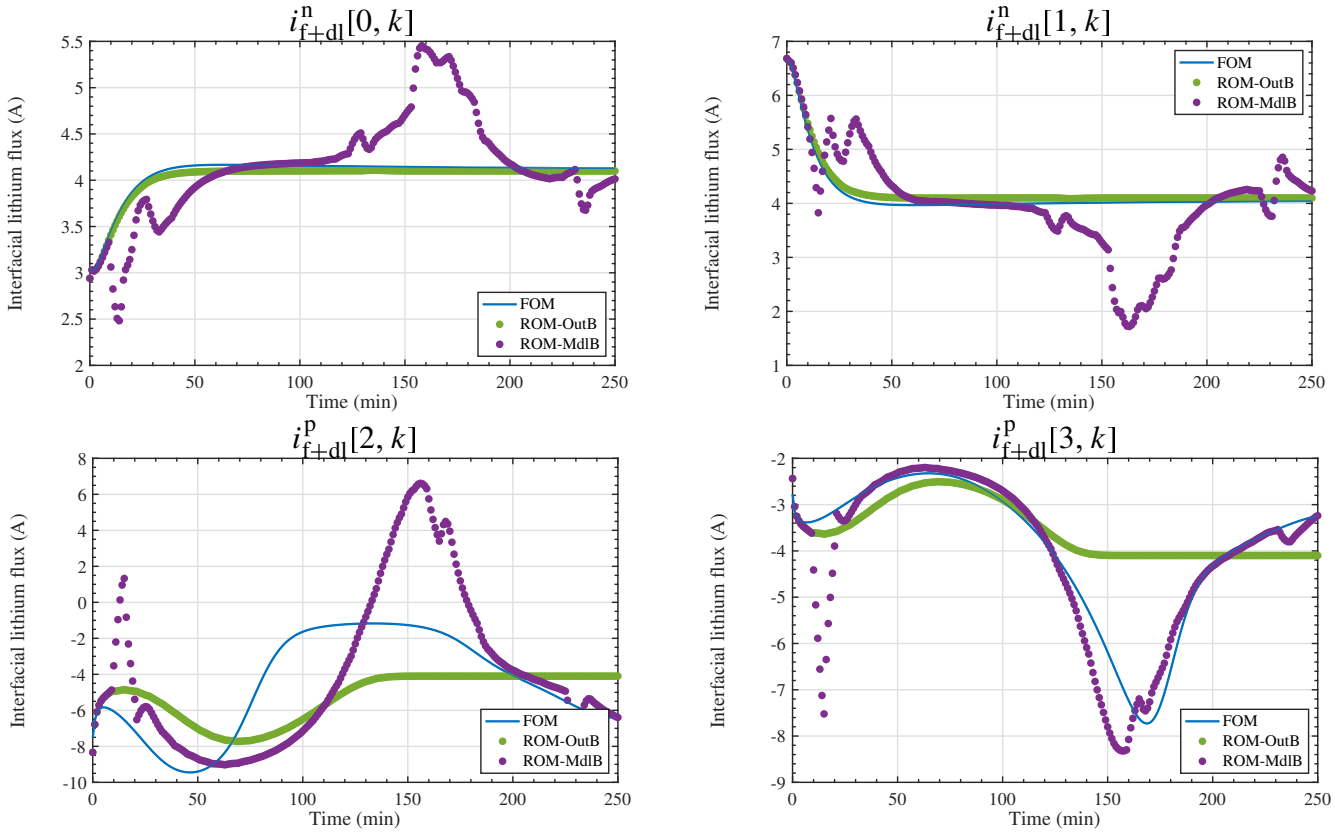
- Potential in the electrolyte was:



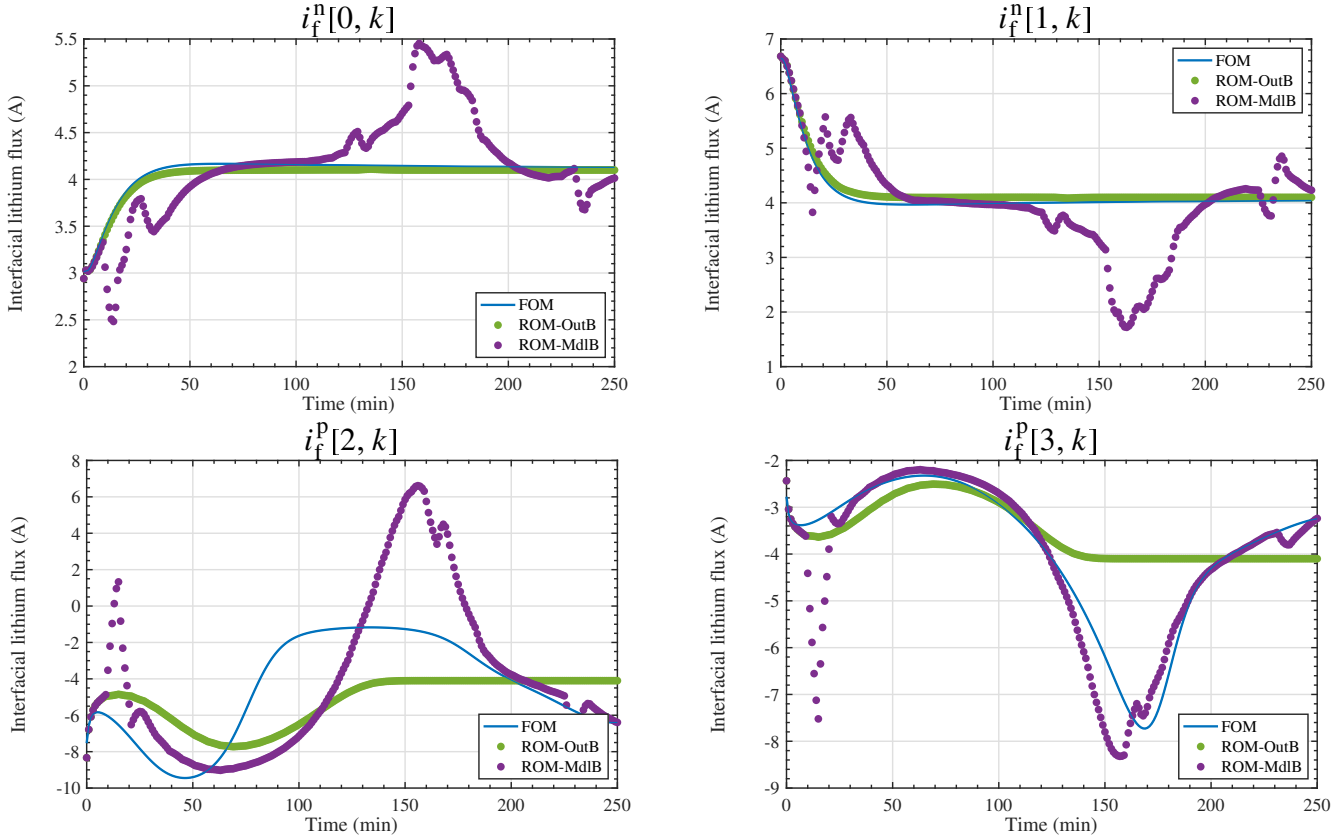
- Interphase potential difference was:



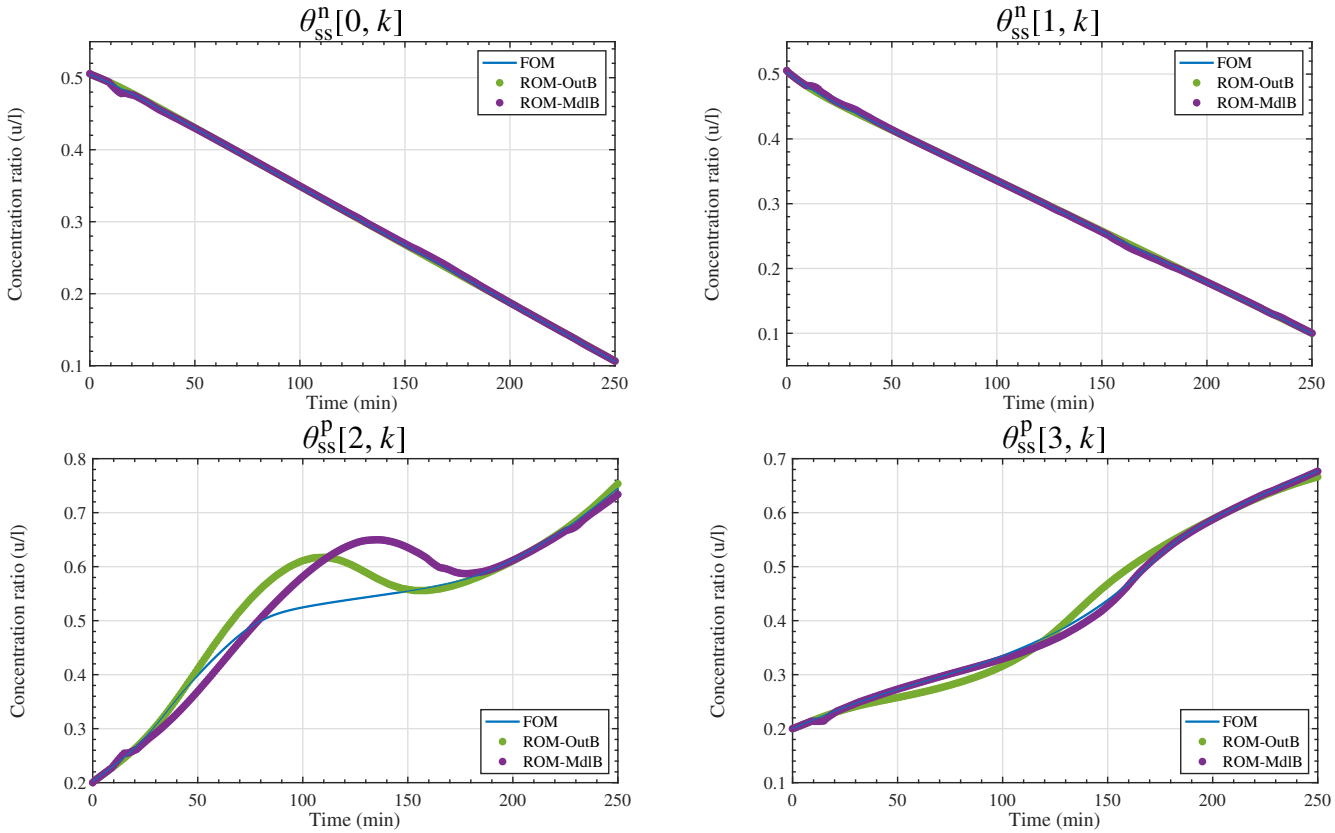
- Total interfacial lithium flux was:



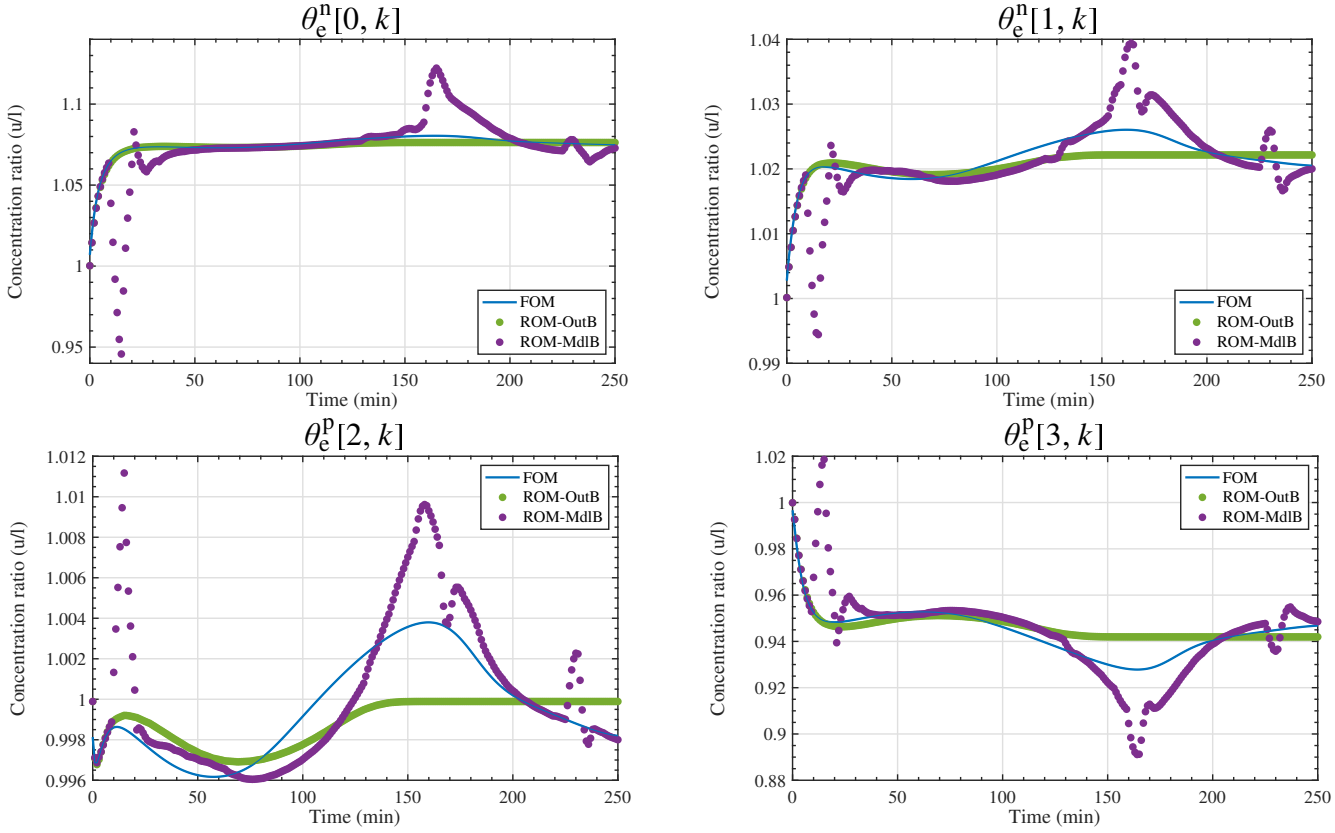
- Faradaic interfacial lithium flux:



- Solid surface concentration was:



- Electrolyte concentration was:



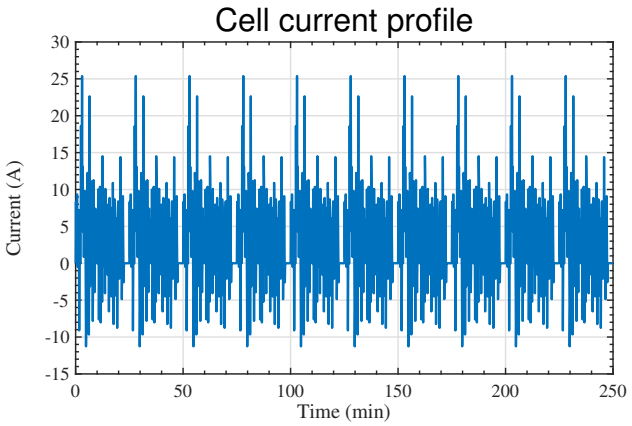
Dynamic current profile

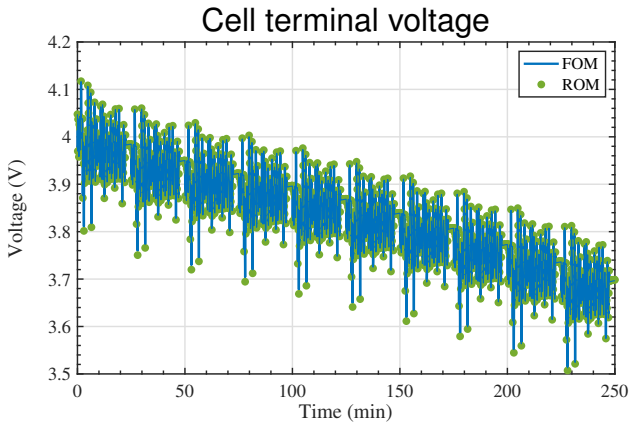
- The second scenario investigated a simulation of ten charge-depleting UDDS cycles, initialized at rest at 95 % SOC.
- Same ROMs were used as those used with constant-current profile.

RMS errors for each variable (output blending)

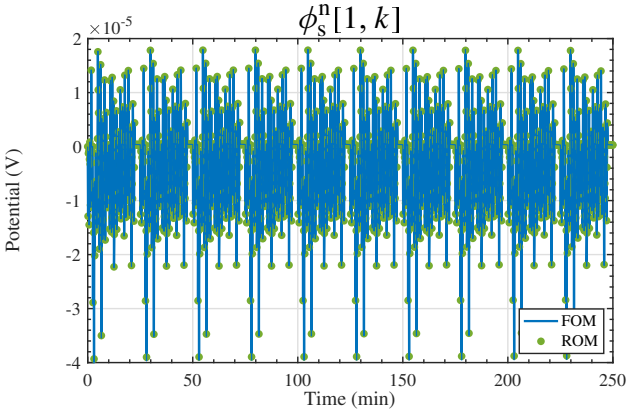
$\tilde{x}$	$i_{\mathrm{f+dl}}$ [A]	i_{f} [A]	i_{dl} [A]	θ_{ss} [u/l]	θ_{e} [u/l]	ϕ_{s} [mV]	ϕ_{e} [mV]	$\phi_{\mathrm{s\text{-}e}}$ [mV]
0	9.406×10^{-2}	9.406×10^{-2}	—	1.429×10^{-4}	8.859×10^{-4}	—	0.645	0.645
1	3.202×10^{-1}	3.202×10^{-1}	—	3.088×10^{-4}	8.804×10^{-4}	7.608×10^{-5}	0.819	0.819
2	2.050×10^{0}	2.050×10^{0}	—	2.822×10^{-2}	9.134×10^{-4}	1.560×10^{0}	0.851	1.718
3	3.820×10^{-1}	3.820×10^{-1}	—	1.296×10^{-2}	2.428×10^{-3}	1.857×10^{0}	1.722	0.554

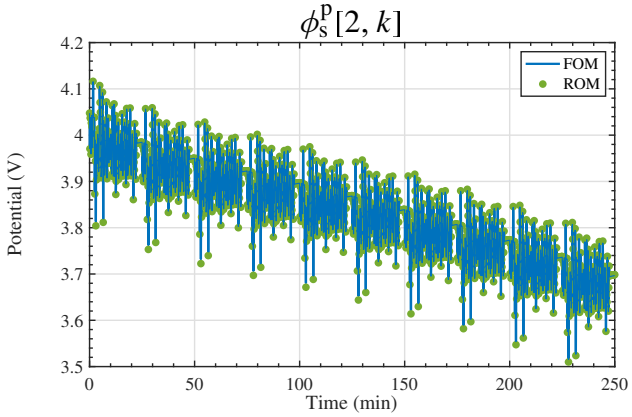
- The simulation overall inputs and outputs were:



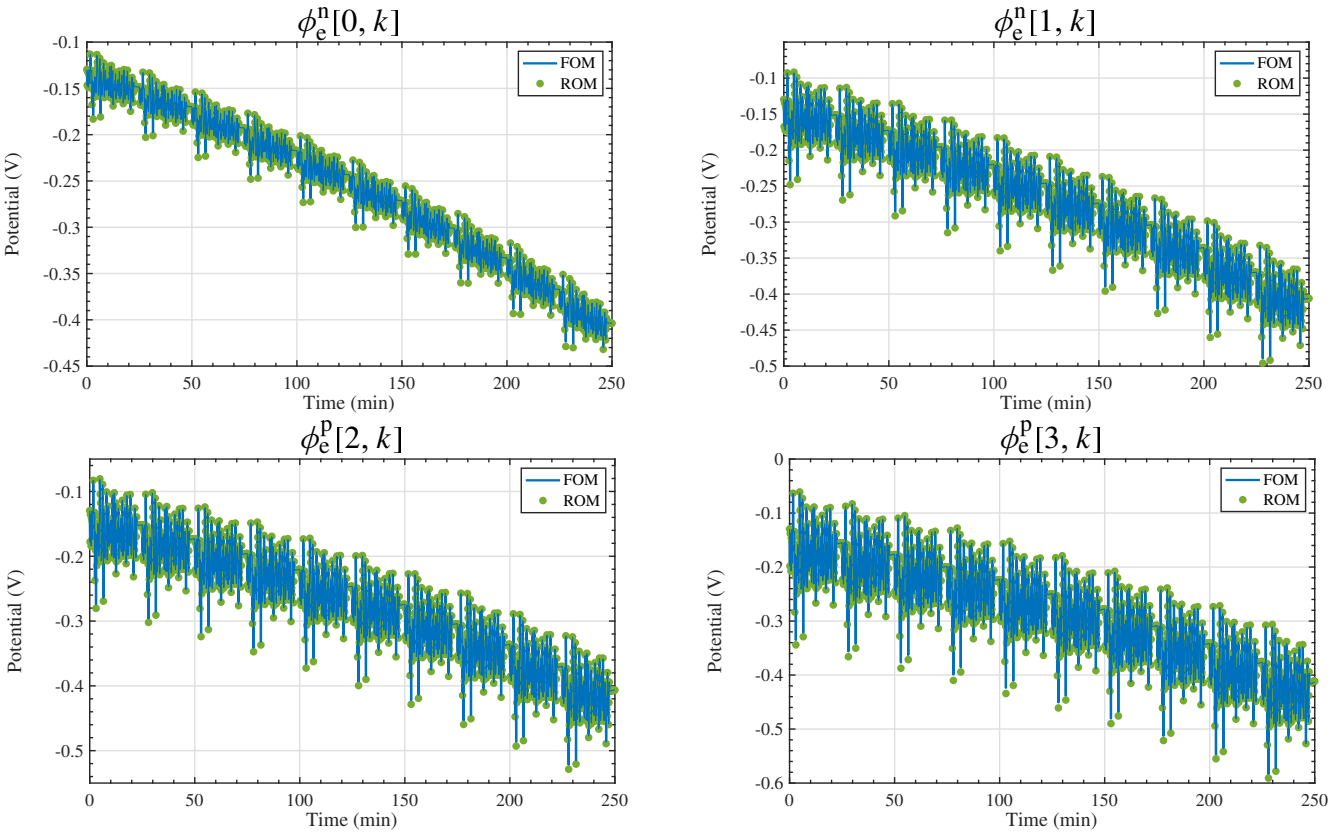


- Potential in the solid was (note: $\phi_{\mathrm{s}}^{\mathrm{n}}[0,k] = 0$ and $\phi_{\mathrm{s}}^{\mathrm{p}}[3,k] = v_{\mathrm{cell}}[k]$):

