User's manual to CAFES

A.S. Avaro, J.G. Santiago, 06/30/2022

CAFES (Client-based Application for Fast Electrophoresis Simulation) is a web-based online tool for fast simulations of complex electrophoresis processes (e.g., peak and plateau-mode isotachophoresis). It requires no compilation nor installation.

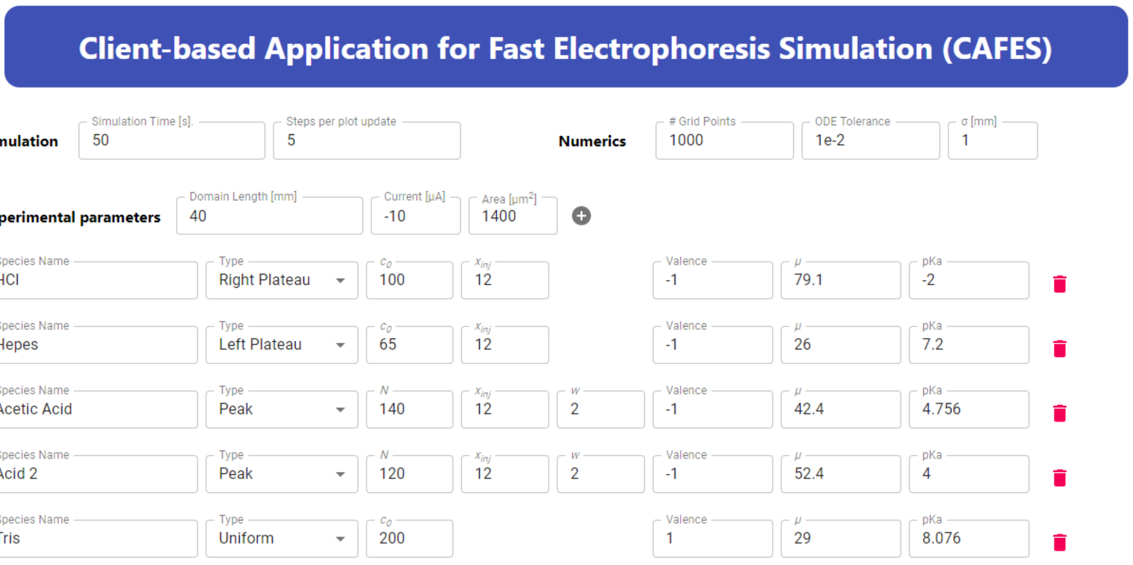
Reference: Avaro, A.S., Sun, Y., Jiang, K., Bahga, S.S., Santiago, J.G. "Web-based open-source tool for isotachophoresis", *Analytical Chemistry*, **2021**, 93, 47, 15768–15774, https://doi.org/10.1021/acs.analchem.1c03925.

A. Access

CAFES is available at https://microfluidics.stanford.edu/cafes. You can also find a link on the Stanford Microfluidics Laboratory website, under "Resources." (https://microfluidics.stanford.edu/resources). The tool is divided into three panels: simulation parameters, species database and results.

B. Simulation parameters

The first panel presents the simulation parameters. We detail here all the simulation variables and the corresponding fields in the GUI (**Fig. 1**).


Client-based Application for Fast Electrophoresis Simulation (CAFES)

Simulation — Simulation Time [s]: 50; Steps per plot update: 5

Numerics — # Grid Points: 1000; ODE Tolerance: 1e-2; σ [mm]: 1

Experimental parameters — Domain Length [mm]: 40; Current [μA]: -10; Area [μm²]: 1400

Species Name	Type	c_0 / N	x_{inj}	w	Valence	μ	pKa
HCl	Right Plateau	c_0: 100	12		-1	79.1	-2
Hepes	Left Plateau	c_0: 65	12		-1	26	7.2
Acetic Acid	Peak	N: 140	12	2	-1	42.4	4.756
Acid 2	Peak	N: 120	12	2	-1	52.4	4
Tris	Uniform	c_0: 200			1	29	8.076

Figure 1: Graphical user interface (GUI) of CAFES: Simulation parameters.

