Ion Permeation Analyser Documentation & Usage Guide

> **Repository:** github.com/s-crowhurst/ion_permeation_analyser
> **Example Script:** `example.py`
> **Modules:** `io.py`, `segmentation.py`, `membrane.py`, `crossings.py`

Overview

Ion Permeation Analyser automates the identification of **ion permeation events** through membrane channels using MD simulation trajectories. It segments trajectories to deal with periodic boundaries, locates the start and end of the channel (membrane bounds), and counts complete crossing events through the channel.

ⓘ Core Algorithm Summary

1. PBCs are dealt with by **segmenting trajectories** into continuous segments by splitting them at any large jumps.
2. The ion density profile across a given axis is used to determine where the membrane is. Most ions exist on the intracellular or extracellular side, very few exist in the channel vestibule. Thus, inflection points of the density curve are used to define the **start** and **end** of the channel.
3. Segments which pass through **both** the start and the end are counted as permeating segments, and their direction of permeation is recorded.

Repository Structure

```
ion_permeation_analyser/
├── example.py # Example usage script
├── ion_analysis/
│ ├── **init**.py
│ ├── io.py # Input/output and data reading
│ ├── segmentation.py # Handles PBCs and trajectory segmentation
```

```
| ├── membrane.py # Detects membrane boundaries
| ├── crossings.py # Determines ion permeation events
```

Installation

> You can clone or download the repository manually

```
# Clone from GitHub
git clone https://github.com/s-crowhurst/ion_permeation_analyser.git

# OR

# Download the ZIP from GitHub and unpack to your working directory.
unzip ion_permeation_analyser.zip
```

> ⚠ **Warning**
>
> You **must** ensure dependencies are installed (e.g., `MDAnalysis`, `numpy`, `pandas`, `scipy`).

How It Works

1) `io.py` Reading Trajectories

> **Summary**
>
> Loads topologies and trajectories, extracts ion positions, times, and IDs, and merges them into a single **MultiIndex DataFrame**.

Input: list of `.tpr` and `.xtc` files
Output: MultiIndex DataFrame containing coordinates per simulation per frame per ion.

2) `segmentation.py` Handling PBCs

> **Summary**

Ions which leave the periodic cell are wrapped back in on the opposite side, causing non-physical large jumps. This module identifies and splits trajectories at these jumps to ensure all segments are continuous.

Algorithm:

1. Compute displacements between consecutive frames.
2. Flag unusually large jumps using a threshold that is either provided or calculated using a standard deviation threshold:

```
threshold = mean(|disp|) + std_thresh * std(|disp|)
```

> Default behaviour here splits trajectories at any frame to frame displacements that are greater than two standard deviations away from the mean displacement for that ion.