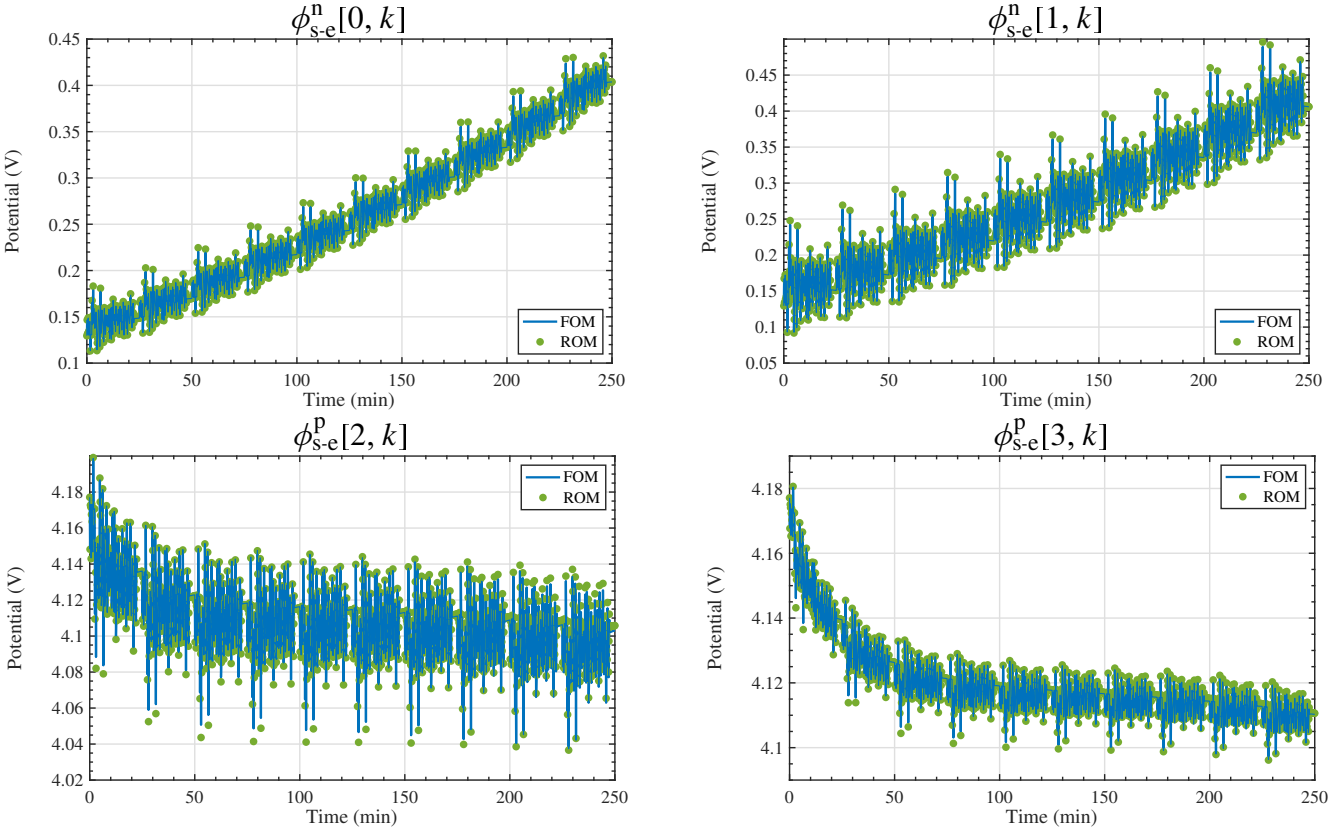




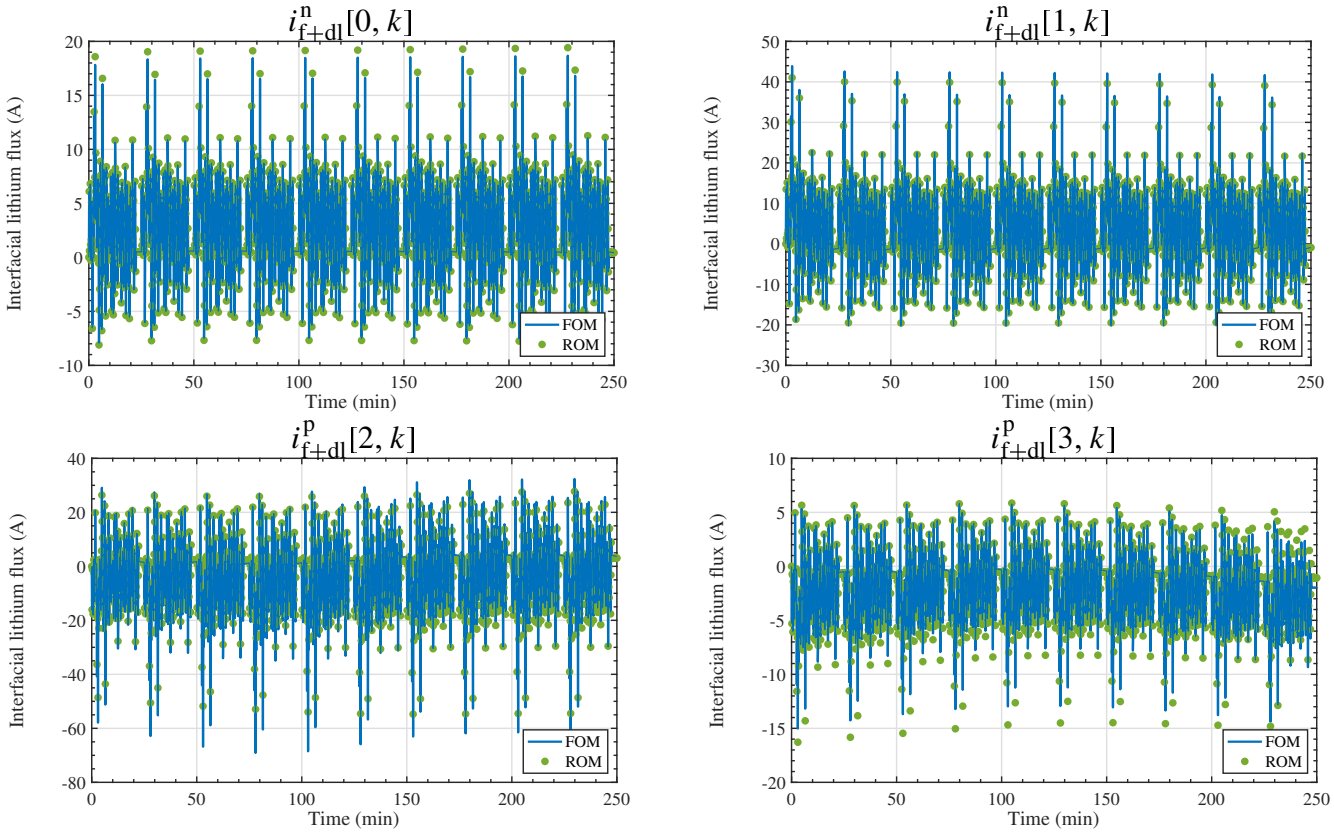
- Potential in the electrolyte was:



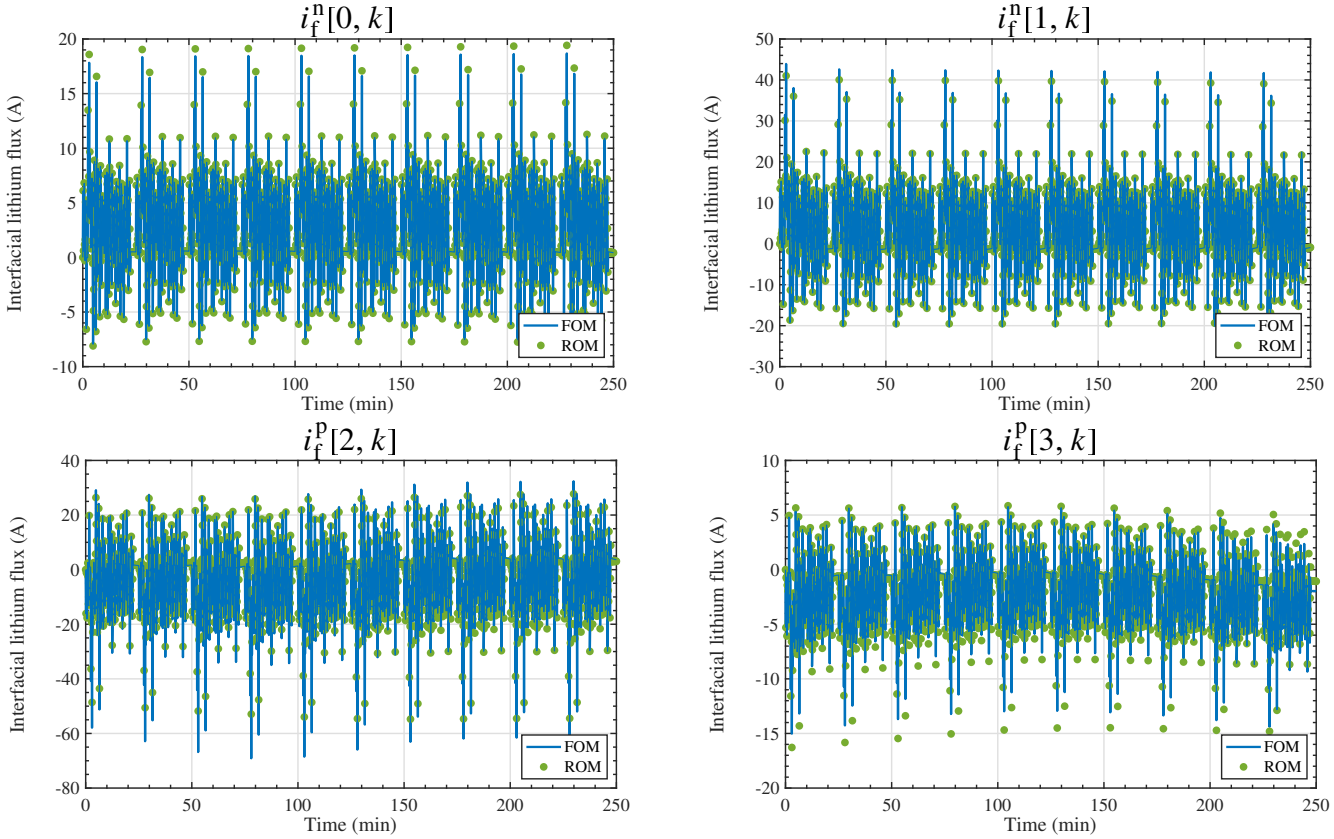
- Interphase potential difference was:



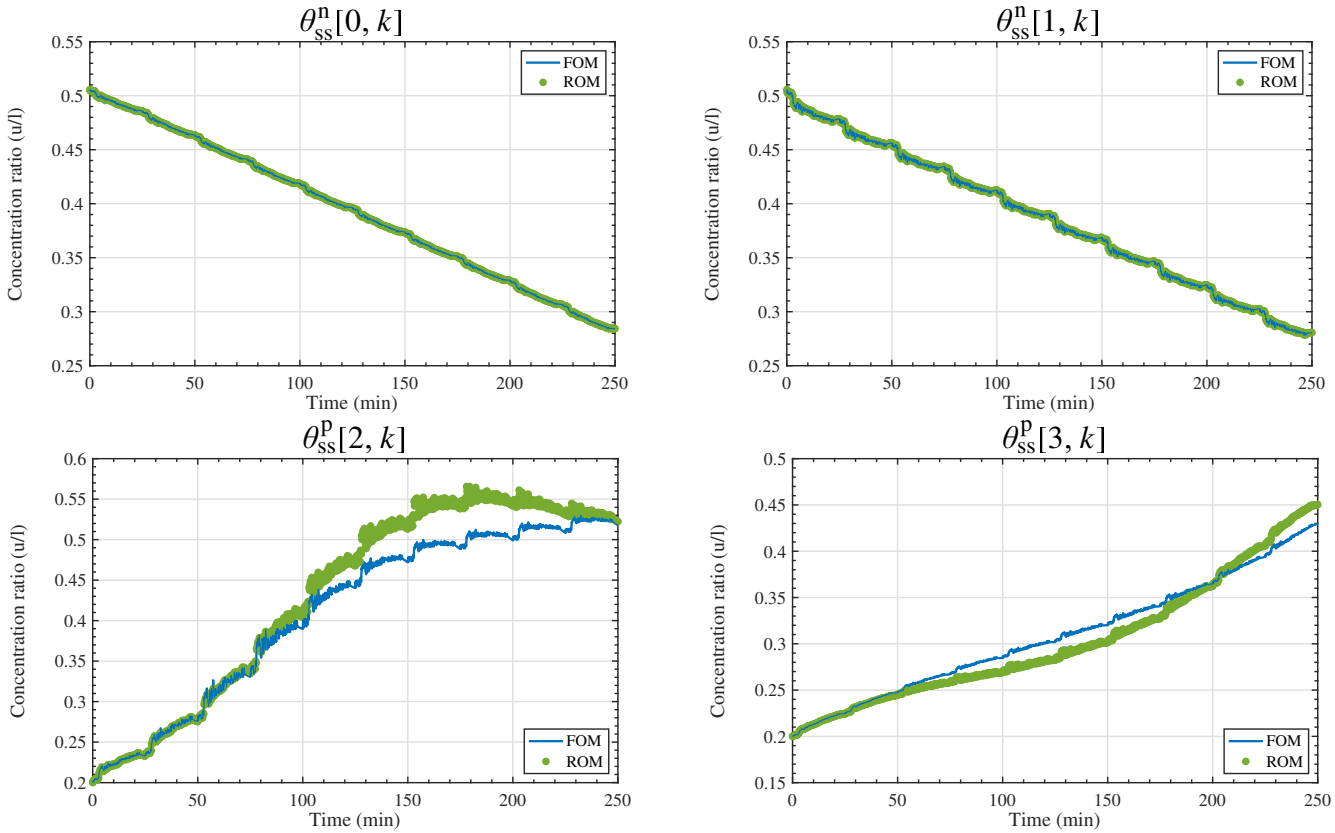
- Total interfacial lithium flux was:



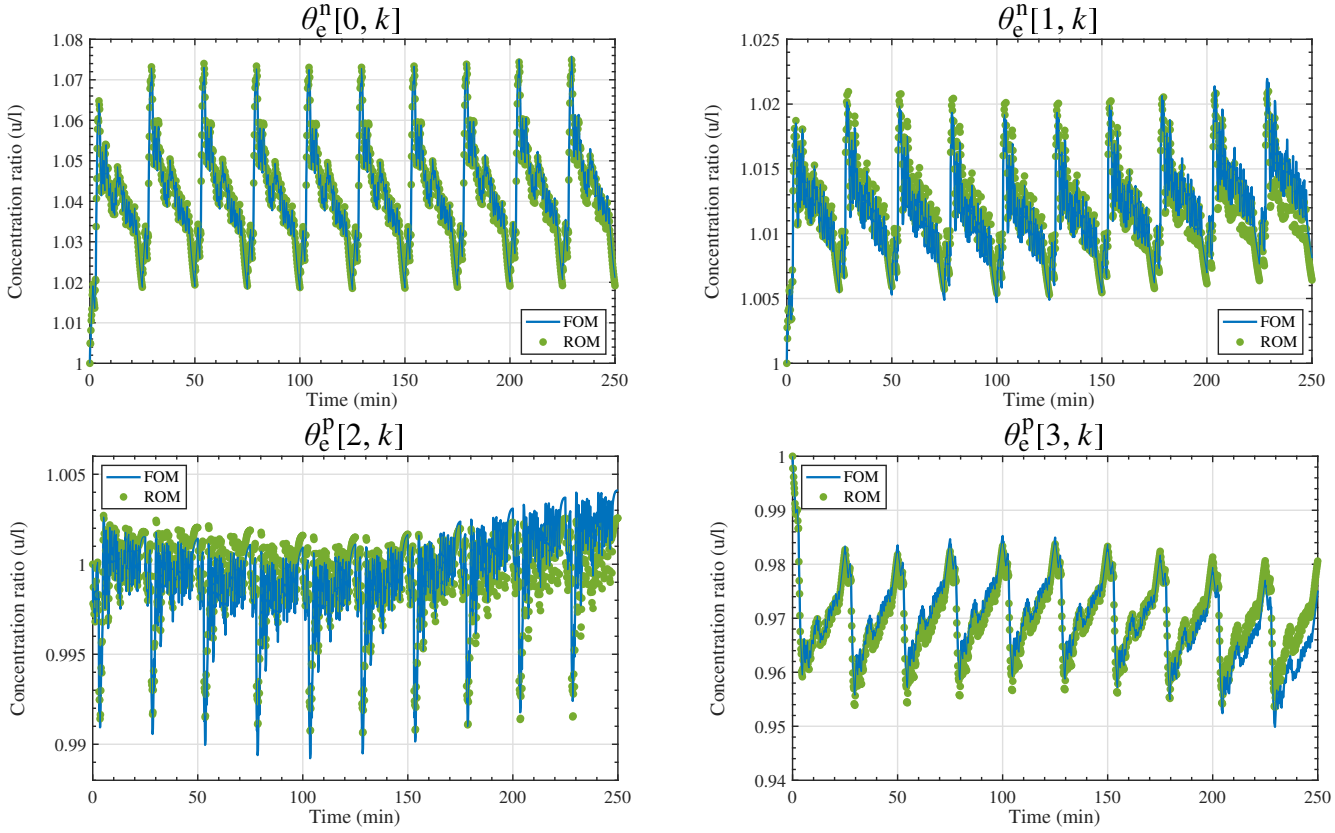
- Faradaic interfacial lithium flux:



- Solid surface concentration was:



- Electrolyte concentration was:



4.9: Simulating constant voltage and constant power

Simulating constant voltage

- The simulation strategy as described so far requires $i_{\text{app}}[k]$ as input and produces $v_{\text{cell}}[k]$ as output (as well as electrochemical variables).
- But, what if we desire for $v_{\text{cell}}[k]$ to be the input instead of $i_{\text{app}}[k]$ (e.g., to simulate the CV portion of CC/CV charging)?
- We take an approach similar to that presented in ECE5720, but complicated by the fact that ROM voltage is nonlinear in $i_{\text{app}}[k]$.

IDEA: Compute, iteration-by-iteration, the exact required applied current at that iteration to achieve the desired terminal voltage.

- The net effect is that $v_{\text{cell}}[k]$ *appears* to be the input since we are controlling the actual model input to enforce the desired $v_{\text{cell}}[k]$, making $i_{\text{app}}[k]$ (and the electrochemical variables) the output.
- To see how to do this, note that we can write present voltage as a nonlinear function of the present state and present input:

$$v_{\text{cell}}[k] = g_v(\boldsymbol{x}[k], i_{\text{app}}[k]).$$

- The present state $\boldsymbol{x}[k]$, however, is not a function of the present input. It is a function only of the prior inputs.
- So, the present state simply creates a bias point for cell voltage, and the applied current modulates the actual voltage around this bias.
- We can think of the operation of computing $i_{\text{app}}[k]$ as a kind of inverse of $g_v(\cdot)$ for the particular present value of $\boldsymbol{x}[k]$:

$$i_{\text{app}}[k] = g_v^{-1}(v_{\text{cell}}[k]; \boldsymbol{x}[k]).$$

- How to compute this inverse? Digging into the equations that compute $v_{\text{cell}}[k]$, we cannot find a closed-form expression.
- So, instead, we use nonlinear optimization to search for $i_{\text{app}}[k]$ that causes $v_{\text{cell}}[k] = v_{\text{desired}}[k]$.

Simulating constant power

- Power is equal to applied current multiplied by terminal voltage.
- In terms of the analysis to date, $p_{\text{cell}}[k] = v_{\text{cell}}[k] i_{\text{app}}[k]$.
- To simulate constant power, we use nonlinear optimization to search for $i_{\text{app}}[k]$ that causes $p_{\text{cell}}[k] = p_{\text{desired}}[k]$.
- Note that there will be two solutions(!): we need to be careful to find the solution that results in positive $v_{\text{cell}}[k]$.
  - If we desire negative power, we constrain $i_{\text{app}}[k]$ to be negative;
  - If we desire positive power, we constrain $i_{\text{app}}[k]$ to be positive.

