Comprehensive Guide: Simulating Vapor-Liquid Equilibrium of a Binary Mixture using AmberTools & GROMACS

Mohammad Torikh

June 21, 2025

Contents

Abstract

This document provides a comprehensive, step-by-step guide for simulating the Vapor-Liquid Equilibrium (VLE) of a binary organic mixture (exemplified by Methanol-Propanol) using a combination of AmberTools for parameterization and system building, and GROMACS for Molecular Dynamics (MD) simulations. It details each stage from individual molecule preparation, force field parameterization using the General Amber Force Field (GAFF2), system assembly, equilibration protocols, to VLE production runs and initial data analysis. Common pitfalls, troubleshooting advice, and explanations of key commands and concepts are included to assist users, particularly those new to MD simulations or this specific workflow. The goal is to enable the user to generate reliable VLE data points suitable for constructing phase diagrams.

1 Introduction

Vapor-Liquid Equilibrium (VLE) data is crucial in chemical engineering for the design and optimization of separation processes like distillation. Experimental determination of VLE data can be time-consuming and expensive. Molecular Dynamics (MD) simulations offer a powerful computational alternative to predict VLE behavior from the underlying intermolecular forces. This guide focuses on a common workflow employing the widely used GAFF2 force field (suitable for organic molecules) via AmberTools, and the efficient GROMACS MD engine.

2 Workflow Overview

The overall process can be visualized as follows:

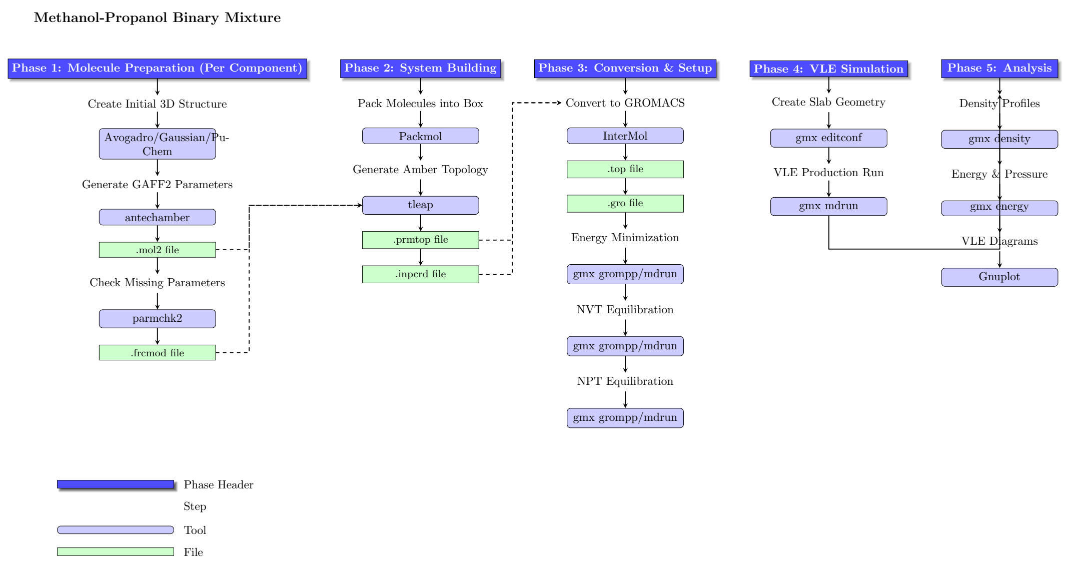

Figure 1: Overall workflow for VLE simulation.

The flowchart depicts:

1. Single Molecule Prep (PDB → antechamber → parmchk2 → .mol2 + .frcmod) - Loop for each component
2. System Building (Packmol → .pdb_packed → tleap → .prmtop + .inpcrd)
3. Conversion (InterMol → .top + .gro)
4. GROMACS Equilibration (Minimization → NVT → NPT)
5. VLE Setup (editconf to create slab)
6. VLE Production (NVT long run)
7. Analysis (gmx density, gmx energy, plotting)

3 Software Versions

This guide was developed and tested using the following software versions. Minor variations in newer or older versions might lead to slightly different command outputs or GUI options.

- AmberTools: 22 (antechamber, parmchk2, tleap)
- Packmol: 20.15.1
- InterMol: 0.1.2 (via Conda)
- GROMACS: 2024.5-conda_forge
- Gnuplot: 6.0 (or your preferred plotting tool)
- Operating System: Ubuntu Server (headless) on which simulations were run.

4 Phase 1: Single Molecule Preparation & Parameterization

This phase must be completed for *each unique chemical species* in your mixture. The goal is to obtain GAFF2-compatible parameters and a Tripos Mol2 file that describes the molecule's topology, atom types, and partial charges. For our example, this will be done for Methanol (`MOH`) and 1-Propanol (`PRH`).

4.1 1.1. Create Initial 3D Structure (PDB format)

The very first step in parameterizing a novel molecule for molecular dynamics simulations is to obtain a reasonable three-dimensional (3D) representation of its structure. This initial structure serves as the input for `antechamber`, which will analyze its topology (which atoms are bonded to which) and geometry to assign atom types and calculate partial charges. While `antechamber` can perform some geometry adjustments, starting with a chemically sensible structure can prevent errors and lead to more reliable parameters. This structure is typically provided in the Protein Data Bank (PDB) format (`.pdb`).

Methods for Obtaining Initial PDB Structures

There are several ways to generate a PDB file for your molecule:

- **Using Molecular Sketching/Building Software (Recommended):** This is the most common and flexible approach, especially for molecules not readily available in databases or for novel structures.
  - **Software Examples:**
    - *Avogadro:* A free, open-source, cross-platform molecular editor and visualizer. It's user-friendly and excellent for building organic molecules, adding hydrogens, and performing quick geometry clean-ups using built-in force fields like UFF or MMFF94.
    - *GaussView:* A commercial GUI for Gaussian, but also a capable molecule builder.
    - *ChemDraw/ChemDoodle:* Primarily 2D chemical drawing tools, but many versions can generate 3D coordinates and export to PDB.
    - *Jmol/BIOVIA Draw/Other Editors:* Various other free or commercial tools exist.
  - **General Process in a Builder (e.g., Avogadro):**
    1. *Build the heavy atoms:* Sketch the carbon backbone and place heteroatoms (O, N, S, etc.).
    2. *Set correct bond orders:* Ensure single, double, triple, and aromatic bonds are correctly assigned. Most builders attempt this automatically but review is good.
    3. *Add hydrogens:* Use the software's function to add hydrogens to satisfy valencies (e.g., Avogadro: Build → Add Hydrogens).
    4. *Perform a rough geometry optimization/cleanup:* This is crucial. Use a simple, fast molecular mechanics force field (Avogadro: Extensions → Optimize Geometry). This step ensures bond lengths, angles, and dihedrals are reasonable, preventing overly strained or clashed initial structures that could cause problems for `antechamber`. This is not meant to be a high-accuracy quantum mechanical optimization.

5. *Inspect the 3D structure:* Rotate and examine the molecule to ensure it looks chemically correct.
6. *Save As PDB:* Export the structure in PDB format (`.pdb`). Pay attention to options related to atom naming or residue naming if the software provides them, though we will often standardize these manually.

- **From Online Databases:** For common molecules, PDB files or similar 3D coordinate files might be available from chemical databases like:
  - PubChem (`https://pubchem.ncbi.nlm.nih.gov/`)
  - ChemSpider (`http://www.chemspider.com/`)
  - NIST Chemistry WebBook (`https://webbook.nist.gov/chemistry/`)

  If downloading, ensure you get a 3D conformer, and verify that the structure makes sense. You might still want to open it in a molecular editor to check/assign residue and atom names as per the conventions below.

- **Manual Creation (Advanced Users / Very Simple Molecules):** It is technically possible to create a PDB file using a plain text editor. However, the PDB format has very strict column requirements for each piece of information (atom serial, atom name, residue name, coordinates, etc.). This method is highly error-prone for beginners and generally not recommended unless for extremely simple cases or by experts familiar with the format.

PDB File Content and Formatting Requirements

A PDB file is a plain text file with specific formatting. For `antechamber` processing of a single molecule type, your PDB file should ideally contain only the atoms for one instance of that molecule.

Key PDB Record: ATOM/HETATM lines These lines contain the core information. Here's a breakdown of the essential columns (1-indexed), followed by an example:

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM " or "HETATM"	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name. (e.g., " C1 ", "C1M ")
17	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name. (e.g., "MOH")
22	Character	chainID	Chain identifier. (e.g., "A")
23 - 26	Integer	resSeq	Residue sequence number. (e.g., " 1")
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8.3)	x	Orthogonal coordinates for X.
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y.
47 - 54	Real(8.3)	z	Orthogonal coordinates for Z.
55 - 60	Real(6.2)	occupancy	Occupancy. (e.g., " 1.00")
61 - 66	Real(6.2)	tempFactor	Temperature factor. (e.g., " 0.00")
77 - 78	LString(2)	element	Element symbol, right-justified. (e.g., " C")
79 - 80	LString(2)	charge	Charge on the atom.

Conceptual example lines:

```
ATOM      1  C1M MOH A   1      -1.376   0.638  -0.000  1.00  0.00           C
HETATM    5  C2P PRH A   1      -0.109  -0.201   0.000  1.00  0.00           C
```

- **Cols 1-6: Record Name** (`ATOM` or `HETATM`). For general molecules, `HETATM` is often appropriate, but `ATOM` usually works too for `antechamber` input.
- **Cols 7-11: Atom Serial Number** (Integer, right-justified).
- **Cols 13-16: Atom Name** (String, typically 4 characters. Left-justified if ¡4 chars, e.g., `" C1 "`, `" O "`. Spaces are significant. Our strategy of using unique 3-character names like `C1M` fits well, becoming e.g., `"C1M "`).

- **Cols 18-20: Residue Name** (String, 3 characters, uppercase recommended, e.g., `MOH`).
- **Col 22: Chain Identifier** (Character, e.g., `A`). Can be left blank for single molecules.
- **Cols 23-26: Residue Sequence Number** (Integer, right-justified). Usually `1` for a single, isolated molecule.
- **Cols 31-38: X Coordinate** (Real, F8.3 format, e.g., `-1.376`).
- **Cols 39-46: Y Coordinate** (Real, F8.3 format, e.g., `0.638`).
- **Cols 47-54: Z Coordinate** (Real, F8.3 format, e.g., `-0.000`). *Strict adherence to these coordinate column formats is vital, as `antechamber` will fail if they are misaligned.*
- **Cols 55-60: Occupancy** (Real, F6.2, e.g., `1.00`). Typically `1.00`.
- **Cols 61-66: Temperature Factor** (Real, F6.2, e.g., `0.00`). Typically `0.00` for input structures.
- **Cols 77-78: Element Symbol** (String, right-justified, e.g., `" C"`, `" H"`).