B.1 Simulation

- Simulation Time
  - *Description*: Total physical time that you will simulate (and not the time it will take for the program to perform the simulation as this varies with complexity and the computer you are using). Note that the simulation uses an adaptive time advancement scheme, and it determines the time step for the computation. Hence, the time step cannot be customized by the user.
  - *Units*: second [s].
  - *Input type*: Positive float.
- Steps per plot update
  - *Description*: This is the number of computational steps before each updated graphic/plot that will be displayed on the screen in the browser. This helps

determine the refresh rate of the results panel. If this value is set to n, then the results plotted in the result panel will be updated once every n computed time steps. Small values produce a smooth animation but slow down the simulation. I recommend starting with $n = 5$.
  - *Units*: None.
  - *Input type*: Positive integer.

B.2 Numerical parameters

- # Grid Points
  - *Description*: Number of grid points. Increase this value for more resolution and more precise results. However, this can slow down the simulation.
  - *Units*: None.
  - *Input type*: Positive integer.
- ODE Tolerance
  - *Description*: Absolute tolerance for the ODE integration. Increase this value for faster simulations. Typical values range from 10^{-4} to 10^{-2}. Setting this value lower than 10^{-4} is not recommended as it may impair the correct integration of the system.
  - *Units*: None.
  - *Input type*: Positive float.
- σ
  - *Description*: Interface width. This variable corresponds to σ in **Eq. (3)** and **(5)** in **Ref.** that describe analytically the initial concentration profile. See **Fig. 2**.
  - *Units*: millimeter [mm].
  - *Input type*: This value is read-only and set to 1 mm.

B.3 Experimental parameters

- Domain length
  - *Description*: Physical domain length (the actual physical length you will simulate).
  - *Units*: millimeter [mm].
  - *Input type*: Positive float.
- Current
  - *Description*: Ionic current applied to the system.
  - *Units*: microampere [µA].
  - *Input type*: Positive or negative float.
- Area
  - *Description*: Cross-sectional area of the system. This value is used to compute all fluxes. For example, a NS12A Perkin Elmer chip with a "D" shaped channel that is 20 µm deep and 70 µm wide has an area of 1630 µm^2.
  - *Units*: micrometer squared [µm^2].
  - *Input type*: Positive float.

B.4 Chemical species

CAFES is a tool for multi-species transport. A given species can be added using the "+" icon next to the "Area" box or using the electrolyte database which we will discuss next. A species can be removed from the simulation by clicking on the pink trash icon at the far right of the corresponding line. Each species is characterized by the following values.

- Species Name
  - *Description*: Name tag for the species (used only for identification purposes).
  - *Units*: None.
  - *Input type*: String.
- Type
  - *Description*: Injection type. This variable controls the initial concentration profile type. Samples can be injected as “Left/Right plateau” (semi-infinite injection, see **Eq. (3)** in **Ref.**), “Peak” (finite injection, see **Eq. (5)** in **Ref.**) or “Uniform” (constant concentration on the whole domain).
  - *Units*: None.
  - *Input type*: Drop-down menu.
- c_0
  - *Description*: For “Left/Right Plateau” and “Uniform” only: Initial concentration (see **Eq. (3)** in **Ref.**).
  - *Units*: millimolar [mM].
  - *Input type*: Positive float.
- N
  - *Description*: For “Peak” only: Initial amount of injected species (see **Eq. (4)** in **Ref.**).
  - *Units*: picomole [pmol].
  - *Input type*: Positive float.
- x_{inj}
  - *Description*: For “Left/Right Plateau”: location of the initial concentration boundary. For “Peak”: location of the center of the peak of the initial concentration profile. See **Fig. 2**.
  - *Units*: millimeter [mm].
  - *Input type*: Positive float.
- w
  - *Description*: For “Peak” type injection zone only: Thickness of the initial concentration boundary at the sample injection point. Note this value can be larger or smaller than σ. Such cases will result in respectively top-hat or peak distributions (see **Eq. (5)** in **Ref.**). See **Fig. 2**.
  - *Units*: millimeter [mm].
  - *Input type*: Positive float.
- Valence
  - *Description*: Valence of the species.
  - *Units*: None.
  - *Input type*: Integer. For multivalent families of species, the valences must be separated by commas and should not discontinue.
- μ
  - *Description*: Absolute mobility of the species.
  - *Units*: 10^{-9} meter squared per volt second [10^{-9} m^2/(V·s)].
  - *Input type*: Positive float. For families of species, the mobilities must be separated by commas and be ordered in the same way than the corresponding valences.