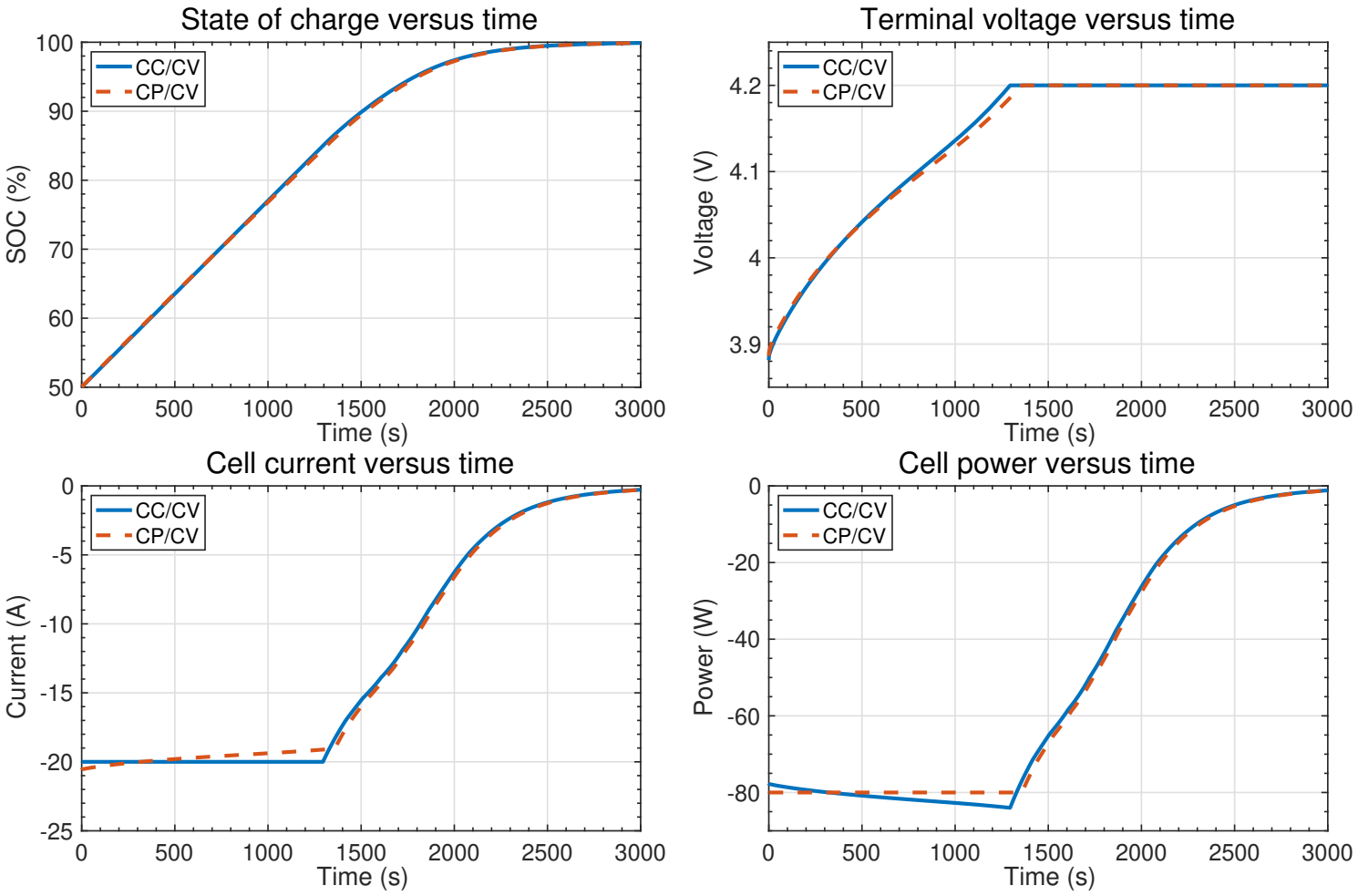
```
for k = 0:length(ik)-1
  if Pmode % input to simulation is applied power
    if pk(k+1) > 0 % search for ik to meet power spec...
      % initialize search using prior value of cell voltage
      ik(k+1) = fmincon(@(x) ...
                  (pk(k+1) - x*simStep(x,Tk(k+1),cellState))^2,...
                 pk(k+1)/Vcell,[],[],[],[],0,Inf,[],options);
    else
      ik(k+1) = fmincon(@(x) ...
                  (pk(k+1) - x*simStep(x,Tk(k+1),cellState))^2,...
                 pk(k+1)/Vcell,[],[],[],[],-Inf,0,[],options);
    end
  end

  % now, attempt to update with user-supplied value of Iapp
  [Vcell,newCellState] = simStep(ik(k+1),Tk(k+1),cellState);
```

```
  % switch from current mode to constant-voltage mode
  if Vcell > Vmax
    ik(k+1) = fminbnd(@(x) (Vmax - simStep(x,Tk(k+1),cellState))^2,...
                min(0,ik(k+1)),0);
    [Vcell,newCellState] = simStep(ik(k+1),Tk(k+1),cellState);
  end
  if Vcell < Vmin
    ik(k+1) = fminbnd(@(x) (Vmin - simStep(x,Tk(k+1),cellState))^2,...
                0,max(0,ik(k+1)));
    [Vcell,newCellState] = simStep(ik(k+1),Tk(k+1),cellState);
  end
  cellState = newCellState;

  % Update information to user every 100 iterations
  if (rem(k,100) == 0),fprintf('%d %2.8f\n',k,ROMout.cellSOC(k+1)); end
end
```

Example: Charge from 50% to 100% SOC using CC/CV and CP/CV



4.10: Simulating battery packs

- The framework developed for CV or CP simulations can be extended to enable the simulation of battery packs comprising many cells.

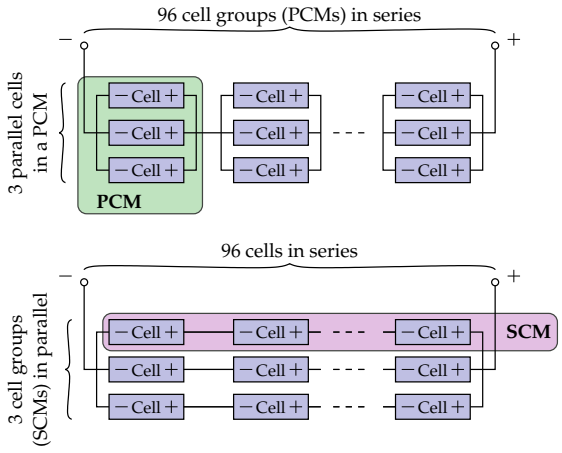
Series-connected cells

- Cells connected in series each experience the same current.
- If all cells have the same initial state and identical parameters, then all cells have exactly the same state and voltage for all time, so we need to simulate only one cell (the others will be identical).
- This is not generally true, however, so we can simulate all cells' dynamics by keeping state and model information for every cell, updating once per sample interval.
- Can also include inter-cell "interconnect" resistance term when computing pack voltage:

$$v_{\text{pack}}[k] = \left(\sum_{j=1}^{N_{\text{s}}} v_{\text{cell},j}[k] \right) - N_{\text{s}} R_{\text{interconnect}} i_{\text{app}}[k].$$

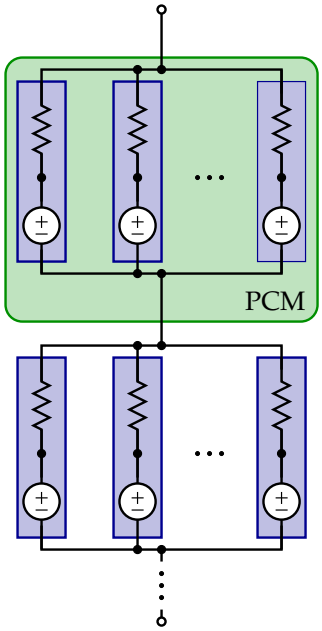
Parallel-connected cells and modules

- Series-connected packs are common for low-energy, high-power applications, such as HEV.
- High-energy applications usually have cells and even entire sub-packs that are connected in parallel.
- One extreme is PCM; other is SCM.
- If cells differ, we need to simulate all cells individually, but how?



Simulating PCM-based packs

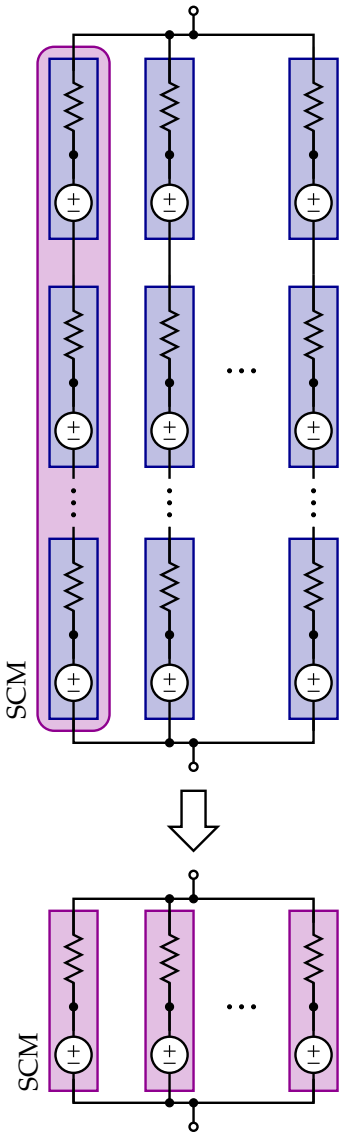
- Consider first PCM, recalling that cell voltage comprises a "fixed" part that does not depend on the present cell current, and a "variable" part that does depend on present cell current.
- We can model cells in parallel as drawn, where the voltage source in each branch is the fixed part, and the (nonlinear) resistor current is the variable part.



- By KVL, all terminal voltages must be equal; by KCL, the sum of the branch currents must equal the total battery pack current.
  - With ECM, could solve for all branch-current and voltage values;
  - Due to the nonlinear voltage equation with PBM, we cannot so must use nonlinear optimization again.
- We assume that total pack applied current $i_{\text{app}}[k]$ is given. Then,
  1. We "guess" the common cell voltage $v_{\text{cell}}[k]$ of all PCM cells.
  2. We compute the required $i_{\text{app},j}[k]$ for the individual cells to meet this $v_{\text{cell}}[k]$.
  3. We compute the total current $i_{\text{tot}}[k] = \sum_{j=1}^{N_{\text{p}}} i_{\text{app},j}[k]$.
  4. We revise $v_{\text{cell}}[k]$ and loop from step 2 until $i_{\text{tot}}[k] = i_{\text{app}}[k]$.
- This procedure gives us all branch currents and all PCM voltages.
- We update the model for each cell using its individual $i_{\text{app},j}[k]$ and proceed to the next iteration.

Simulating SCM-based packs

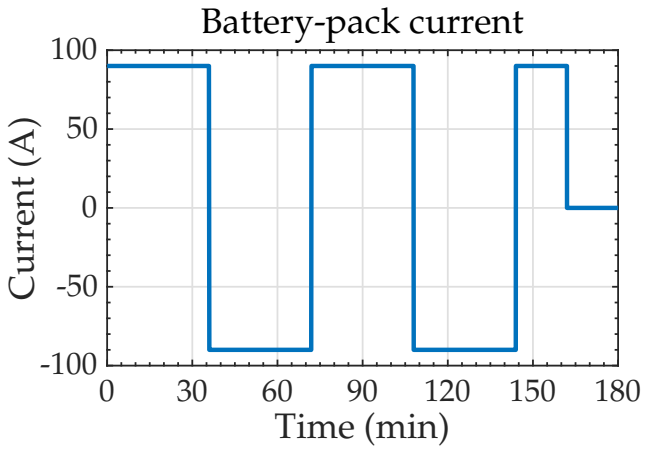
- Approach to simulating SCM is very similar to simulating PCM.
- Each cell in a SCM has its own "fixed" and "variable" parts.
- All "fixed" parts sum to give an equivalent voltage source; all "variable" parts sum to give an equivalent (nonlinear) resistance.
- Each SCM collapses to something that looks like a single high-voltage cell.
- We assume that total pack applied current $i_{\text{app}}[k]$ is given. Then,
  1. We "guess" the common pack voltage $v_{\text{pack}}[k]$.
  2. We compute the required $i_{\text{app},j}[k]$ for the individual branches of the SCM to meet this $v_{\text{pack}}[k]$. Note that the optimization to find $i_{\text{app},j}[k]$ is summing over all N_{s} series-connected cells (all the fixed and variable parts of all the cells) in module j.
  3. We compute the total current $i_{\text{tot}}[k] = \sum_{j=1}^{N_{\text{p}}} i_{\text{app},j}[k]$.
  4. We revise $v_{\text{pack}}[k]$ and loop from step 2 until $i_{\text{tot}}[k] = i_{\text{app}}[k]$.