Best Practice: Residue Naming

Consistency is Key: The residue name (e.g., `MOH` for methanol, `PRH` for propanol) assigned in the PDB file should be:

- A 3-character uppercase string (convention).
- Unique for each distinct molecule type in your system.
- The same name you will provide to `antechamber` via the `-rn` flag.
- The same name that will appear in the `.mol2` file generated by `antechamber`.
- The same name used in `tleap` when defining the residue (e.g., `MOH = loadmol2 moh.mol2`).
- The same name used in the `mixture_packed.pdb` file by `packmol`.

Pitfall: Using common biochemical residue names (e.g., `ALA`, `LYS`) for non-biological molecules can sometimes cause conflicts if standard biomolecular force fields are also loaded in `tleap`. As we saw with ‘POL‘ causing issues, choosing slightly more unique names like ‘PRH‘ (Propanol Hydrocarbon-like) can prevent ‘tleap‘ from misinterpreting or ”splitting” residues.

Best Practice: Atom Naming

Uniqueness within Residue: Atom names within a single residue definition must be unique (e.g., a methanol molecule cannot have two atoms named `C1`). Standard PDB atom names are up to 4 characters. **Strategy for this Workflow (Global Uniqueness):**

- For this specific workflow, especially due to the Packmol PDB workaround (see Section 5.1), we recommend using **globally unique atom names** across different molecule types.
- Example:
  - Methanol (`MOH`): `C1M, H1M, H2M, H3M, O1M, HOM`
  - Propanol (`PRH`): `C1P, H1P, H2P, C2P, H3P, H4P, C3P, H5P, H6P, H7P, O1P, HOP`
- This global uniqueness (e.g., `C1M` vs `C1P`) helps `tleap` definitively assign atoms to the correct residue type when reading the `mixture_packed.pdb` file, even if residue information were somehow ambiguous (though residue names should prevent this). It's crucial for the Packmol workaround.

Formatting: PDB atom names are typically 4 characters. If your chosen names are shorter (e.g., 3 characters like `C1M`), they should be left-justified in the 4-character field, followed by a space (e.g., `"C1M "`). Many tools handle this automatically if you specify a 3-character name. **Pitfall:** Inconsistent atom naming between the `.pdb` used for `antechamber`, the resulting `.mol2`, and the `.pdb` used for `packmol` will cause `tleap` to fail with "atom not found" errors.

Example: Conceptual PDB Structure for Methanol (`methanol_unique.pdb`)

Below is a conceptual representation of what the `methanol_unique.pdb` file might look like, emphasizing the unique atom and residue naming used in this workflow. Coordinates are illustrative.

```
ATOM      1  C1M MOH A   1       0.000   0.000   0.000  1.00  0.00           C
ATOM      2  H1M MOH A   1       0.000   0.000   1.090  1.00  0.00           H
ATOM      3  H2M MOH A   1       1.028   0.000  -0.363  1.00  0.00           H
ATOM      4  H3M MOH A   1      -0.514  -0.890  -0.363  1.00  0.00           H
ATOM      5  O1M MOH A   1      -1.300   0.500   0.000  1.00  0.00           O
ATOM      6  HOM MOH A   1      -1.900   0.000   0.000  1.00  0.00           H
TER
END
```

Listing 1: Example PDB content for Methanol (MOH)

This detailed PDB provides `antechamber` with a clear starting point, minimizing ambiguities in atom type assignment and connectivity.

4.2 1.2. Generate GAFF2 Atom Types & Charges (`antechamber`)

Once you have a valid PDB file for your individual molecule (e.g., `methanol_unique.pdb` with residue name `MOH`), the next step is to use the `antechamber` program from the AmberTools suite. The primary purposes of running `antechamber` at this stage are:

- **Assign GAFF2 Atom Types:** `antechamber` analyzes the chemical environment of each atom (its element type, connectivity, and bond orders) and assigns it a specific GAFF2 (General Amber Force Field version 2) atom type. These atom types (e.g., `c3` for an sp3 carbon, `oh` for an alcohol oxygen, `hc` for a hydrogen bonded to an aliphatic carbon) are crucial because all subsequent force field parameters (bond lengths, angles, dihedrals, Lennard-Jones parameters) are defined based on these types.
- **Calculate Partial Atomic Charges:** For most simulations, especially those involving polar molecules or condensed phases, accurate representation of electrostatic interactions is vital. `antechamber` can employ various quantum mechanical (QM) methods followed by charge fitting procedures to derive a set of partial atomic charges that reproduce the molecule's electrostatic potential. A widely used and generally robust method for GAFF is AM1-BCC.

- **Generate a Tripos Mol2 File:** The output of this step is typically a `.mol2` file. This is a standard chemical file format that, in this context, will store the molecule's 3D coordinates, bond connectivity, the newly assigned GAFF2 atom types, the calculated partial charges, and the residue name you specified. This `.mol2` file becomes a key input for subsequent steps (`parmchk2` and `tleap`).

Command Structure

The general command for running `antechamber` is:

```
antechamber -i <input_file> -fi <input_format> -o <output_file> -fo <
    output_format> -c <charge_method> -s <verbosity> -rn <residue_name> -nc <
    net_charge>
```

Listing 2: General antechamber command structure

For our Methanol (`MOH`) example:

```
antechamber -i methanol_unique.pdb -fi pdb -o moh.mol2 -fo mol2 -c bcc -s 2 -rn
     MOH -nc 0
```

Listing 3: Antechamber command for Methanol (MOH)

Explanation of Command Options

`-i methanol_unique.pdb` Specifies the input file. In our case, this is the PDB file we prepared in step 1.1.

`-fi pdb` Specifies the format of the input file. Here, it's `pdb`.

`-o moh.mol2` Specifies the desired output file name. It's conventional to use the molecule's identifier and the `.mol2` extension.

`-fo mol2` Specifies the format of the output file, which is Tripos `mol2`.

`-c bcc` Specifies the charge calculation method. `bcc` stands for AM1-BCC (Bond Charge Correction). This is a two-stage process: first, a semi-empirical quantum calculation (AM1) is performed to get an initial set of charges. Then, "Bond Charge Corrections" (BCCs) are applied based on bond types to refine these charges. AM1-BCC charges are generally recommended for use with the GAFF/GAFF2 force field as they tend to provide a good balance of accuracy and computational speed for a wide range of organic molecules. Other common options include `gas` (for Gasteiger charges, which are empirical and very fast but less accurate for condensed phases) or `resp` (Restrained Electrostatic Potential fitting, which can be more accurate but requires a QM electrostatic potential, often from a higher-level QM calculation like HF/6-31G*). For general organic molecules with GAFF2, `bcc` is a robust default.

`-s 2` Sets the status information level (verbosity). `2` provides a good amount of information about the steps `antechamber` is taking without being overwhelming.

`-rn MOH` Assigns the residue name in the output `.mol2` file. This is a critical flag. The residue name provided here (e.g., `MOH`) will be embedded in the `moh.mol2` file. Later, when `tleap` loads this `moh.mol2` file, it will define a molecular unit (residue template) with this name. It is essential for consistency, especially if your input PDB also used this residue name. We used `MOH` in our `methanol_unique.pdb` and ensure it here too.

`-nc 0` Specifies the net charge of the molecule. Methanol is a neutral molecule, so its net charge is `0`. For ions, you would set this to the appropriate integer charge (e.g., `1` for a sodium ion, `-1` for a chloride ion). `antechamber` will try to derive partial charges that sum up to this net charge.

What Happens During an `antechamber` Run?

1. **Reading Input:** Parses the input PDB file for atomic coordinates and (if present and unambiguous) connectivity.

2. **Atom and Bond Typing (Initial):** Determines initial atom and bond types based on geometry and element.

3. **SQM Calculation (Semi-empirical QM):** If AM1-BCC is chosen, `antechamber` calls an external semi-empirical QM program (usually `sqm`, which is part of AmberTools) to perform an AM1 calculation on the molecule. This calculation optimizes the geometry (if requested, though usually not heavily for this step) and calculates Mulliken charges and bond orders. You'll see files like `sqm.in`, `sqm.out`, `sqm.pdb` being created and deleted in your working directory during this step.

4. **BCC Application:** The Mulliken charges from the AM1 calculation are then adjusted using the Bond Charge Correction parameters to yield the final AM1-BCC charges.

5. **GAFF2 Atom Type Assignment:** Based on the (potentially QM-refined) geometry, connectivity, and element types, final GAFF2 atom types are assigned.

6. **Writing Output:** The `moh.mol2` file is written, containing:

   - The molecule name (usually taken from the residue name or input file).
   - The number of atoms, bonds, etc.
   - An `@<TRIPOS>ATOM` section listing each atom with its coordinates, GAFF2 atom type (e.g., `c3`, `oh`, `hc`), its residue name (e.g., `MOH`), and its calculated partial charge.
   - An `@<TRIPOS>BOND` section listing the bonds.

Checking the Output (`moh.mol2`)

After `antechamber` finishes (hopefully without errors), it's good practice to open the generated `.mol2` file (e.g., `moh.mol2`) with a text editor and quickly inspect it:

- Verify the molecule name at the top is what you expect (e.g., `MOH`).

- In the `@<TRIPOS>ATOM` section:

  – Check that the assigned GAFF2 atom types (usually the 5th column after coordinates) look reasonable for your molecule. For example, an sp3 carbon should be `c3`, an alcohol oxygen `oh`, an alcohol hydrogen `ho`, hydrogens on sp3 carbons `hc` or `h1`, `h2`, etc. depending on their environment.

  – Confirm the residue name (usually the 7th or 8th column, depends on exact Mol2 variant) is correct (e.g., `MOH`).

  – Look at the partial charges (last column). Do they make sense chemically? (e.g., oxygen should be negative, hydroxyl hydrogen positive, carbons can be slightly positive or negative depending on neighbors). The sum of these charges should equal the `-nc` value you specified.

Troubleshooting `antechamber`

- **PDB Formatting Errors:** As seen earlier, `antechamber` is strict about PDB coordinate columns. If it complains about this, re-check your input PDB formatting.

- **Unusual Valencies / Weird Geometry:** If your input PDB has chemically incorrect structures (e.g., a carbon with 5 bonds, or extremely short/long bonds), `antechamber` might fail or assign strange atom types. A quick geometry cleanup in a molecular editor often helps.

- **`sqm` Failures:** Sometimes the `sqm` calculation can fail for very strained or unusual molecules. The `sqm.out` file (if it remains) might contain clues. This is rarer for simple organic molecules.

- **Net Charge Mismatch:** If you specify an `-nc` value that is impossible for the given atoms (e.g., `-nc 1` for methane), it will likely error out or produce strange charges.