    Example values (fully ionized): Cl^{-}: 79.1×10^{-9} m^2/(V·s), HEPES: 26×10^{-9} m^2/(V·s), DNA: 45×10^{-9} m^2/(V·s) (Stellwagen *et al.*, *Biopolymers*, **1997**, 42, 687-703).

- pKa
    - *Description*: pKa of the species.
    - *Units*: None.
    - *Input type*: Positive or negative float. For families of species, the pKa must be separated by commas and be ordered in the same way than the corresponding valences.

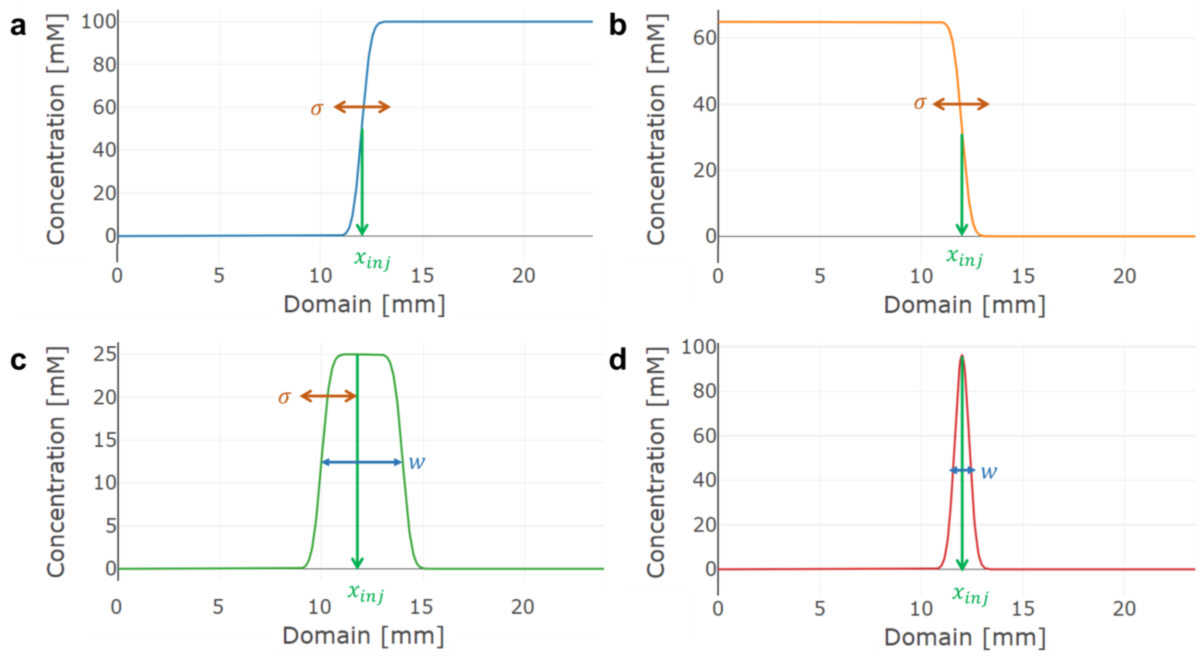
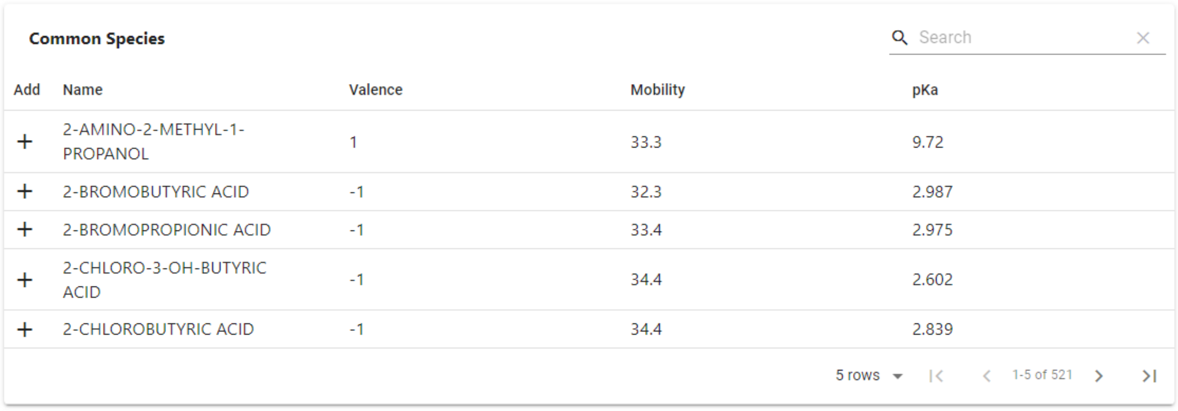