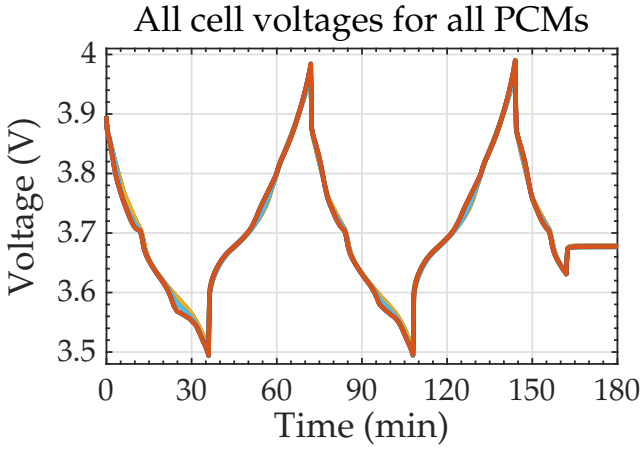
- This procedure gives us all branch currents and the pack voltage.
- We update the model for each cell using its individual $i_{\text{app},j}[k]$ and proceed to the next iteration.

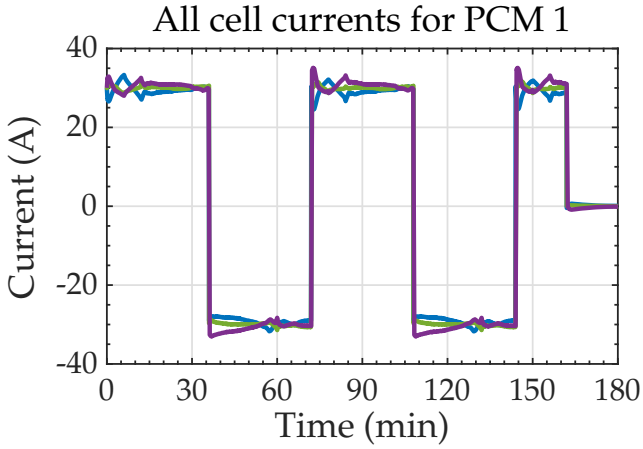
Example simulations of PCM-based and SCM-based packs

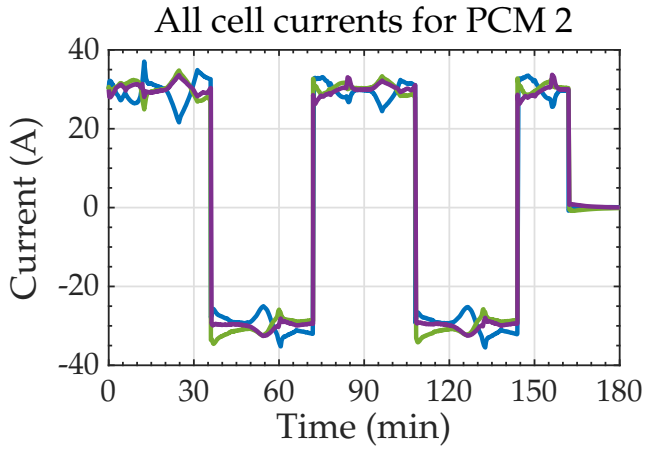
- We briefly consider a simulation comparison of a PCM-based battery pack versus an SCM-based pack. (Both packs had $N_\mathrm{s} = N_\mathrm{p} = 3$.)
- The nine cell ROMs were composed by slightly perturbing NMC30 parameter values.
- The pack input current cycled the cells at a nominal 1C rate twice between nominal SOCs of 80 % and 20 % before a final discharge to bring the nominal SOC of each cell to 50 %. The pack then rested.

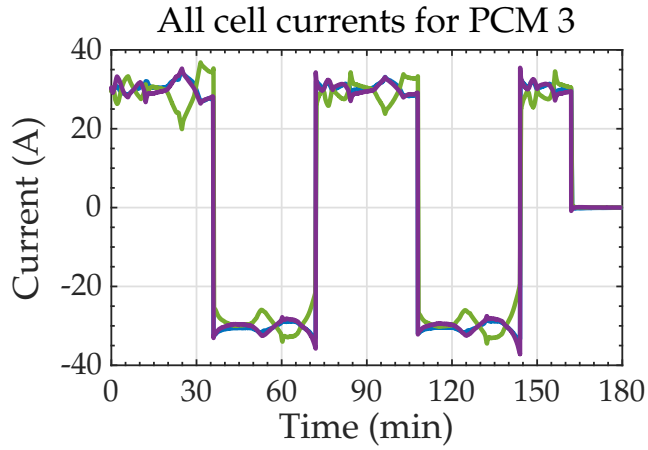


- PCM-based battery-pack results are shown below:

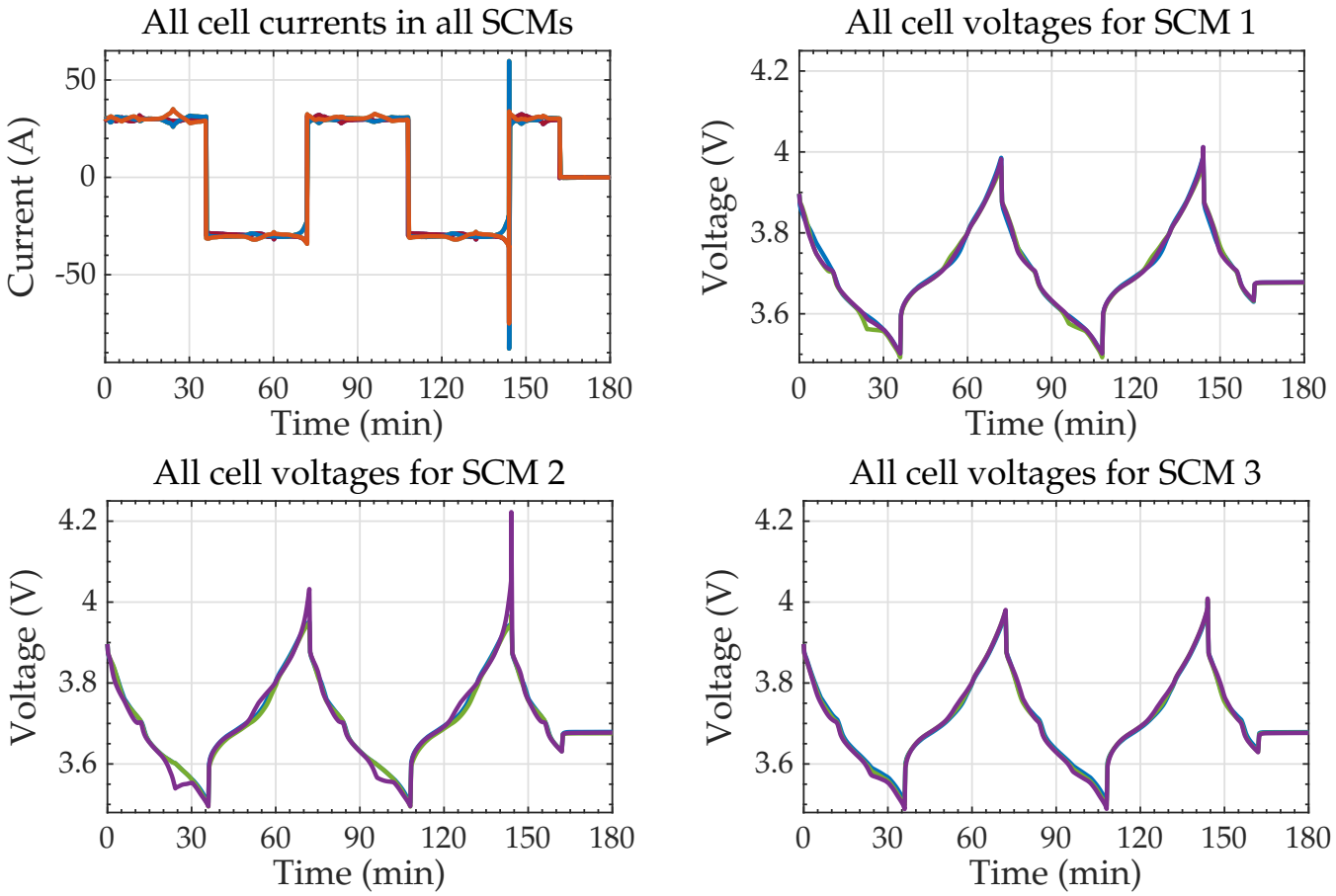






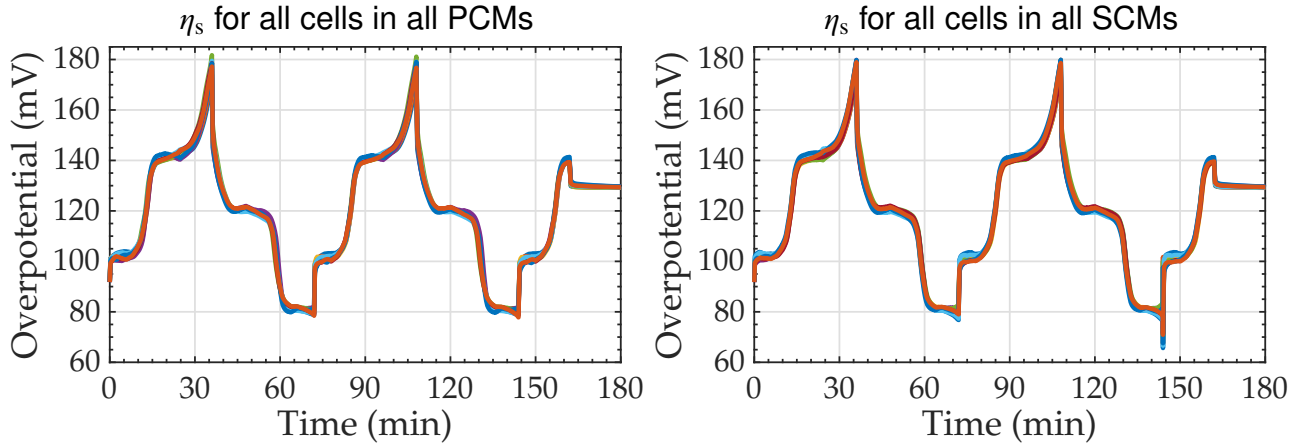


- The voltages of all cells within the same PCM are identical so there are only three distinct voltage profiles for this simulation.
- However, every cell's input current can be different, depending on each cell's present state and parameters, which is what we observe.
- SCM-based battery-pack results are shown below:



- In this case, currents for all cells within the same SCM are identical so there are only three distinct current profiles for this simulation.
- But, every cell's voltage can be different, depending on each cell's present state and parameters, which is what we observe.
- Comparing PCM/SCM results, we note an interesting artifact in SCM at $t = 144$ min, when battery-pack input current switches sign.
- By 144 min, transients caused by initializing all cells to the same SOC have died out, so this cycle's results differ from those of original cycle.