By successfully running `antechamber`, you now have a GAFF2-typed and charged representation of your molecule in a `.mol2` file, ready for the next step with `parmchk2`.

4.3 1.3. Check for Missing Force Field Parameters (`parmchk2`)

After `antechamber` has successfully assigned GAFF2 atom types to your molecule and generated the `.mol2` file (e.g., `moh.mol2`), the next step is to ensure that all necessary force field parameters (for bonds, angles, and dihedrals involving these atom types) are actually present in the standard GAFF2 library. While GAFF2 is quite comprehensive for common organic functional groups, it's possible that a specific combination of atom types in your molecule, particularly around unusual linkages or novel structures, might not have pre-defined parameters. The `parmchk2` program from AmberTools is designed to address this.

Purpose of `parmchk2`

- **Identify Missing Parameters:** `parmchk2` reads your `.mol2` file (which contains the GAFF2 atom types assigned by `antechamber`). It then compares the bonds, angles, and dihedral terms implied by your molecule's topology against the parameters available in the specified GAFF2 library.
- **Generate Missing Parameters by Analogy (if possible):** If parameters are missing, `parmchk2` attempts to estimate them by analogy to similar, known parameters within the GAFF2 force field. For example, if a specific C-C-O-H dihedral is missing but other similar C-C-O-X dihedrals exist, it might derive a plausible parameter.
- **Highlight Critically Missing Parameters:** If `parmchk2` cannot find a suitable analogy or if a parameter is deemed critical and absolutely missing, it will flag this, often with a comment like "`ATTN: NEEDS TOPOLOGY MODIFICATION`".
- **Create a `.frcmod` File:** The primary output of `parmchk2` is a Force Field Modification file (`.frcmod`). This file contains entries for any parameters that `parmchk2` found to be missing from the main GAFF2 library and for which it could generate or suggest values. `tleap` will later read this `.frcmod` file to supplement the main GAFF2 parameters.

Command Structure

The general command for running `parmchk2` is:

```
parmchk2 -i <input_mol2_file> -f <input_format> -o <output_frcmod_file> -s <
    force_field_type> [other_options]
```

Listing 4: General parmchk2 command structure

For our Methanol (`MOH`) example:

```
parmchk2 -i moh.mol2 -f mol2 -o moh.frcmod -s gaff2
```

Listing 5: Parmchk2 command for Methanol (MOH)

Explanation of Command Options

`-i moh.mol2` Specifies the input file, which is the `.mol2` file generated by `antechamber`. This file contains the crucial GAFF2 atom types.

`-f mol2` Specifies the format of the input file as `mol2`.

`-o moh.frcmod` Specifies the desired output file name for the force field modifications. It's conventional to use the molecule's identifier with the `.frcmod` extension.

`-s gaff2` Specifies the main force field type to check against. `gaff2` refers to the General Amber Force Field version 2. If you were using the original GAFF, you would use `-s gaff`.

`-a Y` **or** `-a N` **(Optional)** Print all parameters found by `parmchk2` to standard output (`Y`) or not (`N`, default). Useful for debugging or detailed inspection but not usually needed for the `.frcmod` file itself.

`-p $AMBERHOME/dat/leap/parm/gaff2.dat` **(Optional)** Explicitly provide the path to the main GAFF2 parameter file if `parmchk2` has trouble finding it (usually not necessary if AmberTools is sourced correctly).

What to Expect in the Output (`moh.frcmod`)

- **For common, well-parameterized molecules** (like methanol with GAFF2): The `.frcmod` file might be very short or even practically empty except for comments and section headers (`MASS, BOND, ANGLE, DIHEDRAL, IMPROPER, NONBON`). This indicates that GAFF2 already contains all the necessary parameters for the atom types found in your molecule.

- **For more complex or unusual molecules:** The `.frcmod` file will list any parameters that `parmchk2` generated. Each entry will specify the atom types involved and the derived parameter values (e.g., force constant, equilibrium bond length/angle, periodicity and phase for dihedrals). Example of a dihedral entry: `c3-c3-c3-hc 1 0.156 0.000 3.000 GAFF2, C-C-C-H, from ccnmrht (ParmBrown,1998)` It's good practice to review these generated parameters. If many "`ATTN:...`" messages appear, it might indicate a problem with the initial atom typing or a truly novel chemical moiety that requires more careful parameter development (beyond the scope of this automated workflow).

What `parmchk2` Does Not Do

- It does not re-calculate charges or change atom types. These come from `antechamber`.

- It does not guarantee the accuracy of parameters generated by analogy, especially for highly unusual systems. These are best estimates.

After Running `parmchk2`

No direct user action is usually required with the `.frcmod` file itself, other than noting its existence. `tleap` will automatically load and use this file in a later step to ensure all parts of your molecule are described by a complete set of force field parameters. If `parmchk2` prints many "`ATTN`" messages to the screen or in the `frcmod`, it's a sign to be cautious and perhaps re-evaluate your molecule's structure or the atom types assigned by `antechamber`. For simple molecules like methanol and propanol, `parmchk2` typically runs quickly and without issues when using GAFF2.

5 Phase 2: System Building (Packing and Amber Topology)

In this phase, we will use the `.mol2` and `.frcmod` files generated for each individual component (Methanol and Propanol in our case) to construct a simulation box containing a specified number of each molecule. We will then use `tleap` to assign the complete set of GAFF2 force field parameters to this entire system.

5.1 2.1. Pack Molecules into a Simulation Box (`packmol`)

Purpose of `packmol`

The preceding steps (Phase 4) focused on preparing individual molecule "blueprints" – defining their structure, assigning GAFF2 atom types, calculating charges (`.mol2` files), and identifying any missing force field parameters (`.frcmod` files). However, to simulate a bulk system, especially a mixture, we need an initial three-dimensional arrangement of many copies of these molecules within a defined simulation volume (the "box").

This is where `packmol` comes in. Packmol (available at `http://m3g.iqm.unicamp.br/packmol`) is a powerful and widely used software package specifically designed to create initial configurations for molecular dynamics simulations. Its core function is to "pack" molecules of specified types into defined regions (like cubes, spheres, or more complex geometries) while attempting to satisfy spatial constraints, avoid severe atomic overlaps (clashes), and achieve a target number density or number of molecules.

Starting from a well-packed, reasonably dense initial configuration is crucial because:

- **Reduces Equilibration Time:** A good starting point is closer to an equilibrium liquid state than, for example, a very dilute gas that would take a very long time to condense.

- **Prevents Extreme Forces:** If molecules are initially placed too close together, the resulting repulsive forces at the start of an energy minimization or MD run can be astronomically large, potentially causing the simulation to become unstable or "blow up." Packmol's tolerance setting helps mitigate this.

Our Specific Strategy with `packmol`

In an ideal scenario, one might prefer to provide `packmol` with the `.mol2` files generated by `antechamber`. These `.mol2` files are rich in information, containing not only coordinates but also the GAFF2 atom types, partial charges, and bond connectivity for each defined residue (`MOH` and `PRH`). This would ensure that `packmol` is working with the most complete representation of our parameterized molecules.

However, during the development of this workflow, we encountered an issue where the specific version of `packmol` (20.15.1) available on our simulation server did not correctly process the `filetype mol2` directive in the input script. It defaulted to interpreting all input structures as PDB files, leading to errors when it tried to parse a `.mol2` file using PDB formatting rules.

To circumvent this specific `packmol` version limitation, we adopted the following robust workaround:

1. **Leverage Uniquely Named PDBs:** In Phase 4.1, we emphasized creating input PDB files (e.g., `methanol_unique.pdb`, `propanol_unique.pdb`) where each atom has a unique name that also implicitly identifies its molecule type (e.g., `C1M`, `O1M` for methanol; `C1P`, `O1P` for propanol). The residue names (`MOH`, `PRH`) were also consistently applied.

2. **Inform `packmol` to Expect PDBs:** In the `packmol.inp` script, we explicitly set `filetype pdb`.

3. **`packmol`'s Action:** `packmol` then reads the coordinates, atom names, and residue names directly from these `_unique.pdb` files. It arranges the specified number of each molecule type within the defined box.

4. **Output Consistency:** The resulting output file, `mixture_packed.pdb`, will therefore contain all the molecules with these same unique atom names (e.g., `C1M`, `C1P`) and residue names (`MOH`, `PRH`).

5. **`tleap` Compatibility:** This strategy is crucial because in the next phase (Section 5.2), `tleap` will first load our original `.mol2` files (e.g., `moh.mol2`, `pol.mol2`). These `.mol2` files were generated by `antechamber` from the `_unique.pdb` files and thus also contain these exact same unique atom names and the correct residue definitions (`MOH`, `PRH`). When `tleap` then loads the `mixture_packed.pdb`, it can perfectly match the atoms and residues in the packed PDB with the molecular templates it has just learned from the `.mol2` files. This ensures correct assignment of atom types, charges, and force field parameters.

This PDB-based workaround, while a response to a specific tool behavior, highlights the general importance of maintaining consistent naming across all stages of a simulation setup pipeline. If a different, fully MOL2-compatible version of `packmol` were used, the `structure` lines in `packmol.inp` could directly point to the `.mol2` files from `antechamber`.