Figure 2: Injection types. Shown are concentration profiles (solid lines) for three different injection types: **a** Right plateau **b** Left plateau **c** Peak with $w > \sigma$ **d** Peak with $w < \sigma$. x_{inj}, σ and w are respectively the injection boundary location (**a** and **b**)/injection center location (**c** and **d**), interface width and thickness of the concentration boundary (**c** and **d**).

C. Electrolyte database

CAFES offers a database of 521 common species (303 weak acids, 161 weak bases, and 57 ampholytes) and their relevant physicochemical properties (valences, mobilities and pKa). This data is based on the work of Hirokawa *et. al* (Hirokawa, T., Kiso, Y. *J. Chromatogr. A* **1982**, 252, 33−48).



Common Species

Search

Add	Name	Valence	Mobility	pKa
+	2-AMINO-2-METHYL-1-PROPANOL	1	33.3	9.72
+	2-BROMOBUTYRIC ACID	-1	32.3	2.987
+	2-BROMOPROPIONIC ACID	-1	33.4	2.975
+	2-CHLORO-3-OH-BUTYRIC ACID	-1	34.4	2.602
+	2-CHLOROBUTYRIC ACID	-1	34.4	2.839

5 rows 1-5 of 521

Figure 3: Graphical user interface (GUI) of CAFES: Common species database.

The “+” icon on the extreme left of each row (in the “Add” column) adds the corresponding species in the simulation and automatically fills the relevant fields.

A search bar on the top right of the panel allows to search in this database. Type (at least) the first two letters of the species to add in the simulation to filter the database by name. The database can also be ordered by name, valence, mobility or pKa by clicking on the respective column headers. Multiple clicks revert the order.

D. Results

D.1 Start the simulation

Once the simulation parameters are complete, initiate the simulation using the “Start” button (see **Fig. 4**). Once the simulation has started, it can be paused at any time using the “Pause” button. A paused simulation can be resumed by clicking the “Start” button again. If all parameters are correctly input, the current simulated time t_s should appear and increase in the title of the plots: “Concentration / pH @ t = t_s”

D.2 Store the simulation parameters

Any simulation parameter profile (that includes all input parameters presented in the *Simulation Parameters* section for a given simulation) can be saved and downloaded as a JavaScript Object Notation file (JSON) using the “Save config” button. Later, this file can be uploaded in CAFES using the “Load config” button and all the simulation parameters will be automatically filled using the parameters stored in the JSON file.