3. Assign new segment IDs after each jump.
4. Return DataFrame with `segment_id`, `time`, and positions.

3) `membrane.py` Locating the Membrane

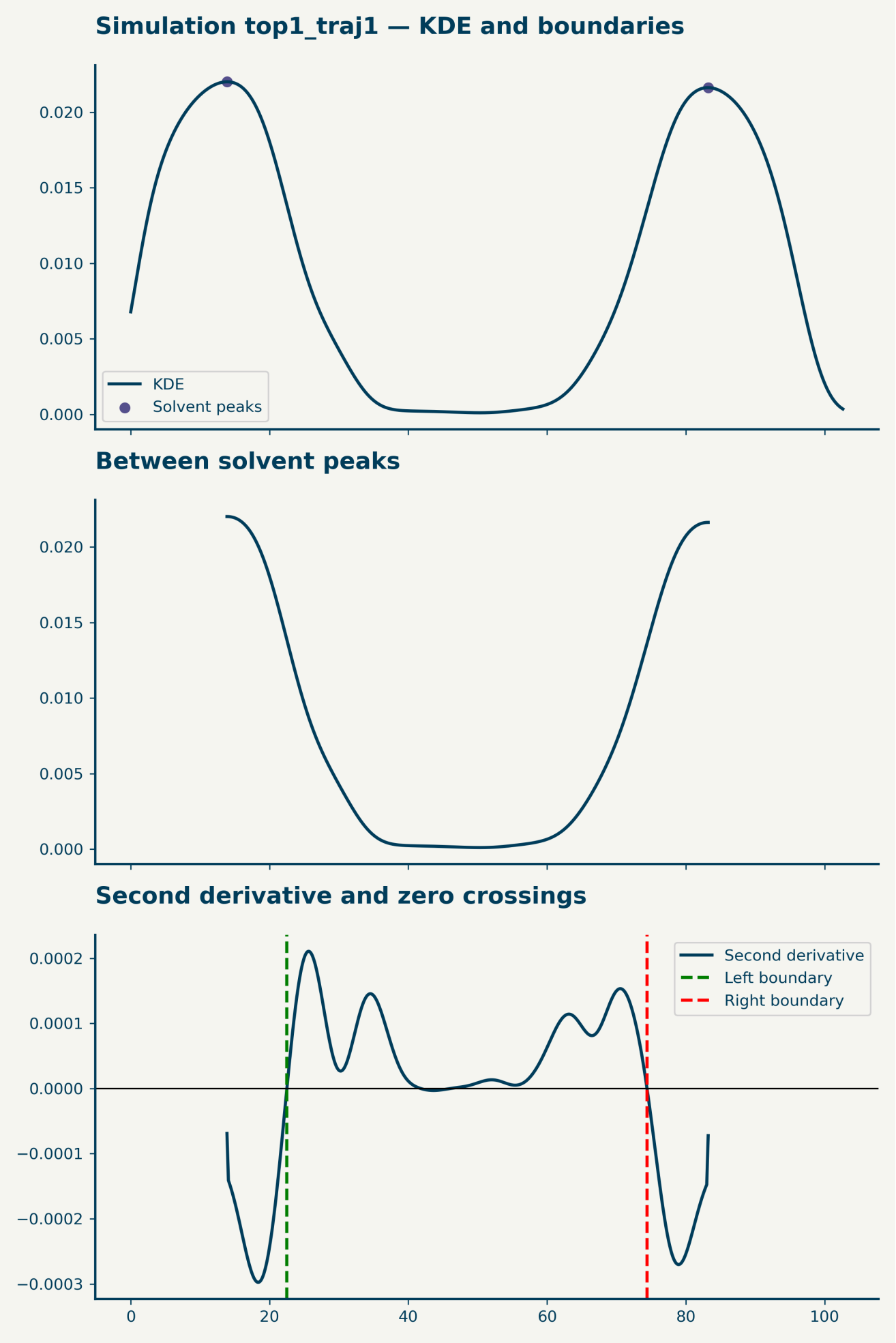
> ⚠ **Warning**
>
> The membrane is located by analysing the **ion density distribution** along a chosen axis. This algorithm is reliant on enough frames to create a representative density and two principle assumptions:
>
> > **1) Symmetric Bimodal Ion Distribution**
> > One of the highest density points will be in the intracellular space and the other in the extracellular space. Implicitly, there will be **two** broadly symmetric regions of high density.
>
> > **2) Central Linear Membrane**
> > The membrane region falls between these two peaks and contains a comparably lower ion density. The membrane is broadly linear and axis aligned

Steps:

1. **Smooth density using Gaussian KDE.**
   Using a Gaussian to smooth out the densities along the chosen axis should yield a plot with a large peak on either side (corresponding to bulk solvent) and a trough in the centre (corresponding to the membrane).

2. **Identify the two highest peaks and truncate**
   The two highest peaks in the ion density define the edges of the region of interest. This avoids including low ion density tails at the simulation boundaries, which can distort the shape and derivatives of the density profile.

   Note: this step assumes the bulk solvent peaks fully flank the channel region (Assumption 2)
3. **Compute curvature (2nd derivative).**
   The inflection points in the second derivative correspond to the transition between bulk solvent and channel.
4. **Locate zero-crossings proximal to solvent (near peaks)**
   Moving from the bulk solvent peaks (Step 2) towards the centre of the channel, identify the inflexion points in the curve. These correspond to the start and end of the channel (membrane region).