Creating the `packmol.inp` File

`packmol` is controlled by a plain text input file, conventionally named with a `.inp` extension. Below is the `packmol.inp` file tailored for our methanol-propanol mixture, incorporating the PDB input strategy.

```
# Packmol input file for Methanol (MOH) and Propanol (PRH) mixture
# Version: 1.0
# Author: Mohammad Torikh
# Date: June 21, 2025
# Purpose: Generate initial packed configuration for VLE simulation.

# Global settings
# ----------------
# tolerance: Minimum distance (in Angstroms) between atoms of different
    molecules.
# Packmol uses an optimization algorithm to try and satisfy this.
# A value of 2.0 Angstroms is a common starting point to prevent severe initial
     clashes.
# Smaller values might allow for denser packing but increase clash risk.
```

```
tolerance 2.0

# filetype: Specifies the format of the coordinate files provided in the ’structure’ blocks.
# Due to observed issues with ’mol2’ handling in Packmol v20.15.1 on our system,
# we use ’pdb’. This necessitates that the input PDB files contain
# all necessary atom naming information consistent with what tleap will expect.
filetype pdb

# output: The name of the PDB file Packmol will generate, containing all molecules.
output mixture_packed.pdb

# Molecule definitions and packing instructions
# ------------------------------------------

# Methanol (MOH)
# ’structure’ keyword indicates the start of a definition for a molecule type.
# ’methanol_unique.pdb’ is the template PDB file for a single methanol molecule.
# This file must contain the residue name ’MOH’ and unique atom names (e.g., C1M, O1M).
structure methanol_unique.pdb
  # ’number’ specifies how many copies of this molecule to pack.
  number 200
  # ’inside cube L L L X Y Z’ defines a cubic region for packing.
  # Here, ’0. 0. 0.’ are the coordinates of one corner of the cube.
  # ’40.’ is the side length of the cube in Angstroms.
  # The cube will span from (0,0,0) to (40,40,40) Angstroms.
  # The total volume is 40^3 A^3 = 64 nm^3.
  # The number of molecules and box size should be chosen to achieve a
  # density roughly appropriate for a liquid mixture. This can be estimated:
  # Mass_MOH = 200 * 32.04 g/mol; Mass_PRH = 100 * 60.10 g/mol
  # Total_mass_amu = (200*32.04 + 100*60.10) amu
  # Volume_A^3 = 40^3 A^3
  # Density ~ Total_mass_amu / Volume_A^3 * (1.66054e-24 g/amu) / (1e-24 cm^3/A^3) g/cm^3
  # For this example, it yields ~0.65 g/cm^3, a reasonable starting point for organic liquids.
  # This box will later be equilibrated under NPT conditions.
  inside cube 0. 0. 0. 40.
end structure

# Propanol (PRH)
# ’propanol_unique.pdb’ is the template PDB for a single 1-propanol molecule.
# This file must contain the residue name ’PRH’ and unique atom names (e.g., C1P, O1P).
structure propanol_unique.pdb
  number 100
  # Molecules of propanol will be packed into the same cubic region defined above.
  # Packmol will attempt to place all 300 molecules (200 MOH + 100 PRH)
  # within this 40x40x40 Angstrom cube.
  inside cube 0. 0. 0. 40.
end structure

# Note: Packmol offers many other options for more complex packing, such as
# different geometric constraints (spheres, boxes, around existing structures),
# fixed molecules, etc. Consult the Packmol user guide for advanced usage.
# For our VLE purpose, a simple cubic packing is sufficient as a starting point.
```

Listing 6: Packmol input file (`packmol.inp`) for Methanol-Propanol mixture

Running `packmol`

To execute `packmol`, you redirect the input file using standard input redirection (`<`):

```
packmol < packmol.inp
```

Listing 7: Running Packmol

This command should be run in the directory containing `packmol.inp`, `methanol_unique.pdb`, and `propanol_unique.pdb`.

Interpreting `packmol` Output

During its run, `packmol` provides valuable feedback on the terminal:

- It confirms the input file being read.
- It lists the types of coordinate files specified (e.g., `pdb`).
- It reports the output file name.
- It details the number of independent structures (molecule types) and their atom counts.
- It will show progress as it attempts to pack each molecule type and then all molecules together, often showing optimization loop information and function values related to satisfying constraints.
- A successful run typically ends with a "Success!" message and the final objective function value (ideally close to zero, indicating constraints were well met).
- It also prints a citation request for the Packmol paper.

Output Files from `packmol`

- `mixture_packed.pdb` **(Primary Output):** This is the PDB file containing the coordinates of all packed molecules.
  - It will contain `ATOM` or `HETATM` records for all atoms (in our case, 200 methanols $\times \sim 6$ atoms/MOH + 100 propanols $\times \sim 12$ atoms/PRH = ~ 2400 atoms).
  - Each atom record will retain the unique atom names (e.g., `C1M, O1M, C1P, O1P`) and residue names (`MOH, PRH`) from the input template PDBs.
  - Residue sequence numbers will be assigned by `packmol` to distinguish individual molecules (e.g., the first methanol might be residue 1, the second residue 2, etc., up to 200, then propanols might start from 201).
  - The last line of this PDB file might contain box vector information, often as a `CRYST1` record, or sometimes GROMACS-style box vectors if the Packmol version supports it well for PDB output. `tleap` will handle defining the box explicitly in the next step.
- `packmol.log` **(Optional):** A log file may be generated containing more verbose details of the packing process, which can be useful for debugging if `packmol` fails or struggles to pack the system.

Quick Verification of `mixture_packed.pdb`

After `packmol` completes, a quick sanity check of the output PDB is recommended:

- **File Size:** Ensure `mixture_packed.pdb` has a non-zero and substantial file size.
- **Atom and Residue Names:**

```bash
grep "ATOM " mixture_packed.pdb | grep " MOH " | head -n 5 # Check first
    few methanol atoms
grep "ATOM " mixture_packed.pdb | grep " PRH " | head -n 5 # Check first
    few propanol atoms
```
Listing 8: Checking atom and residue names in packed PDB

Confirm the residue names are `MOH` and `PRH` respectively, and that the atom names match your unique naming scheme (e.g., `C1M, O1P`).

- **Visual Inspection (Optional but Recommended):** If you have access to a molecular visualizer (like VMD or PyMOL) that can read PDB files, loading `mixture_packed.pdb` can give you a visual confirmation that the molecules are packed within a box and don't have obvious, gross overlaps.

With a successful `mixture_packed.pdb`, you have a well-defined starting configuration for your entire mixture, ready for `tleap` to build the complete molecular topology for the Amber force field.

5.2 2.2. Generate Amber Topology & Coordinate Files (`tleap`)

After successfully preparing individual molecule parameter files (`.mol2` and `.frcmod` for each component) and creating an initial packed configuration of the mixture (`mixture_packed.pdb`), the next crucial step is to use `tleap`. `tleap` (and its graphical counterpart `xleap`) is a core program in the AmberTools suite responsible for building the complete molecular topology and assigning parameters for a given system according to the Amber force fields.

Purpose of `tleap`

For our workflow, `tleap` will perform several key functions:

- **Load Force Field:** It starts by loading a base force field, in our case, GAFF2 (`leaprc.gaff2`), which contains the standard parameters for many common atom types, bonds, angles, and dihedrals.
- **Load Custom Parameters:** It then loads the `.frcmod` files we generated with `parmchk2`. These files provide any missing parameters specific to our methanol (`MOH`) and propanol (`PRH`) molecules that weren't already in the base GAFF2.
- **Define Molecular Units (Residue Templates):** `tleap` reads our `.mol2` files (e.g., `moh.mol2`, `pol.mol2`[1]). From these, it learns the definition of each of our custom molecular units (`MOH` and `PRH`), including their atom names, GAFF2 atom types, partial charges, and internal connectivity (bonds). These definitions become reusable "templates."
- **Assemble the System:** It reads the `mixture_packed.pdb` file generated by `packmol`. For each atom in this PDB, `tleap` attempts to match its atom name and residue name (e.g., atom `C1M` in residue `MOH`) to the atoms within the corresponding loaded molecular unit templates (`MOH` or `PRH`).
- **Parameter Assignment:** Once atoms are matched, `tleap` uses the GAFF2 atom types (from the `.mol2` templates) to look up and assign all necessary force field parameters (bond stretching, angle bending, dihedral torsion, non-bonded Lennard-Jones, and electrostatic interactions using the charges from the `.mol2` files) for every interaction in the entire system.
- **Define Box Information:** We instruct `tleap` about the simulation box dimensions.
- **Output Amber System Files:** Finally, `tleap` saves two critical files:
  - A parameter/topology file (`.prmtop`): This text file contains a complete description of the molecular system's topology (all atoms, residues, bonds, angles, dihedrals, exclusions) and all associated force field parameters (force constants, equilibrium values, charges, Lennard-Jones parameters). It's the "recipe book" for the simulation.

[1]Note: `pol.mol2` was the filename used for propanol's `.mol2` file, which internally defined the residue as `PRH` after troubleshooting the 'POL' naming issue.

– An coordinate file (`.inpcrd`): This text file contains the initial atomic coordinates for all atoms in the system, usually matching those from the input PDB (`mixture_packed.pdb`) but now in Amber's specific format and order.

The `tleap_mixture.in` Script

`tleap` is controlled by a script file (conventionally with a `.in` extension). Here's the script we developed, with detailed explanations:

```
# tleap input script for creating Amber system files for Methanol-Propanol mixture
# Version: 1.0
# Author: Mohammad Torikh
# Date: June 21, 2025

# --- Load Force Field and Custom Parameters ---

# 'source' command loads another leap script or a parameter set.
# 'leaprc.gaff2' loads the General Amber Force Field version 2.
# This file defines standard atom types, bond, angle, dihedral, and LJ parameters.
# It must be sourced before loading molecule definitions that use GAFF2 atom types.
# AmberTools needs to be installed and its environment sourced for tleap to find this.
source leaprc.gaff2

# 'loadamberparams' reads a .frcmod file.
# These files contain parameters identified as missing or needing modification by parmchk2.
# We load one for each unique molecule type we parameterized.
loadamberparams moh.frcmod  # Parameters for Methanol (MOH)
loadamberparams pol.frcmod  # Parameters for Propanol (PRH).
                            # Note: This pol.frcmod was generated from pol.mol2,
                            # which itself was created by antechamber using "-rn PRH".

# --- Define Molecular Units (Residue Templates) ---

# 'loadmol2' reads a Tripos Mol2 file and uses it to define a new molecular unit (residue template).
# The variable on the left (e.g., MOH) becomes the name of this unit within tleap.
# The .mol2 file (e.g., moh.mol2) must contain:
#   1. Atom names (e.g., C1M, O1M, H1M...)
#   2. Correct GAFF2 atom types for each atom (assigned by antechamber)
#   3. Partial atomic charges for each atom (calculated by antechamber)
#   4. Bond connectivity.
#   5. The residue name embedded within the .mol2 file (from antechamber's -rn flag)
#      should ideally match the variable name here for clarity (e.g., MOH for moh.mol2).
#      When tleap loads `pol.mol2` (which was created with `antechamber ... -rn PRH`),
#      the unit defined *inside* that .mol2 file is named PRH. We assign it to a tleap
#      variable also named PRH for consistency.
MOH = loadmol2 moh.mol2
PRH = loadmol2 pol.mol2  # This pol.mol2 defines the PRH unit

# --- Load Packed Coordinates and Assemble the System ---

# 'loadpdb' reads a PDB file and creates a new system unit.
# It uses the residue and atom names from the PDB file to identify which
# pre-defined molecular units (MOH, PRH that we just loaded) to use for each
# residue encountered in the PDB.
# CRITICAL:
#   - The residue names in 'mixture_packed.pdb' (e.g., "MOH", "PRH") MUST match
#     the names of the units we defined above via loadmol2.
#   - The atom names within each residue in 'mixture_packed.pdb' (e.g., "C1M", "O1M"
#     for an MOH residue) MUST match the atom names within the corresponding
#     loaded .mol2 template (e.g., moh.mol2).
# This matching is why our unique atom naming strategy with the Packmol PDB workaround was essential.
mixture = loadpdb mixture_packed.pdb

# --- System Checks and Box Definition ---

# 'check' is a very important command. It inspects the specified unit (here, 'mixture')
# for missing parameters (bonds, angles, dihedrals) or other inconsistencies.
# Any errors or serious warnings here usually indicate a problem with atom typing,
# missing .frcmod parameters, or incorrect connectivity.
# Minor warnings about non-integral net charge for the whole system are common
# for GAFF/AM1-BCC due to floating point sums, and usually acceptable if small.
check mixture

# 'set <unit> box { X Y Z }' defines the periodic boundary box dimensions for the unit.
# The PDB file from Packmol might not contain a standard CRYST1 record that tleap uses,
# or its dimensions might not be precisely what we want to record in the prmtop.
# We set it to the dimensions used in Packmol (40.0 40.0 40.0 Angstroms).
# These dimensions will be written to the .prmtop and .inpcrd files.
# GROMACS will later read these dimensions from the converted .gro file.
set mixture box { 40.0  40.0  40.0 }

# --- Save Output Files ---

# 'saveamberparm' saves the topology (.prmtop) and coordinates (.inpcrd) for the specified unit.
# <unit>: The tleap unit to save (here, 'mixture').
# <prmtop_file>: Name of the output parameter/topology file (e.g., 'mixture.prmtop').
# <inpcrd_file>: Name of the output coordinate file (e.g., 'mixture.inpcrd').
saveamberparm mixture mixture.prmtop mixture.inpcrd

# 'quit' exits tleap.
quit
```