D.3 Plotly interface

Fig. 4 shows the direct output of the simulation that is shown in real time (i.e., while the simulation is running) in the GUI. The results panel consists in two subpanels. The first subpanel shows the concentration profiles at $t = t_s$. Each concentration profile is labeled, and the corresponding species can be found in the legend on the right of the subpanel. A simple click on an entry of the legend hides the corresponding concentration profile. Click again on the greyed label to show the plot again. Double click on a label in the legend to show the corresponding profile only (and hide all other plots). The second (bottom) subpanel shows the pH and electric field (in V/mm) profiles. The corresponding legend entry work in the same way than the concentration profiles to hide/show these plots.

The Plotly interface (for both subpanels) has several functions available on the top right of the first subpanel. All functions have a corresponding icon (specified here in parenthesis). This includes zoom on a custom area (magnifying lens), pan over the domain (double-arrowed cross), zoom in and out (“+” and “-”), autoscale (framed X-shaped double-arrowed cross), reset axes (home), toggle/deactivate spike lines on both axes (top right corner), show the closest data on hover (single arrow), and compare data on hover (two arrows).

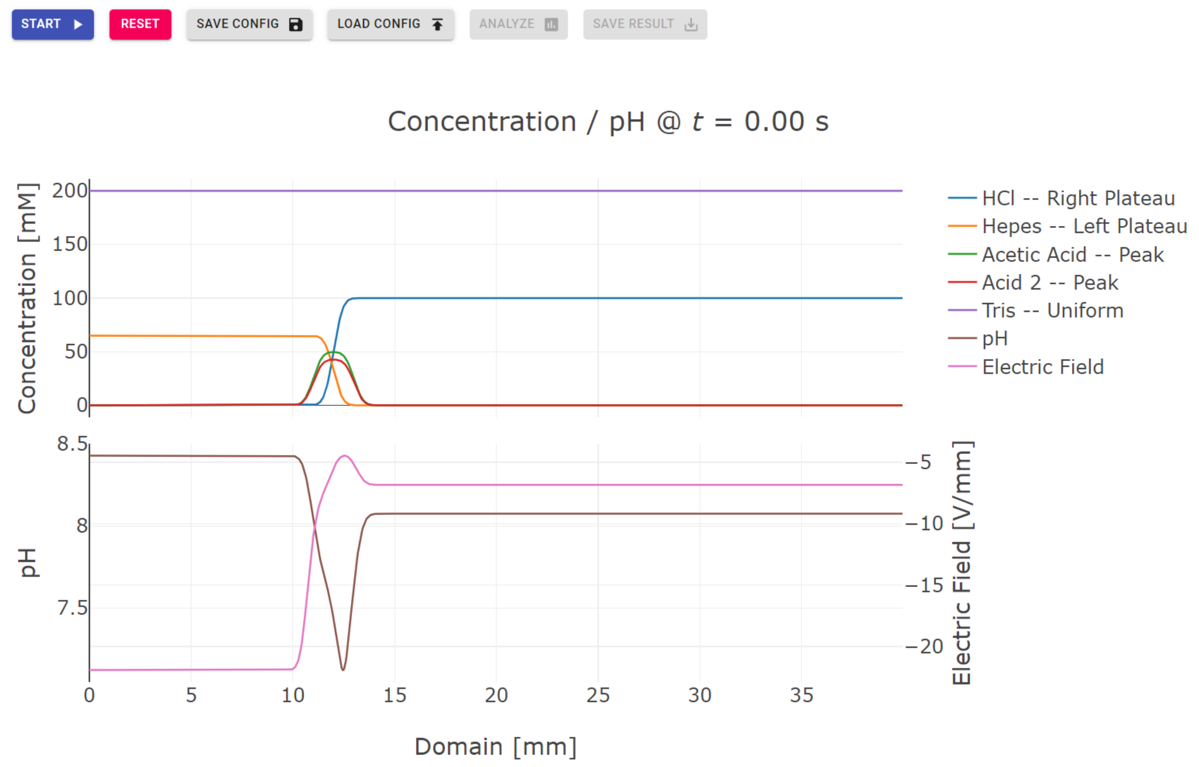


Figure 4: Graphical user interface (GUI) of CAFES: Results panel.

D.4 Analyze mode

Once the simulation has been computed up to a satisfactory time (i.e. up to the input simulation time or paused at some arbitrary time), the results of the simulation for all time can be shown using the “Analyze” button. Note that a paused simulation cannot resume after the “Analyze” mode is activated. In the “Analyze” mode, any time step in the simulation can be plotted using the “Simulation Playback” slider.

D.5 Download simulation results

The simulations results can be downloaded as a compressed ZIP file using the “Save result” button. We detail post-processing steps for this file in the next section of this manual and provide a Python code to transform the result file into a MATLAB data file.

D.6 Reset the simulation

To run the simulation with new simulation parameters, click on the “Reset” button. All simulation results will then be discarded and cannot be recovered.

E. Data post-processing and MATLAB conversion

It is convenient to import the results of this simulation in MATLAB (for plotting or further computations/simulations). You can also use the other Python scripts stored in the "python" folder on GitHub to manipulate the simulation data directly. Note that Python is completely free and open source. It is probably already installed on your computer. If not, you can download it here https://www.python.org/downloads/. Make sure to have all the required modules installed (using pip: https://pip.pypa.io/en/stable/installation/).