A Note On Edge Detection

To identify the boundaries of the membrane or ion channel, we chose a method that combines Gaussian kernel density estimation (KDE) with second-derivative zero-crossings.

Alternative approaches, such as detecting maxima of the first derivative, thresholding, or using Sobel/Canny-like filters, are either more sensitive to noise or require arbitrary thresholds.

---

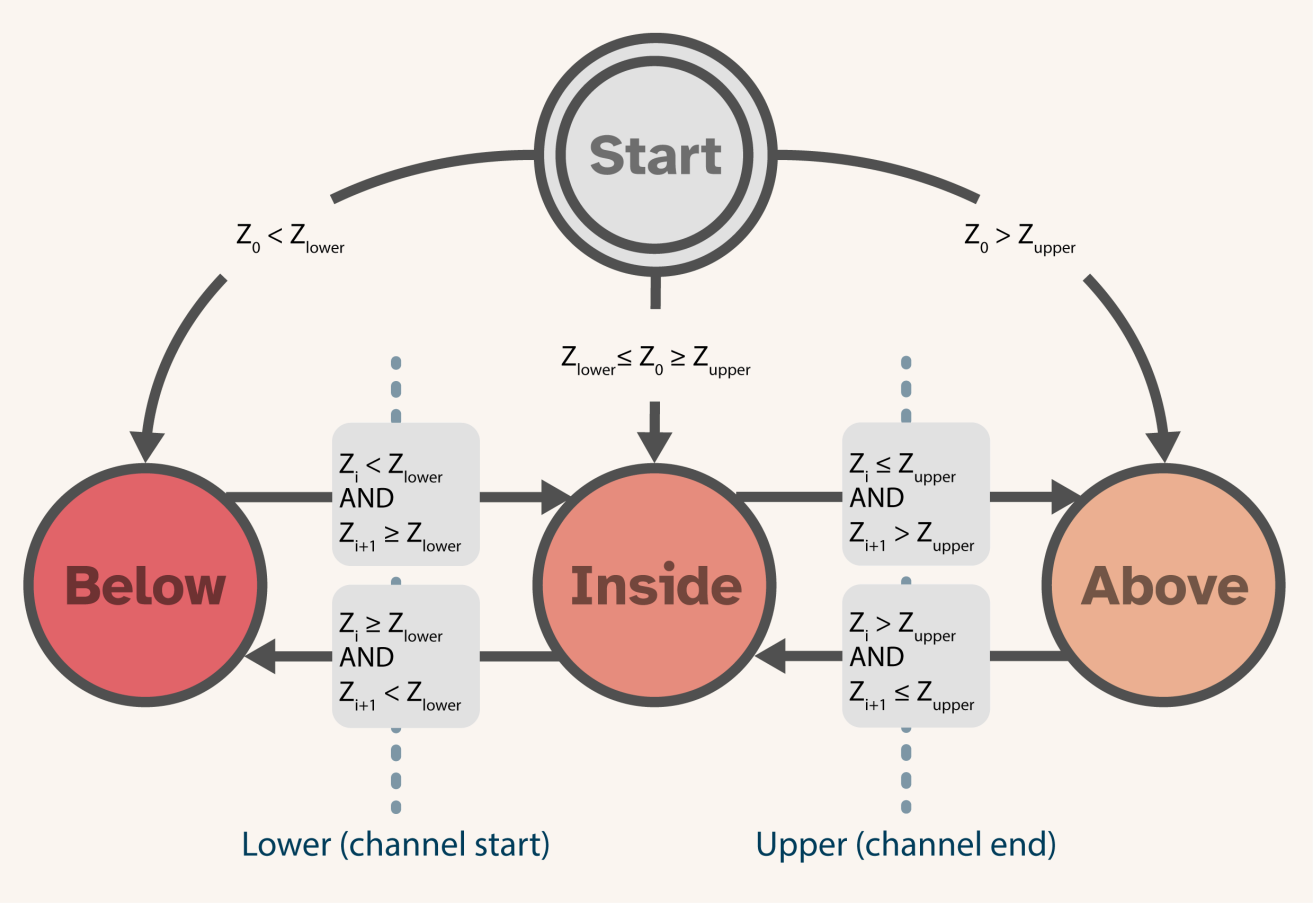
4) `crossings.py` Detecting Permeation Events

Summary

Uses segmented trajectories and membrane boundaries to detect full channel permeation events. This is the main logic to determine permeation events from the segments and membrane bounds determined previously.