Listing 9: `tleap` input script (`tleap_mixture.in`) for Methanol-Propanol

Running `tleap`

`tleap` is executed by passing its input script using the `-f` flag:

```
tleap -f tleap_mixture.in
```

Listing 10: Running `tleap`

Interpreting `tleap` Output and `leap.log`

- **Loading Messages:** `tleap` will report loading `leaprc.gaff2`, your `.frcmod` files, and your `.mol2` files. When loading a `.mol2` file, it will typically say "`Reading MOLECULE named PRH`" (or `MOH`), confirming the internal name of the unit defined in the `.mol2` file.
- **PDB Loading:** When loading `mixture_packed.pdb`, it will indicate how many atoms it read.
- **`check mixture` Output:** Pay close attention to any errors or warnings here.
  - "`Atom ... does not have a type.`" or "`Unknown atom type...`": This is serious. It means an atom in your system couldn't be matched to a GAFF2 type, or the type is missing parameters. This usually points to an error in your `.mol2` files or a mismatch between atom names in the PDB and the `.mol2` templates.
  - "`Could not find bond/angle/dihedral parameter...`": This indicates missing force field parameters. Ensure your `.frcmod` files were loaded and are correct. If `parmchk2` didn't flag these as critically missing ("`ATTN`"), `tleap` might still proceed by using generic or zeroed parameters, which is undesirable.
  - "`WARNING: The unperturbed charge of the unit (...) is not integral/zero.`": As discussed, for a large neutral system built with AM1-BCC charges, a very small, non-zero net charge (e.g., `-0.0999`) is common due to floating-point summations and usually acceptable.
- **"Split Residue" Issue (Our previous problem):** Recall that when we initially used `POL` as the residue name for propanol, `tleap` outputted many warnings like:

  ```
  Name change in pdb file residue 1 ; this residue is split into POL and OL.
  ```

  Followed by: "`Unknown residue: OL`" and then "`FATAL: Atom .R<OL ...>.A<HOP ...> does not have a type.`" This indicated that `tleap` was misinterpreting the `POL` residue name from the PDB, possibly due to a name conflict or an internal parsing rule, and was incorrectly dividing the propanol molecules. Changing the residue name to the more unique `PRH` throughout all stages (PDB input to `antechamber`, `-rn PRH` flag, `pol.mol2` content, and `mixture_packed.pdb`) resolved this critical issue. This highlights the sensitivity of `tleap` to residue naming.
- **Final Messages:** A successful `tleap` run for this kind of system will typically exit with "`Errors = 0`" and a small number of warnings (often related to the net charge or unassigned chain types for non-polymeric units).
- **`leap.log`:** This file contains a complete transcript of the `tleap` session, including all commands, outputs, warnings, and errors. It is the primary resource for debugging `tleap` problems.

Output Files from `tleap`

- **`mixture.prmtop`:** The Amber parameter/topology file. This is a text file containing all force field parameters for the entire system.
- **`mixture.inpcrd`:** The Amber coordinate file, containing the initial coordinates of all atoms in the system.
- **`leap.log`:** The log file.

Successfully generating `mixture.prmtop` and `mixture.inpcrd` without fatal errors means you have a complete Amber-format description of your mixed system, ready for the next step: conversion to GROMACS format.

6 Phase 3: Conversion to GROMACS and Simulation Setup

With the `mixture.prmtop` (topology and parameters) and `mixture.inpcrd` (coordinates) files successfully generated by `tleap`, we now have a complete description of our mixed molecular system in the Amber file format. However, our goal is to run the Molecular Dynamics simulations using GROMACS, which uses its own distinct file formats for topology (`.top`) and coordinates (`.gro`). Therefore, a conversion step is necessary.

6.1 3.1. Convert Amber Files to GROMACS Format (InterMol)

Purpose of Conversion and Choice of InterMol

The primary goal here is to translate the information contained in the Amber `mixture.prmtop` and `mixture.inpcrd` files into equivalent GROMACS `mixture_intermol.top` and `mixture_intermol.gro` files. This involves:

- Mapping Amber atom types (defined in GAFF2) to GROMACS atom types.
- Converting bond, angle, dihedral, and improper dihedral parameters to the functional forms and units used by GROMACS.
- Transferring atomic charges and Lennard-Jones parameters.
- Writing out the coordinates in GROMACS `.gro` format.
- Crucially for GROMACS: Structuring the topology (`.top`) file correctly. GROMACS expects separate `[ moleculetype ]` definitions for each distinct type of molecule in the system (e.g., one for `MOH`, one for `PRH`). It then requires a `[ system ]` section to name the overall system and a `[ molecules ]` section at the end to list each `moleculetype` and the number of instances of that molecule present in the simulation.

While AmberTools' ParmEd program can perform Amber-to-GROMACS conversions using its `outparm` command, we found in our specific workflow that it tended to produce a "monolithic" GROMACS `.top` file. This means it often defined the entire system as a single large `[ moleculetype ]` rather than creating distinct `[ moleculetype ]` blocks for `MOH` and `PRH` and then listing them in a final `[ molecules ]` section. This monolithic structure leads to a "No molecules were defined in the system" error when using `gmx grompp`.

To overcome this, we turned to InterMol (`https://github.com/shirtsgroup/InterMol`). InterMol is a Python-based tool specifically designed for converting between various molecular simulation file formats, including Amber and GROMACS. It is generally more robust in correctly identifying and separating different residue types from an Amber `prmtop` into distinct `[ moleculetype ]` definitions in the GROMACS `.top` file and correctly generating the `[ molecules ]` section.

Installing InterMol (if not already done)

InterMol can be installed into your Conda environment:

```
conda install -c conda-forge intermol
```

Listing 11: Installing InterMol with Conda

During our setup, we confirmed that version 0.1.2 was installed via Conda.

The InterMol Python Conversion Script (convert_with_intermol.py)

InterMol can be used via its command-line interface or as a Python library. While the command-line module `python -m intermol.convert` worked reliably for our installed version (see below), for more programmatic control or if the direct CLI has issues, a Python script using its API is an alternative. The following script uses internal API components:

```python
# convert_with_intermol.py
# Author: Mohammad Torikh
# Date: June 21, 2025
# Purpose: Convert Amber prmtop/inpcrd to GROMACS top/gro using InterMol API.

import intermol # Main import to check version and access submodules
import intermol.amber.amber_IO as amber_IO # For reading Amber files
import intermol.gromacs.gromacs_IO as gmx_IO # For writing GROMACS files
from intermol.system import System # The central System object in InterMol

```

```python
print(f"Using InterMol version: {intermol.__version__ if hasattr(intermol, '
    __version__') else 'Unknown (version attribute not found)'}")
print("Starting InterMol conversion...")

# Define input Amber file names
prmtop_file = 'mixture.prmtop'
coord_file = 'mixture.inpcrd'

# Define output GROMACS file names
gromacs_top_out = 'mixture_intermol.top'
gromacs_gro_out = 'mixture_intermol.gro'

# Create an empty InterMol System object. This object will be populated
# with data from the Amber files and then used to write the GROMACS files.
current_system = System(name="methanol_propanol_mixture") # Name is for
    internal tracking

try:
    # === Reading Amber Files ===
    print(f"Reading Amber topology from: {prmtop_file}")
    # Instantiate the Amber parameter/topology reader with the prmtop file
    amber_topology_reader = amber_IO.AmberParameters(prmtop_file)
    # Call its read method to parse the file
    amber_topology_reader.read()
    # Populate our System object with the information from the prmtop
    amber_topology_reader.fill_system(current_system)

    print(f"Reading Amber coordinates from: {coord_file}")
    # Instantiate the Amber coordinate reader with the inpcrd file
    amber_coordinate_reader = amber_IO.AmberCoordinates(coord_file)
    # Call its read method
    amber_coordinate_reader.read()
    # Populate our System object with the coordinate information
    # This links the coordinates to the topology already in current_system
    amber_coordinate_reader.fill_system(current_system)

    print("Amber files successfully loaded into InterMol System object.")

    # === Writing GROMACS Files ===
    print(f"Writing GROMACS topology to: {gromacs_top_out}")
    # Instantiate the GROMACS topology writer, passing it our populated System
        object
    gromacs_topology_writer = gmx_IO.GromacsTopology(current_system)
    # Call its write method, providing the desired output filename
    gromacs_topology_writer.write(gromacs_top_out)

    print(f"Writing GROMACS coordinates to: {gromacs_gro_out}")
    # Instantiate the GROMACS structure (.gro) writer
    gromacs_structure_writer = gmx_IO.GromacsStructure(current_system)
    # Call its write method
    gromacs_structure_writer.write(gromacs_gro_out)

    print("Conversion using InterMol API completed successfully.")

except FileNotFoundError as e:
    print(f"Error: Input file not found - {e}")
except ImportError as e:
    print(f"Error: Could not import an InterMol module. Is InterMol installed
        correctly? - {e}")
    print(f"Attempted to use 'intermol.amber.amber_IO' and 'intermol.gromacs.
        gromacs_IO'.")
except AttributeError as e:
    print(f"Error: An attribute was not found, this might indicate an API
        mismatch with your InterMol version - {e}")
except Exception as e:
    print(f"An unexpected error occurred during InterMol conversion: {e}")
```

Listing 12: Python script for InterMol conversion (`convert_with_intermol.py`)

Alternative InterMol Invocation (Command Line using `-m`)

During our troubleshooting, we found that for InterMol version 0.1.2 installed via Conda, the most straightforward way to perform the conversion was to use its module execution capability:

```
python -m intermol.convert --amb_in mixture.prmtop mixture.inpcrd --gromacs --
    oname mixture_intermol
```

Listing 13: Running InterMol via command line module

`python -m intermol.convert` This tells Python to run the `convert.py` script located within the `intermol` package.