E.1 Conversion to MATLAB data file

We here specify the protocol to convert the output ZIP file into a MATLAB data file (.mat).

- Extract the downloaded ZIP into a "result" folder on your computer.
- Create a first Python file using the code 1 provided at the end of this section. Name this file "matlab_converter.py". Alternatively, download matlab_converter.py from the "python" folder in the following GitHub repository: https://github.com/alvinsunyixiao/itp-websim. The Python script should be stored at the same directory than the "Simulation Results" folder (e.g., directly in the "result" folder).
- Create a second Python file using code 2 provided at the end of this section. Name this file "utils.py". Alternatively, download utils.py from the "python" folder in the following GitHub repository: https://github.com/alvinsunyixiao/itp-websim. The Python script should be stored at the same directory than the "Simulation Results" folder (e.g., directly in the "result" folder).
- Open a terminal in the "result" folder. To do, in Windows, open the "result" folder in the File Explorer, then right-click and open in terminal. Alternatively open PowerShell and change the location to the "result" folder.
- Run the following command line:

  ```
  python matlab_converter.py -filename "./Simulation Results"
  ```

  Note: If you would like to place the "result" file in another location, you should specify the corresponding directory in this command line after -filename.
- The script will now prompt you to enter the name of a MATLAB table file for the analyzed data. Enter the desired name of the MATLAB table and press Enter.
- The newly created MATLAB table (.mat) file should appear within the folder where the Python script is located. This file contains the time array, the spatial grid, the electrolyte concentration profiles at all times, the hydronium concentration at all times, and the electric field profile.
- Open MATLAB. Type "load" followed by the chosen name of the MATLAB data file. Let us denote N_x the number of grid points, N_t the number of time steps in the simulation and N_c the number of simulated species. The following variables appear in your workspace:
    - `grid`
        - *Description*: x-locations of the grid points.
        - *Dimensions*: N_x
    - `time`
        - *Description*: Time array. Stores the physical time of each time step of the simulation.
        - *Dimensions*: N_t
    - `ctable`
        - *Description*: Concentration profiles at all times.
        - *Dimensions*: $N_t \times N_c \times N_x$