Procedure:

1. **Define regions:**
   - Outside solvent = ±1
   - Channel interior = 0
2. **Get z-positions and times for each segment**
   Process each segment individually
3. **Count permeation events.**
   Each permeation is a complete traversal of the channel and involves both an entry and exit event.
   - Entry: outside → inside (−1→0 or +1→0)
   - Exit: inside → outside (0→±1)



4. Determine crossing direction (upward or downward)
   If an ion entered from below and exits above: **upward**.
   If entered from above and exits below: **downward**.

Example Usage (`example.py`)

Example Usage of the `ion_analysis` package.

```
from pathlib import Path
from ion_analysis.io import read_multiple_positions
from ion_analysis.segmentation import segment_trajectories
from ion_analysis.membrane import determine_membrane_bounds
from ion_analysis.crossings import classify_crossings

# List simulations
sims = [
    ("top1.tpr", "traj1.xtc"),
    ("top2.tpr", "traj2.xtc"),
    ("top3.tpr", "traj3.xtc"),
    ("top4.tpr", "traj4.xtc"),
    ("top5.tpr", "traj5.xtc"),
]
```

```
# Load positions for all simulations
print("Loading ion positions...")
positions_df = read_multiple_positions(sims, ion_resname="K")
print("Positions loaded:")
print(positions_df.head())

# Segment trajectories to remove PBC jumps
print("\nSegmenting trajectories...")
seg_df = segment_trajectories(positions_df)
print("Segments:")
print(seg_df.head())

# Filter out tiny segments for efficiency
filtered_df = seg_df.groupby(["simulation", "ion_id",
"segment_id"]).filter(lambda x: len(x) >= 5)
print("\nFiltered segments:")
print(filtered_df.head())

# Compute membrane boundaries for each simulation
print("\nDetermining membrane bounds...")
membrane_bounds = determine_membrane_bounds(filtered_df, axis='z')
print("Membrane bounds:")
print(membrane_bounds)

# Classify membrane crossing events
print("\nClassifying crossings...")
crossings_df = classify_crossings(filtered_df, membrane_bounds)
print(crossings_df.head(30))

# Summary: count crossings per simulation
crossings_per_sim = crossings_df.groupby('simulation').size()
print("\nCrossings per simulation:")
print(crossings_per_sim)
```

How to Use:

- **Replace the filenames** in `sims = [...]` with your own topology (`.tpr`) and trajectory (`.xtc`) files either as string filenames. Each pair should correspond to one simulation you want to analyse. It is most straightforward to put your files inside the ion_permeation_analyser folder with the example.py script.
- (Optional) **Change the ion name** in `ion_resname="K"` to match the residue name of the ion in your topology (e.g. `"NA"`, `"CL"`, `"CA"`, `"MG"`).

You can check this using `gmx dump -s top.tpr | grep resname` or in VMD.

- (Optional) **Adjust the membrane axis** in `axis='z'` if your membrane is oriented differently (`'x'` or `'y'` instead). The algorithm assumes the membrane is roughly planar and aligned with this axis.
- (Optional) The line filtering out tiny segments (`len(x) >= 5`) can be relaxed or tightened depending on your frame stride or noise level. These short segments usually arise when an ion briefly crosses a periodic boundary starting a new segment and then immediately wrapping back without actually entering or exiting the membrane.
  - Raise if performance is slow **and** your trajectories are very finely sampled (small time steps between frames).
  - Lower if it is plausible for an ion to cross your channel within 5 frames
- (Optional) You can modify printing sections for more or less verbose output.

Running on a HPC

Hint

If you cannot use `git clone` on the HPC:

1. Download the repository ZIP locally from GitHub.
2. Use `scp` or `rsync` to copy it to the HPC:

```
scp ion_permeation_analyser.zip user@hpc.server:/path/to/workdir
```

3. Unzip and run `example.py` inside a virtual environment.