`--amb_in mixture.prmtop mixture.inpcrd` Specifies the input Amber files (topology first, then coordinates).

`--gromacs` A flag indicating that the desired output format is GROMACS.

`--oname mixture_intermol` Sets the base name for the output files. InterMol will create `mixture_intermol.top` and `mixture_intermol.gro`.

This command-line method is often less prone to minor API changes than writing a custom Python script that uses internal InterMol classes. For this documentation, using this command-line approach is recommended for its simplicity if it works with the installed InterMol version.

Expected Output and Verification

InterMol should print some informational messages, including a citation request, and indicate "Finished!" upon successful completion. The output files are:

- `mixture_intermol.top`: The GROMACS topology file.
- `mixture_intermol.gro`: The GROMACS coordinate file.

Crucial Verification of `mixture_intermol.top`

This is the most important check. Open `mixture_intermol.top` with a text editor or use `grep` and `tail` to verify its structure:

- **Separate `[ moleculetype ]` Sections:**

  ```
  grep "\[ moleculetype \]" mixture_intermol.top
  ```

  Listing 14: Checking for moleculetype sections

  You **must** see this line appear twice (once for `MOH`, once for `PRH`).

- **Moleculetype Names:** Check the names assigned by InterMol.

  ```
  grep -A 1 "\[ moleculetype \]" mixture_intermol.top
  ```

  Listing 15: Checking moleculetype names

This should show something like:

```
[ moleculetype ]
MOH          3 ; nrexcl
--
[ moleculetype ]
PRH          3 ; nrexcl
```

Listing 16: Expected moleculetype names in .top file

The names `MOH` and `PRH` should match what `tleap` used.

- **`[ system ]` and `[ molecules ]` Sections:** Check the end of the file.

```
tail -n 10 mixture_intermol.top
```

Listing 17: Checking system and molecules sections

You **must** find a `[ system ]` section followed by a `[ molecules ]` section that correctly lists the number of each molecule type. This section is critical for `gmx grompp`.

```
[ system ]
; Name
methanol_propanol_mixture ; (or whatever name InterMol/the script assigned)

[ molecules ]
; Compound        nmols
MOH                  200
PRH                  100
```

Listing 18: Expected system and molecules sections in .top file

The names `MOH` and `PRH` here must exactly match those in the `[ moleculetype ]` definitions earlier in the file.

If these structural elements are present and correct in `mixture_intermol.top`, then you have successfully converted your Amber system into a GROMACS-compatible format.

6.2 3.2. Energy Minimization (GROMACS)

Purpose and Critical Importance of Energy Minimization

After creating an initial system configuration and converting it to the GROMACS format (`mixture_intermol.gro` and `mixture_intermol.top`), the system is likely to contain some unfavorable atomic arrangements:

- Steric Clashes: Atoms from different molecules, or even within the same flexible molecule, might be too close together.
- Strained Geometries: Bond lengths, angles, or dihedrals might be distorted from their ideal equilibrium values.
- Unfavorable Electrostatic Interactions: Atoms with like charges might be in close proximity.

Starting a Molecular Dynamics (MD) simulation directly from such a high-energy, unrelaxed configuration would result in enormous initial forces, leading to:

- Extremely large atomic displacements.
- Failure of constraint algorithms (like LINCS), as observed with LINCS WARNINGS and subsequent crashes in earlier attempts.
- Numerical instability, causing the simulation to "blow up."

Energy Minimization is a computational process that gently adjusts atomic coordinates to find a nearby local minimum on the potential energy surface. It iteratively moves atoms to reduce net forces, lowering the overall potential energy. Unlike MD, it typically doesn't involve velocities or temperature. It is an **absolutely essential step** before any equilibration or production MD run.

The `minim.mdp` File

GROMACS simulations are controlled by parameters in an `.mdp` file. For energy minimization:

```
; minim.mdp - GROMACS MDP file for Energy Minimization
; Author: Mohammad Torikh
; Date: June 21, 2025

integrator   = steep      ; Algorithm: Steepest Descent, robust for initial
    minimization.
                          ; 'cg' (Conjugate Gradients) can be more efficient
                             later.

emtol        = 1000.0     ; Tolerance for convergence (kJ/mol/nm).
                          ; Stops when max force (Fmax) < this value.
                          ; 1000.0 is common; smaller (e.g., 100.0) for more
                             rigor.

emstep       = 0.01       ; Initial step size for minimization (nm). Max
    displacement per step.

nsteps       = 5000       ; Max number of minimization steps.
                          ; Stops if 'emtol' reached or 'nsteps' completed.
                          ; 5000 steps usually sufficient for Fmax < 1000 kJ/mol/
                             nm.

; Parameters for neighbor searching, electrostatics, and VdW
; Should be generally consistent with subsequent MD runs.
cutoff-scheme     = Verlet     ; Required for modern GROMACS.
nstlist           = 20         ; Freq. to update neighbor list (every 20 steps).
rlist             = 1.2        ; Short-range neighbor list cutoff (nm).
coulombtype       = PME        ; Particle Mesh Ewald for long-range electrostatics
    .
rcoulomb          = 1.2        ; Short-range electrostatic cutoff (nm).
rvdw              = 1.2        ; Short-range van der Waals cutoff (nm).
pbc               = xyz        ; Periodic Boundary Conditions in all directions.
```

Listing 19: GROMACS MDP file for Energy Minimization (`minim.mdp`)

GROMACS Commands for Energy Minimization

1. **`gmx grompp` (The GROMACS Pre-processor):** Compiles input files (`.gro`, `.top`, `.mdp`) into a binary run input file (`.tpr`).

```
gmx grompp -f minim.mdp -c mixture_intermol.gro -p mixture_intermol.top -o
    system_em.tpr -maxwarn 5
```

Listing 20: GROMACS `grompp` for minimization

   - `-f minim.mdp`: Specifies the minimization parameter file.
   - `-c mixture_intermol.gro`: Initial GROMACS coordinate file.
   - `-p mixture_intermol.top`: GROMACS topology file.
   - `-o system_em.tpr`: Output binary run input file.
   - `-maxwarn 5`: Allows a few non-fatal warnings.

   `grompp` performs checks; serious issues will cause errors here.

2. **`gmx mdrun` (The GROMACS Simulation Engine):** Performs the energy minimization using the `.tpr` file.

```
gmx mdrun -deffnm system_em -v
```

Listing 21: GROMACS `mdrun` for minimization

- `-deffnm system_em`: Sets default base filename for input/output.
- `-v`: Verbose output.

Interpreting `mdrun` Output for Minimization

The screen output (and `system_em.log`) will show:

```
Steepest Descents:
   Tolerance (Fmax)   =  1.00000e+03
   Number of steps    =         5000
Step=    0, Dmax= 1.0e-02 nm, Epot=  7.78292e+04 Fmax= 1.79434e+05, atom= 2175
Step=    1, Dmax= 1.0e-02 nm, Epot=  7.13068e+04 Fmax= 5.88890e+04, atom= 2175
...
Step=   40, Dmax= 4.6e-03 nm, Epot=  2.49397e+03 Fmax= 9.15563e+02, atom= 2363

writing lowest energy coordinates.

Steepest Descents converged to Fmax < 1000 in 41 steps
Potential Energy  =  2.4939675e+03
Maximum force     =  9.1556299e+02 on atom 2363
Norm of force     =  8.3557272e+01
```

Listing 22: Example `mdrun` output during minimization

Key things to note:

- `Epot` (Potential Energy) should generally decrease.
- `Fmax` (Maximum Force) should decrease significantly.
- Convergence message: "`Steepest Descents converged to Fmax < 1000...`" indicates success.
- "`writing lowest energy coordinates.`" confirms the minimized structure is saved.

Output Files from Minimization

- `system_em.gro` **(Most Important):** Coordinates after minimization. Input for NVT.
- `system_em.log`: Detailed log.
- `system_em.edr`: Energy file (can plot Epot and Fmax using `gmx energy`).
- `system_em.trr`: Trajectory file (if output steps were set).

If minimization fails, it might indicate severe clashes or force field issues.

6.3 3.3. NVT Equilibration (GROMACS)

Purpose of NVT Equilibration

After energy minimization (`system_em.gro`), the system is at a potential energy minimum but "cold." NVT (constant Number of particles, Volume, Temperature) equilibration aims to:

- **Introduce Kinetic Energy:** Assign initial velocities corresponding to the target temperature.
- **Thermalize the System:** Allow kinetic energy (temperature) to distribute according to Maxwell-Boltzmann.
- **Allow Further Structural Relaxation:** Atoms explore conformational space at target temperature, maintaining the box volume from minimization.

This step is crucial before NPT equilibration and VLE production.

The nvt_equil.mdp File

```
; nvt_equil.mdp - GROMACS MDP file for NVT Equilibration
; Author: Mohammad Torikh
; Date: June 21, 2025
; Purpose: Bring the minimized system to the target temperature at constant volume.

integrator              = md        ; Leap-frog integrator for molecular dynamics.
dt                      = 0.002     ; Timestep in picoseconds (ps). 2 fs is standard.
nsteps                  = 50000     ; Total steps: 50000 * 0.002 ps/step = 100 ps.
                                    ; Common for initial equilibration: 100 ps to 1 ns.

; Output control (nst* values are in number of steps)
nstxout-compressed      = 500       ; Save compressed coordinates (.xtc) every 1 ps.
nstvout                 = 0         ; Do not save velocities for equilibration.
nstenergy               = 500       ; Save energy data to .edr every 1 ps.
nstlog                  = 500       ; Write to log file (.log) every 1 ps.

; Neighbor searching
cutoff-scheme           = Verlet
nstlist                 = 20        ; Freq. to update neighbor list (20 steps = 40 fs).
rlist                   = 1.2       ; Short-range neighbor list cutoff (nm).

; Electrostatics and Van der Waals
coulombtype             = PME
rcoulomb                = 1.2       ; Short-range electrostatic cutoff (nm).
rvdw                    = 1.2       ; Short-range Van der Waals cutoff (nm).
pme_order               = 4         ; Interpolation order for PME (cubic).
fourierspacing          = 0.16      ; Grid spacing for PME FFT (nm).

; Temperature coupling (Thermostat) - CRUCIAL for NVT
tcoupl                  = V-rescale ; Velocity rescaling thermostat. Robust.
tc-grps                 = System    ; Couple all atoms.
tau_t                   = 0.1       ; Time constant for temperature coupling (ps).
ref_t                   = 298.15    ; Reference temperature (Kelvin).

; Pressure coupling (Barostat) - NOT USED IN NVT
; pcoupl                = No        ; Explicitly turn off.

; Periodic Boundary Conditions
pbc                     = xyz

; Initial Velocity Generation
gen_vel                 = yes       ; Generate velocities from Maxwell distribution.
gen_temp                = 298.15    ; Temperature for Maxwell distribution (K), match ref_t.
gen_seed                = -1        ; Seed for random number generator (-1 for unique).

; Constraints - Essential for 2 fs timestep
constraints             = h-bonds   ; Constrain bonds involving hydrogen.
constraint_algorithm    = lincs     ; LINCS algorithm.
lincs_iter              = 1         ; Number of iterations for LINCS.
```

```
lincs_order                = 4          ; Order of LINCS.
```

Listing 23: GROMACS MDP file for NVT Equilibration (`nvt_equil.mdp`)

GROMACS Commands for NVT Equilibration

1. `gmx grompp`:

   ```
   gmx grompp -f nvt_equil.mdp -c system_em.gro -p mixture_intermol.top -o
       system_nvt.tpr -maxwarn 5
   ```

   Listing 24: GROMACS `grompp` for NVT equilibration

   Uses `system_em.gro` (minimized coordinates).