- `cH`
  - *Description*: Concentration profile of H^+ (in M) at all times. -log10(cH) yields the pH profile at all times.
  - *Dimensions*: $N_t \times N_x$
- `efield`
  - *Description*: Electric field profile (in V/m) at all times.
  - *Dimensions*: $N_t \times N_x$

E.1.a Code 1: matlab_converter.py

```
from utils import SimResult
import scipy.io
import argparse

if __name__ == "__main__":
    """
    Converts the result ZIP file into MATLAB (.mat) data file
    """
    parser = argparse.ArgumentParser(
        description='Converts CAFES results into MATLAB table.')
    # Input: Location of the result file
    parser.add_argument('-filename', type=str,
                        help='Location of the CAFES result file')
    args = parser.parse_args()
    filename = args.filename

    # Input: Name of the MATLAB data file
    filetag = input('Name of the MATLAB data file : ')
    # Import results
    sim_results = SimResult.from_directory(filename)

    # Save .mat file
    scipy.io.savemat(filetag+".mat", {
        'ctable': sim_results.concentration_tsn,
        'cH': sim_results.cH_tn,
        'efield': sim_results.efield_tn,
        'time': sim_results.time_t,
        'grid': sim_results.grid_n,
    })
```

E.1.b Code 2: utils.py

```
import base64
import json
import os
import numpy as np

class SimResult:
    """ class abstraction of simulation results """
    def __init__(self, inputs, grid_n, concentration_tsn, cH_tn,
efield_tn, time_t):
        """
        Args:
            inputs:             all simulation inputs
            grid_n:             spatial discretization grid of the
channel domain
            concentration_tsn:  all time slices of concentration
matrix in [mole / m^3]
```

```
            cH_tn:              all time slices of Hydrogen ion
concentration in [mole / liter]
            efield_tn:          all time slices of electric field in
[V / m]
            time_t:             simulated time steps
        Note:
            <variable name>_xyz is a naming convention for an array
of shape (x, y, z)
            Dimension Definition:
                n:  number of grid points
                s:  number of species
                t:  number of time steps
        """
        self.inputs = inputs
        self.grid_n = grid_n
        self.concentration_tsn = concentration_tsn
        self.cH_tn = cH_tn
        self.efield_tn = efield_tn
        self.time_t = time_t
    @staticmethod
    def from_directory(directory):
        """
        Args:
            directory: directory that stores the uncompressed result
files
        Returns:
            a parsed SimResult object containing all the simulation
result data as well
            as experimental setup
        """
        input_file = os.path.join(directory, "inputs.json")
        concentration_tsn_file = os.path.join(directory,
"concentration_tsn.bin")
        cH_tn_file = os.path.join(directory, "cH_tn.bin")
        efield_tn_file = os.path.join(directory, "efield_tn.bin")
        time_t_file = os.path.join(directory, "time_t.bin")

        with open(input_file, 'r') as f:
            inputs = json.load(f)

        num_grids = inputs["numGrids"]
        num_species = len(inputs["species"])

        concentration_tsn = np.fromfile(
            concentration_tsn_file, dtype=np.float32).reshape(-1,
num_species, num_grids)
        cH_tn = np.fromfile(cH_tn_file, dtype=np.float32).reshape(-1,
num_grids)
        efield_tn = np.fromfile(efield_tn_file,
dtype=np.float32).reshape(-1, num_grids)
        time_t = np.fromfile(time_t_file, dtype=np.float32)

        return SimResult(
            inputs=inputs,
            grid_n=np.linspace(0, inputs['domainLen'],
inputs['numGrids'], endpoint=False),
            concentration_tsn=concentration_tsn,
            cH_tn=cH_tn,
            efield_tn=efield_tn,
            time_t=time_t
        )
```