- There is no marked difference between PCM results near transitions at $t = 72$ min and 144 min; however, SCM has a large current spike.
- In a PCM, parallel cell connections tend to average out cell-to-cell variations since cell voltages are forced to be equal.
- In an SCM, there is no such averaging. The overall SCM voltages must be equal, but the individual cell voltages may be quite different.
- So, without real-time balancing, SCM appears to be more prone to larger cell-voltage swings than does PCM.
- The primary benefit of using physics-based models versus equivalent-circuit models is that we can also investigate cell internal variables.
- One variable of particular interest is the side-reaction overpotential η_s, which predicts the rate of SEI-layer growth and onset of Li plating.[5]
- The figures below compare η_s for PCM- and SCM-pack simulations.



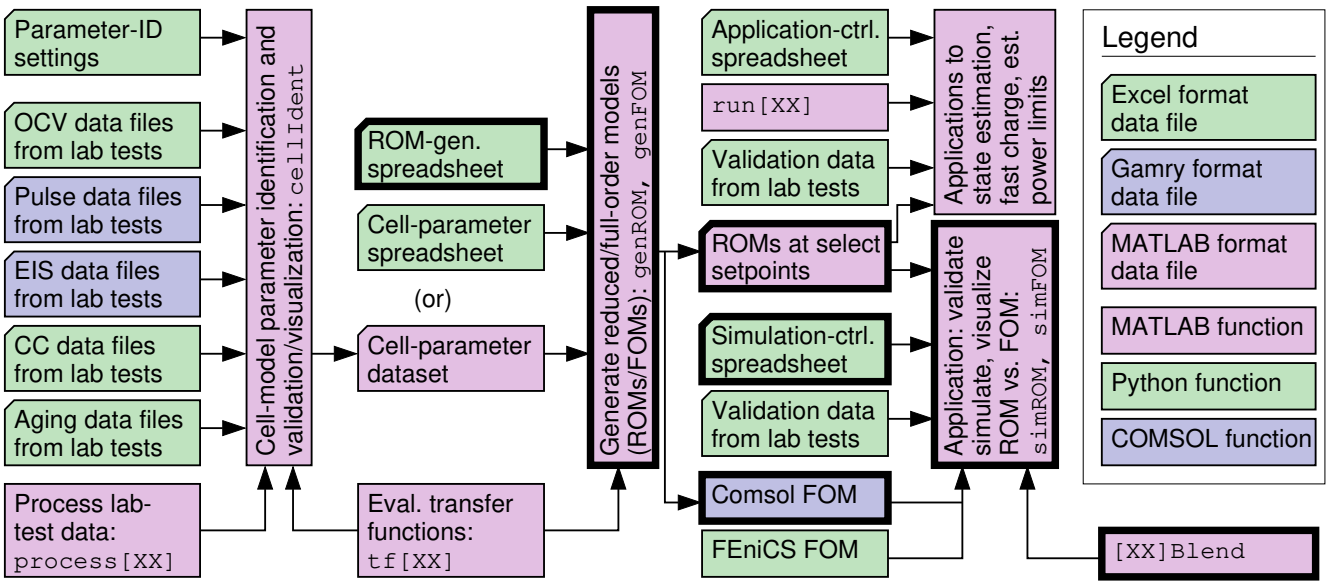
- Since the SCM does not naturally average cell differences the same way as does the PCM, we see greater fluctuation in η_s for SCM.
- Illustrates that SCM is more susceptible to Li plating than PCM since η_s has greater variation if cells have parameter differences.

[5] We will discuss side-reaction overpotential in more detail in Chap. 6. To avoid lithium plating, this value must never become negative.

4.11: MATLAB toolbox

- We return to the overview of the MATLAB toolbox from Ch. 2.
- The highlighted blocks correspond to topics in this chapter.

Lithium-ion toolbox for identification, simulation, battery-management-system application



- Tuning values for the xRAs are stored in custom-format Excel spreadsheets:

See *Instructions* tab for more information

Environmental			
Parameter	**Code Name**	**Value**	**Unit**
Operating Temperature	T	25	[°C]
Operating State of Charge Range	SOC	0:2:100	[%]

xRA Method			
Parameter	**Code Name**	**Value**	**Unit**
Final model sample period	Tsamp	1	[s]
Number of initial non-integrator poles	n	45	[unitless]
Number of final non-integrator poles	nfinal	8	[unitless]
HRA - Frequency samples in [0..1]	HRA_W	[0, logspace(-5,0,499)]	[rad/s]
HRA - Kmax for adding frequency aliases	HRA_Kmax	150	[unitless]
HRA - Oversizing parameter	HRA_ii	45	[unitless]

Transfer Function			
Parameter	**Code Name**	**Value**	**Unit**
	tflist	tfPhie(s,1:3,%s)	[n/a]
		tfPhis(s,1:2,%s)	[n/a]
		tfPhise(s,0:3,%s)	[n/a]
		tfIfdl(s,0:3,%s)	[n/a]
		tfIf(s,0:3,%s)	[n/a]
		tfThetae(s,0:3,%s)	[n/a]
		tfThetass(s,0:3,%s)	[n/a]

- Spreadsheet has sections for specifying set of ROM setpoints, tuning parameters for the xRA to be used, and for listing the transfer functions that are desired in the final model.
  - Note that the "1:3" or "1:2" (etc.) entries designate the $\tilde{x}$ locations at which the transfer functions are to be evaluated.
- MATLAB code `loadXRA.m` loads spreadsheet into workspace:

```
>> xraData = loadXRA('defaultHRA.xlsx')

xraData =

  struct with fields:

          T: 25
        SOC: [1 x 51 double]
      Tsamp: 1
          n: 45
     nfinal: 8
      HRA_W: [1 x 500 double]
   HRA_Kmax: 150
     HRA_ii: 45
         tf: {7 x 1 cell}
```

- To create a ROM (possibly containing many sub-ROMs at many different temperature/SOC setpoints), use `genROM.m`.

```
>> cellData = loadCellParams('cellDoyle.xlsx'); % load cell data
>> xraData = loadXRA('defaultHRA.xlsx'); % load XRA tuning data
>> ROM = genROM(cellData,xraData,'HRA'); % for example, using HRA

ROM =

  struct with fields:

    ROMmdls: [1 x 51 struct]
   cellData: [1 x 1 struct]
    xraData: [1 x 1 struct]
     tfData: [1 x 1 struct]
```

- ROM generation can take some time, so `genROM` displays status updates periodically.

- To execute a ROM, we must specify the initial cell SOC as well as a profile of current and temperature versus time.
- This is done using another custom-format Excel spreadsheet:

See *Instructions* tab for more information		
Time [s]	**Current [A]**	**Temperature [°C]**
0	0	25
1	-3.20782	25
2	0.490061	25
3	4.19956	25
4	6.88824	25
5	11.249	25
6	14.2022	25
7	16.0095	25
8	0.86154	25
9	4.03066	25
10	20.3944	25
11	7.32373	25
12	4.92615	25
13	-0.566978	25

Initial cell SOC [%]
60

This is a single UDDS charge-neutral cycle for a Doyle cell, max rate = 2C

- We load this spreadsheet using MATLAB function `loadInput.m`.

```
>> simData = loadInput('inputUDDS.xlsx')

simData =

  struct with fields:

    time: [1501 x 1 double]
    Iapp: [1501 x 1 double]
       T: [1501 x 1 double]
      Ts: 1
    SOC0: 60
```

- This can now be used with `simROM.m` to simulate the cell for this input profile.
  - This function can either perform output blending (`outBlend`), model blending (`mdlBlend`), or no blending (`nonBlend`):


```
ROMout = simROM(ROM,simData,'outBlend')

ROMout =

  struct with fields:

       blending: 'output-blending'
        negIfdl: [1501 x 2 double]
          negIf: [1501 x 2 double]
        negPhis: [1501 x 1 double]
       negPhise: [1501 x 2 double]
     negThetass: [1501 x 2 double]
       negIfdl0: [1501 x 1 double]
         negIf0: [1501 x 1 double]
        negEta0: [1501 x 1 double]
      negPhise0: [1501 x 1 double]
    negThetass0: [1501 x 1 double]
        posIfdl: [1501 x 2 double]
          posIf: [1501 x 2 double]
        posPhis: [1501 x 1 double]
       posPhise: [1501 x 2 double]
     posThetass: [1501 x 2 double]
       posIfdl3: [1501 x 1 double]
         posIf3: [1501 x 1 double]
        posEta3: [1501 x 1 double]
    posThetass3: [1501 x 1 double]
           Phie: [1501 x 4 double]
         Thetae: [1501 x 4 double]
          Vcell: [1501 x 1 double]
        cellSOC: [1501 x 1 double]
         negSOC: [1501 x 1 double]
         posSOC: [1501 x 1 double]
           time: [1501 x 1 double]
           Iapp: [1501 x 1 double]
              T: [1501 x 1 double]
           Ifdl: [1501 x 4 double]
             If: [1501 x 4 double]
           Phis: [1501 x 2 double]
          Phise: [1501 x 4 double]
        Thetass: [1501 x 4 double]
          xLocs: [1 x 1 struct]
       cellData: [1 x 1 struct]
```

- The output comprises all variables simulated at all points in time in the simulation.
  - This input profile was 1500 seconds long (1501 outputs recorded from $t = 0 \ldots 1500$);
  - Data were recorded at $\tilde{x} = \{0, 1, 2, 3\}$, as appropriate, for every variable of interest.
- If COMSOL is available with LiveLink for MATLAB, then the toolbox can also build COMSOL FOMs:

```
>> FOM = genFOM(cellData)

FOM =

COMSOL Model Object
Name: Lithium-ion cell
Tag: LiIonCell
Identifier: root
```

- The FOM can then be simulated as

```
[FOM,FOMout] = simFOM(FOM,simData)

FOM =

COMSOL Model Object
Name: Lithium-ion cell
Tag: LiIonCell
Identifier: root

FOMout =

  struct with fields:

        negIfdl: [1501 x 21 double]
          negIf: [1501 x 21 double]
         negIdl: [1501 x 21 double]
        negPhis: [1501 x 21 double]
       negPhise: [1501 x 21 double]
```



```
    negThetass: [1501 x 21 double]
      negIfdl0: [1501 x 1 double]
        negIf0: [1501 x 1 double]
       negIdl0: [1501 x 1 double]
     negPhise0: [1501 x 1 double]
   negThetass0: [1501 x 1 double]
          Phie: [1501 x 61 double]
        Thetae: [1501 x 61 double]
          xLocs: [1 x 1 struct]
       posIfdl: [1501 x 21 double]
         posIf: [1501 x 21 double]
        posIdl: [1501 x 21 double]
       posPhis: [1501 x 21 double]
     posPhise: [1501 x 21 double]
    posThetass: [1501 x 21 double]
      posIfdl3: [1501 x 1 double]
        posIf3: [1501 x 1 double]
       posIdl3: [1501 x 1 double]
     posPhise3: [1501 x 1 double]
   posThetass3: [1501 x 1 double]
             T: [1501 x 1 double]
          time: [1501 x 1 double]
          Iapp: [1501 x 1 double]
         Vcell: [1501 x 1 double]
        negSOC: [1501 x 101 double]
        posSOC: [1501 x 101 double]
          Ifdl: [1501 x 42 double]
            If: [1501 x 42 double]
           Idl: [1501 x 42 double]
          Phis: [1501 x 42 double]
         Phise: [1501 x 42 double]
       Thetass: [1501 x 42 double]
```

- The output comprises all variables simulated at all points in time in the simulation.
  - Note that variables are recorded at 61 $\tilde{x}$ locations across the cell.
  - There are 21 (each) locations in the negative and positive electrode-regions.
  - The remainder of the points are in the separator region.