2. `gmx mdrun`:

   ```
   gmx mdrun -deffnm system_nvt -v
   ```

   Listing 25: GROMACS `mdrun` for NVT equilibration

Monitoring NVT Equilibration

- **Temperature:** Use `gmx energy -f system_nvt.edr -o temperature_nvt.xvg` (select Temperature). Should stabilize around `ref_t`.
- **Potential Energy:** Use `gmx energy -f system_nvt.edr -o potential_energy_nvt.xvg` (select Potential). Should reach a stable fluctuation.
- **LINCS Warnings:** Check `system_nvt.log`. Should be absent or rare after minimization.

Output Files from NVT Equilibration

- `system_nvt.gro`: Coordinates at the end of NVT.
- `system_nvt.xtc`: Trajectory.
- `system_nvt.edr`: Energy file.
- `system_nvt.cpt` **(Most Important for next step):** Checkpoint file, saves full system state.

Once temperature and potential energy are stable, the system is ready for NPT.

6.4 3.4. NPT Equilibration (GROMACS)

Purpose of NPT Equilibration

Following NVT, NPT (constant Number of particles, Pressure, Temperature) equilibration aims to:

- **Achieve Target Pressure and Density:** Allow system volume to fluctuate to reach equilibrium density at the specified target pressure and temperature.
- **Further System Relaxation:** Box size changes allow more extensive structural relaxation.
- **Prepare for Slab Creation:** Critical to start VLE with a liquid phase at the correct, stable density.

The `npt.mdp` File

Key consideration: choice of barostat. Berendsen is good for initial stability; Parrinello-Rahman for production NPT if needed later. We use Berendsen here.

```
; npt.mdp - GROMACS MDP file for NPT Equilibration
; Author: Mohammad Torikh
; Date: June 21, 2025
; Purpose: Equilibrate system density at target temperature and pressure.

integrator               = md
dt                       = 0.002      ; 2 fs timestep
nsteps                   = 500000     ; 1 ns (500,000 * 0.002 ps). Monitor density; may need 1-10 ns.

; Output control
nstxout-compressed       = 5000       ; Save coordinates every 10 ps
nstenergy                = 5000       ; Save energies every 10 ps
nstlog                   = 5000       ; Update log file every 10 ps

; Neighbor searching (can be same as NVT)
cutoff-scheme            = Verlet
nstlist                  = 20
rlist                    = 1.2

; Electrostatics and VdW (can be same as NVT)
coulombtype              = PME
rcoulomb                 = 1.2
rvdw                     = 1.2
pme_order                = 4
fourierspacing           = 0.16

; Temperature coupling (Thermostat)
tcoupl                   = V-rescale
tc-grps                  = System
tau_t                    = 0.1
ref_t                    = 298.15     ; Target temperature (Kelvin)

; Pressure coupling (Barostat) - CRUCIAL for NPT
pcoupl                   = Berendsen  ; Berendsen barostat for stable equilibration.
pcoupltype               = isotropic  ; Couple pressure isotropically.
tau_p                    = 2.0        ; Time constant for pressure coupling (ps). Typical: 0.5-5 ps.
ref_p                    = 1.0        ; Reference pressure (bar).
compressibility          = 4.5e-5     ; Compressibility of system (bar^-1). Water's value is often used.

; Periodic Boundary Conditions
pbc                      = xyz

; Initial Velocity Generation
gen_vel                  = no         ; IMPORTANT: Continue from NVT checkpoint file (.cpt).

; Constraints
constraints              = h-bonds
constraint_algorithm     = lincs
lincs_iter               = 1
lincs_order              = 4
```

Listing 26: GROMACS MDP file for NPT Equilibration (`npt.mdp`)

GROMACS Commands for NPT Equilibration

1. `gmx grompp`:

```
gmx grompp -f npt.mdp -c system_nvt.gro -p mixture_intermol.top -t
    system_nvt.cpt -o system_npt.tpr -maxwarn 5
```

Listing 27: GROMACS `grompp` for NPT equilibration

   - `-c system_nvt.gro`: Starting coordinates from NVT.
   - `-t system_nvt.cpt`: Crucial for continuity; reads full state from NVT.

2. `gmx mdrun`:

```
gmx mdrun -deffnm system_npt -v
```

Listing 28: GROMACS `mdrun` for NPT equilibration

Monitoring NPT Equilibration

- **Density:** Most important. Use `gmx energy -f system_npt.edr -o density_npt.xvg` (select Density). Must converge to a stable average.
- **Pressure:** Check with `gmx energy -f system_npt.edr -o pressure_npt.xvg` (select Pressure). Should fluctuate around `ref_p`.
- **Potential Energy, Temperature, Volume:** Also monitor for stability.

Extending NPT Runs (if needed): If density isn't stable, extend the run.

1. Update `npt.mdp` for more steps.
2. Use `gmx convert-tpr` to create a new `.tpr` from the previous one:

```
# Example: extend by another 1 ns (500,000 steps if dt=0.002)
gmx convert-tpr -s system_npt.tpr -extend 1000 -o system_npt_extend.tpr
# -extend value is in ps
```

Listing 29: Extending NPT run with `convert-tpr`

3. Run `mdrun` with the new `.tpr`, continuing from the previous checkpoint:

```
gmx mdrun -deffnm system_npt_extend -cpi system_npt.cpt -v
```

Listing 30: Running extended NPT simulation

   Alternatively, use `gmx mdrun -deffnm system_npt -cpi system_npt.cpt -append` if `nsteps` in the original `.tpr` was large enough or `mdrun` is allowed to run longer by other means.

Output Files from NPT Equilibration

- `system_npt.gro`: Coordinates at end of NPT (should have equilibrated density).
- `system_npt.xtc`: Trajectory.
- `system_npt.edr`: Energy file (density, pressure, volume, etc.).
- `system_npt.cpt`: Final checkpoint file.

Once density is stable, the system is ready for slab creation.

7 Phase 4: VLE Simulation Setup & Production

Having successfully equilibrated our binary mixture in the NPT ensemble (resulting in `system_npt.gro` with an equilibrated liquid density), we are now ready to create the specific geometry required to observe VLE and then run the long production simulation.

7.1 4.1. Create Slab Geometry (`gmx editconf`)

Purpose of the Slab Geometry for VLE Simulations

As discussed previously, to simulate the coexistence of liquid and vapor phases, we need to create a system where both can physically exist and interact. The "slab geometry" is the standard approach for this in MD simulations. It involves:

- Starting with a well-equilibrated bulk liquid phase (from our NPT run).
- Placing this liquid slab into a simulation box that is significantly elongated in one dimension (typically chosen as the Z-axis).
- This elongation creates empty regions (vacuum) above and below the liquid slab along the Z-axis.

When an MD simulation is run on this slab system (usually under NVT conditions, as the overall box volume is now fixed and large), molecules from the liquid surface will evaporate into the vacuum regions, eventually forming a distinct vapor phase. Over sufficient simulation time, a dynamic equilibrium will be established, with molecules continuously transitioning between the liquid and vapor phases. This setup allows us to directly sample the properties of both coexisting phases.

The `gmx editconf` Tool

GROMACS provides the `gmx editconf` tool to manipulate simulation box dimensions and molecule orientations. We will use it to take our NPT-equilibrated box and stretch it along the Z-axis.

Steps to Create the Slab

1. **Determine the Final Box Dimensions from the NPT Run:** The last line of your `system_npt.gro` file contains the final box vectors from the NPT equilibration. For an orthogonal box, these are the X, Y, and Z dimensions of the liquid phase at equilibrium.

```
tail -n 1 system_npt.gro
```

Listing 31: Getting final box dimensions from `system_npt.gro`

   Let's assume this command outputs, for example: `3.03201 3.03201 3.03201` So, $X_{\text{liquid}} = 3.03201$ nm, $Y_{\text{liquid}} = 3.03201$ nm, and $Z_{\text{liquid}} = 3.03201$ nm.

2. **Decide on the New Elongated Z Dimension (Z_{new}):** The Z-dimension of the slab box (Z_{new}) needs to be large enough to accommodate the liquid slab and two sufficiently large vapor regions to minimize finite-size effects or interactions between the periodic images of the slab across the Z-boundary. A common rule of thumb is to make Z_{new} at least 3 to 5 times Z_{liquid}. For our example, if $Z_{\text{liquid}} = 3.03201$ nm, let's choose $Z_{\text{new}} = 4 \times Z_{\text{liquid}}$. $Z_{\text{new}} = 4 \times 3.03201$ nm $= 12.12804$ nm. The X and Y dimensions will remain the same as X_{liquid} and Y_{liquid}.

3. **Calculate Centering Coordinates for the New Box:** We want our liquid slab to be centered within this new, elongated box. The centering coordinates are half of the new box dimensions. center_X $= X_{\text{liquid}}/2 = 3.03201/2 = 1.516005$ nm center_Y $= Y_{\text{liquid}}/2 = 3.03201/2 = 1.516005$ nm center_Z$_{\text{new}}$ $= Z_{\text{new}}/2 = 12.12804/2 = 6.06402$ nm

4. **Execute `gmx editconf`:**

```
gmx editconf -f system_npt.gro -o system_slab_initial.gro -box 3.03201
    3.03201 12.12804 -center 1.516005 1.516005 6.06402
```

Listing 32: Using `gmx editconf` to create slab geometry

Explanation of Options:

- `-f system_npt.gro`: Specifies the input coordinate file (final structure from NPT).
- `-o system_slab_initial.gro`: Output coordinate file with slab geometry.
- `-box` $X_{\texttt{liquid}}$ $Y_{\texttt{liquid}}$ $Z_{\texttt{new}}$: Sets new box dimensions (nm).
- `-center center_X center_Y center_Z`$_{\texttt{new}}$: Places the geometric center of molecules at these coordinates in the new box, ensuring the slab is centered.