- The `xLocs` variable records the value of these locations.

Where to from here?

- In this chapter, you have learned how to:
  - Convert TFs into ROMs using an improved method;
  - Add nonlinear corrections to simulate a cell using a ROM created for a single SOC/temperature linearization setpoint;
  - Simulate over a wide operational range using output blending;
  - Simulate constant-voltage and constant-power profiles;
  - Simulate multicell battery packs.
- So, now you can take a physical cell and go through all the steps to create a ROM that describes that cell and simulate the ROM to predict internal electrochemical variables and cell voltage.
- But, what to do with the ROM?
- The next topics move on to explore concepts in SOC, SOH, and SOF/SOP estimation, as well as fast-charge controls.

Appendix 4.a: Summary of variables

- $\boldsymbol{A}$, state-transition matrix of state-space model (for transient states only)
- $\mathfrak{A}$, state-transition matrix of state-space model (for transient plus integrator states)
- $\boldsymbol{B}$, input matrix of state-space model (for transient states only)
- $\mathfrak{B}$, input matrix of state-space model (for transient plus integrator states)
- $\boldsymbol{C}$, output matrix of state-space model (for transient states only)
- $\mathfrak{C}$, output matrix of state-space model (for transient plus integrator states)
- $\boldsymbol{D}$, direct-feedthrough matrix of state-space model (for models having only transient states and also for models having transient and integrator states)
- $\boldsymbol{G}(s)$, transfer function implemented by state-space model
- $\bar{\boldsymbol{G}}(s)$, transfer function $\boldsymbol{G}(s)$ combined with ZOH
- i, the oversizing parameter used by the HRA
- $k_{\max}$, number of aliased frequency spectra to use when converting continuous-time to discrete-time frequency response
- $\mathcal{O}_i$, extended observability matrix, used by HRA
- $\mathcal{T}_i$, a Toeplitz matrix, part of the development of the HRA
- T_s, sample period of discrete-time model
- $\boldsymbol{x}[k]$, time-varying state of state-space model (transient states only)
- $\mathfrak{X}[k]$, time-varying state of state-space model (transient plus integrator states)
- $\boldsymbol{y}[k]$, linear outputs from state-space model

Appendix 4.b: Nonlinear corrections to linear model

- This appendix addresses how to apply nonlinear corrections to linear ROM outputs to improve predictions of electrochemical variables.

Interfacial lithium flux

- The linear outputs of the ROM for $i_{\mathrm{f+dl}}[\tilde{x}, k]$, $i_{\mathrm{f}}[\tilde{x}, k]$, and $i_{\mathrm{dl}}[\tilde{x}, k]$ are unbiased predictions to the true interface lithium fluxes. That is, no additional corrections are needed.

Solid surface stoichiometry

- The ROM can produce debiased outputs $\tilde{\theta}_{\mathrm{ss}}[\tilde{x}, k]$, from which the nonlinear prediction is made via:

$$\theta_{\mathrm{ss}}[\tilde{x}, k] = \tilde{\theta}_{\mathrm{ss}}[\tilde{x}, k] + \theta^{\mathrm{r}}_{\mathrm{s},0}. \tag{4.10}$$

Phase potential difference

- According to the original definition, the phase-potential difference variable can be computed as:

$$\phi^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x}, k] = \tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x}, k] + U^{\mathrm{r}}_{\mathrm{ocp}}(\theta^{\mathrm{r}}_{\mathrm{s},0}),$$

  where $\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x}, k]$ is obtained directly from the system linear output, and the electrode open-circuit potential function is evaluated at the initial local state of charge that is previously determined from Eq. (4.6).

- However, performance can be improved by looking deeper at what is actually happening.

- The transfer function $\widetilde{\Phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}(z, s) / I_{\mathrm{app}}(s)$ has a pole at the origin, which is removed prior using the HRA to give $\left[\widetilde{\Phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}(z, s)\right]^* / I_{\mathrm{app}}(s)$.

- If there is no double layer in the model, the integrator response could be added back manually as:

$$\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k] = \left[\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k]\right]^{*} + \tilde{\phi}^{\mathrm{res0,r}}_{\mathrm{s\text{-}e}} \times x_0[k],$$

  where $x_0[k]$ is the integrator state of the state-space model.

- However, recall the integrator residues of $\widetilde{\Phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}(\tilde{x},s)/I_{\mathrm{app}}(s)$,

$$\mathrm{res}^{\mathrm{n}}_0 = \frac{-\left|\theta^{\mathrm{n}}_{100}-\theta^{\mathrm{n}}_0\right|\,[U^{\mathrm{n}}_{\mathrm{ocp}}]'}{3600Q-\bar{C}^{\mathrm{n}}_{\mathrm{dl,eff}}\left|\theta^{\mathrm{n}}_{100}-\theta^{\mathrm{n}}_0\right|\,[U^{\mathrm{n}}_{\mathrm{ocp}}]'}$$

$$\mathrm{res}^{\mathrm{p}}_0 = \frac{\left|\theta^{\mathrm{p}}_{100}-\theta^{\mathrm{p}}_0\right|\,[U^{\mathrm{p}}_{\mathrm{ocp}}]'}{3600Q-\bar{C}^{\mathrm{p}}_{\mathrm{dl,eff}}\left|\theta^{\mathrm{p}}_{100}-\theta^{\mathrm{p}}_0\right|\,[U^{\mathrm{p}}_{\mathrm{ocp}}]'}.$$

- We notice that the residuals of the two unstable transfer functions are linked as:

$$\tilde{\phi}^{\mathrm{res0,r}}_{\mathrm{s\text{-}e}} = [U^{\mathrm{r}}_{\mathrm{ocp}}]' \times \tilde{\theta}^{\mathrm{res0,r}}_{\mathrm{ss}}.$$

- Therefore, we can write:

$$\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k] = [\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k]]^{*} + [U^{\mathrm{r}}_{\mathrm{ocp}}]' \times \tilde{\theta}^{\mathrm{res0,r}}_{\mathrm{ss}} \times x_0[k],$$

- Substituting the relationship from Eq. (4.5) yields:

$$\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k] = [\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k]]^{*} + [U^{\mathrm{r}}_{\mathrm{ocp}}]' \left(\theta^{\mathrm{r}}_{\mathrm{s,avg}}[k] - \theta^{\mathrm{r}}_{\mathrm{s},0}\right).$$

- We recognize the rightmost terms as a linearization of $U^{\mathrm{r}}_{\mathrm{ocp}}[\theta^{\mathrm{r}}_{\mathrm{s,avg}}[k]]$.

- Therefore, we achieve more accurate results if we compute:

$$\phi^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k] = [\tilde{\phi}^{\mathrm{r}}_{\mathrm{s\text{-}e}}[\tilde{x},k]]^{*} + U^{\mathrm{r}}_{\mathrm{ocp}}[\theta^{\mathrm{r}}_{\mathrm{s,avg}}[k]]. \tag{4.11}$$

Potential in solid

- The ROM can directly compute debiased $\tilde{\phi}^{\mathrm{r}}_{\mathrm{s}}[\tilde{x},k]$.

- To use the nonlinear prediction, we must add back the term at the current collector:

$$\phi_{\mathrm{s}}^{\mathrm{n}}[\tilde{x},k] = \tilde{\phi}_{\mathrm{s}}^{\mathrm{n}}[\tilde{x},k] + \phi_{\mathrm{s}}^{\mathrm{n}}[0,k] = \tilde{\phi}_{\mathrm{s}}^{\mathrm{n}}[0,k] + 0 \tag{4.12}$$

$$\phi_{\mathrm{s}}^{\mathrm{p}}[\tilde{x},k] = \tilde{\phi}_{\mathrm{s}}^{\mathrm{p}}[\tilde{x},k] + \phi_{\mathrm{s}}^{\mathrm{p}}[3,k] = \tilde{\phi}_{\mathrm{s}}^{\mathrm{p}}[\tilde{x},k] + v_{\mathrm{cell}}[k], \tag{4.13}$$

where $v_{\mathrm{cell}}[k]$ is computed from Eq. (4.7).

Potential in electrolyte

- The ROM can directly compute debiased $\tilde{\phi}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k]$.
- To compute the nonlinear prediction, notice:

$$\begin{aligned}\phi_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] &= \tilde{\phi}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] + \phi_{\mathrm{e}}^{\mathrm{n}}[0,k] \\ &= \tilde{\phi}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] + \underbrace{\phi_{\mathrm{s}}^{\mathrm{n}}[0,k]}_{0} - \phi_{\mathrm{s\text{-}e}}^{\mathrm{n}}[0,k] \\ &= \tilde{\phi}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] - \phi_{\mathrm{s\text{-}e}}^{\mathrm{n}}[0,k],\end{aligned}$$

where $\phi_{\mathrm{s\text{-}e}}^{\mathrm{n}}[0,k]$ is found via Eq. (4.11).

- As a result,

$$\begin{aligned}\phi_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] &= \tilde{\phi}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] - \phi_{\mathrm{s\text{-}e}}^{\mathrm{n}}[0,k] \\ &= \tilde{\phi}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] - [\tilde{\phi}_{\mathrm{s\text{-}e}}^{\mathrm{n}}[0,k]]^{*} - U_{\mathrm{ocp}}^{\mathrm{n}}(\theta_{\mathrm{s,avg}}^{\mathrm{n}}[k]).\end{aligned} \tag{4.14}$$

Lithium stoichiometry in electrolyte

- The ROM can directly compute debiased $\tilde{\theta}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k]$.
- The actual electrolyte concentration ratio prediction is found by:

$$\theta_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] = \tilde{\theta}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] + \theta_{\mathrm{e},0} = \tilde{\theta}_{\mathrm{e}}^{\mathrm{r}}[\tilde{x},k] + 1, \tag{4.15}$$

where we recall that $\theta_{\mathrm{e},0} = 1$.