Output of `gmx editconf`

The command will print information about the old box, the shift applied, and the new box dimensions/volume. Verify that the ”new box vectors” match your specifications. Example output:

```
new center       :   1.516   1.516   6.064 (nm)
new box vectors :   3.032   3.032 12.128 (nm)
new box volume  : 111.49                 (nm^3)
```

Listing 33: Example output from `gmx editconf`

The primary output is `system_slab_initial.gro`. This `.gro` file now contains your molecules arranged as a liquid slab, centered in a box elongated in Z.

Verification (Recommended)

- **Check Box Dimensions:**

```
tail -n 1 system_slab_initial.gro
```

Listing 34: Checking new box dimensions from `system_slab_initial.gro`

  The last line should reflect the new elongated box dimensions (e.g., `3.03201 3.03201 12.12804`).

- **Visual Inspection:** Load `system_slab_initial.gro` into a molecular visualizer (e.g., VMD, PyMOL). You should clearly see:
  - Compact X and Y dimensions.
  - A much longer Z dimension.
  - Molecules clustered as a ”slab” in the middle of the Z-range.
  - Significant empty (vacuum) space above and below the slab along Z.

This `system_slab_initial.gro` file is the starting point for your VLE production simulation.

7.2 4.2. VLE Production Run (NVT, GROMACS)

Purpose of the VLE Production Run

With the slab geometry established (`system_slab_initial.gro`), this long Molecular Dynamics simulation is where the actual Vapor-Liquid Equilibrium is simulated and data for its characterization is collected. Key objectives:

- **Phase Separation:** Molecules from the liquid slab evaporate into vacuum regions, forming a vapor phase, while vapor molecules condense back.
- **Equilibrium Establishment:** Over time, evaporation and condensation rates equalize, leading to dynamic equilibrium. Properties of coexisting phases (densities, compositions, pressure) stabilize.

- **Data Collection:** Generate trajectory (`.xtc` or `.trr`) and energy (`.edr`) files for analysis in Phase 8.

This simulation is typically run under NVT conditions (constant Number of particles, Volume, Temperature).

- **N (Number of particles):** Constant.
- **V (Volume):** Overall simulation box volume (from `gmx editconf`) is constant.
- **T (Temperature):** A thermostat maintains the desired VLE temperature.

Pressure is not explicitly controlled. The pressure that develops, particularly in the equilibrated vapor phase, is the vapor pressure (P_{vap}) of the mixture at the chosen temperature.

The `vle_nvt.mdp` File

Similar to `nvt_equil.mdp` but with a significantly longer run time (`nsteps`) and potentially adjusted output frequencies.

```
; vle_nvt.mdp - GROMACS MDP file for VLE Production Run
; Author: Mohammad Torikh
; Date: June 21, 2025
; Purpose: Simulate two-phase coexistence to determine VLE properties.

integrator               = md         ; Leap-frog integrator.
dt                       = 0.002      ; Timestep (ps) (2 fs).
nsteps                   = 10000000   ; Total steps: 10,000,000 * 0.002 ps/step =
    20 ns.
                                      ; VLE often requires 10s to 100s of ns.
                                      ; For testing, use shorter run (e.g., 1-2
                                        ns).

; Output control - Write less frequently for long runs.
nstxout-compressed       = 10000      ; Save .xtc every 20 ps (10000 steps).
nstvout                  = 0          ; No velocities for full VLE trajectory.
nstfout                  = 0          ; No forces.
nstenergy                = 10000      ; Save .edr every 20 ps.
nstlog                   = 10000      ; Write to .log every 20 ps.

; Neighbor searching
cutoff-scheme            = Verlet
nstlist                  = 20         ; Update neighbor list every 40 fs.
rlist                    = 1.2        ; Short-range neighbor list cutoff (nm).

; Electrostatics and Van der Waals
coulombtype              = PME
rcoulomb                 = 1.2
rvdw                     = 1.2
pme_order                = 4
fourierspacing           = 0.16       ; GROMACS determines PME grid. For
    anisotropic boxes
                                      ; (like slab), grompp might choose e.g. 20
                                        x20x80.
                                      ; If PME load is high (>35-40%), manual
                                        tuning of
                                      ; fourier_nx, fourier_ny, fourier_nz or
                                        fourierspacing
                                      ; might be needed for optimization.

; Temperature coupling (Thermostat)
tcoupl                   = V-rescale
tc-grps                  = System
tau_t                    = 0.1        ; Coupling time constant (ps).
```

```
ref_t                   = 298.15    ; Target temperature (K) for VLE. Key variable.

; Pressure coupling - NONE for NVT
; pcoupl                  = No

; Periodic Boundary Conditions
pbc                     = xyz       ; 3D PBC essential.

; Initial Velocity Generation
gen_vel                 = yes       ; Generate velocities from Maxwell distribution.
                                    ; Starting from system_slab_initial.gro (no velocities).
gen_temp                = 298.15    ; Temperature for Maxwell distribution (K), match ref_t.
gen_seed                = -1        ; Random seed. Use integer for reproducibility.

; Constraints
constraints             = h-bonds
constraint_algorithm    = lincs
lincs_iter              = 1
lincs_order             = 4
```

Listing 35: GROMACS MDP file for VLE Production Run (`vle_nvt.mdp`)

Explanation of Key `vle_nvt.mdp` Parameters

- **`nsteps` (Critical):** Determines total simulation time. VLE often requires long timescales (tens to hundreds of ns). A 20 ns run (`nsteps = 10000000` with `dt = 0.002`) is a reasonable start, but longer might be needed. For initial testing, use a much shorter `nsteps`.
- **`nstxout-compressed`, `nstenergy`, `nstlog`:** Output frequencies reduced for long runs to save disk space (e.g., every 10-50 ps).
- **`fourierspacing` / PME Grid:** `grompp` sets PME grid. For anisotropic boxes, PME can be a bottleneck. If "Estimate for the relative computational load of the PME mesh part" from `grompp` is high (>0.4), manual tuning of `fourier_nx`, `fourier_ny`, `fourier_nz` or `fourierspacing` might be needed for better load balancing in future optimized runs.
- **`ref_t`:** Temperature for VLE properties. To build a T-xy diagram, repeat VLE simulation at different `ref_t` values.
- **`gen_vel = yes`:** Since `system_slab_initial.gro` is static, GROMACS generates initial velocities.

GROMACS Commands for VLE Production Run

1. **`gmx grompp`:**

   ```
   gmx grompp -f vle_nvt.mdp -c system_slab_initial.gro -p mixture_intermol.top -o system_vle.tpr -maxwarn 5
   ```

   Listing 36: GROMACS `grompp` for VLE production

   Uses `system_slab_initial.gro` (slab coordinates) and `mixture_intermol.top`. Review `grompp` output, especially PME grid/load estimate.

2. **`gmx mdrun` (The Long Run):**

```bash
nohup gmx mdrun -deffnm system_vle -v > mdrun_vle.out 2>&1 &
```

Listing 37: GROMACS `mdrun` for VLE production

- `-deffnm system_vle`: Uses `system_vle.tpr`; produces `system_vle.log, .edr, .xtc, .gro, .cpt`.
- `nohup ... &`: Recommended for background execution on a server, redirecting output.

Monitoring the VLE Production Run

- **`mdrun_vle.out` and `system_vle.log`:** Use `tail -f` to monitor progress.
- **Trajectory Visualization (Occasionally):** After significant time (e.g., few ns), transfer `system_vle.xtc` and a coordinate/topology file (`system_slab_initial.gro` or `system_vle.tpr`) to visualize with VMD/PyMOL. Check for:
  - Clear interface formation.
  - Molecules in the vapor phase.
  - System stability (no blow-ups, liquid not drifting due to PBC issues if X/Y box too small).
- **Key Energies/Properties (using `gmx energy`):**
  - Temperature: Should fluctuate around `ref_t`.
  - Pressure: Will fluctuate. Average over equilibrated vapor phase is P_{vap}.
  - Potential Energy and Density: Will change during interface formation, then stabilize. Density will show two plateaus in the profile.

This VLE production run is where the system evolves towards equilibrium. Patience is key for these computationally intensive runs.

8 Phase 5: Analysis and Interpretation

After your long VLE production run (`gmx mdrun -deffnm system_vle`) has completed successfully, you will have a set of output files (primarily `system_vle.xtc`, `system_vle.edr`, and `system_vle.tpr`) that contain the raw data about your system's behavior at the simulated temperature. The goal of this phase is to analyze this data to determine the properties of the coexisting liquid and vapor phases, which constitute one point on your VLE diagram.

8.1 5.1. Density Profiles (`gmx density`)

Purpose

The most direct way to confirm the formation of distinct liquid and vapor phases and to determine their boundaries and compositions is by calculating the density profile of each component (and the total system) along the Z-axis (the dimension perpendicular to the liquid-vapor interfaces).

- A high-density plateau in the center of the box indicates the liquid slab.
- Low-density plateaus at the ends of the box indicate the vapor phases.
- The regions of transition between these plateaus are the liquid-vapor interfaces.

The `gmx density` Tool

GROMACS provides the `gmx density` tool for this purpose.

Commands to Generate Density Profiles

You will run `gmx density` separately for each component and for the total system. It's important to analyze a significant portion of your trajectory, potentially discarding an initial part if the system was still evolving towards phase equilibrium (e.g., if interface formation took some nanoseconds). The `-b` (begin time) and `-e` (end time) flags in `gmx density` can be used for this, but for a first pass, analyzing the whole trajectory is common.

For Methanol (`MOH`):

```
gmx density -f system_vle.xtc -s system_vle.tpr -d Z -o density_Z_MOH.xvg -sl
    120
```

Listing 38: Generating methanol density profile with `gmx density`

- `-f system_vle.xtc`: Input trajectory file.
- `-s system_vle.tpr`: Input run file (topology and system information).
- `-d Z`: Calculate density along the Z-axis.
- `-o density_Z_MOH.xvg`: Output file for methanol density profile (`.xvg` is Grace/XMGRACE format).
- `-sl 120`: Number of slices along Z-axis. For a $\approx$ 12 nm box, 120 slices give 0.1 nm resolution. Adjust for reasonable resolution (0.05 to 0.1 nm/slice often good).

Interactive Prompt: `gmx density` will list atom groups (e.g., System, Other, MOH, PRH). Select the group for Methanol (e.g., by typing its number). In our case, `MOH` was group 2.

For Propanol (`PRH`):

```
gmx density -f system_vle.xtc -s system_vle.tpr -d Z -o density_Z_PRH.xvg -sl
    120
```

Listing 39: Generating propanol density profile with `gmx density`

When prompted, select the group for Propanol (e.g., `PRH`, group 3).

For the Total System:

```
gmx density -f system_vle.xtc -s system_vle.tpr -d Z -o density_Z_total.xvg -sl
     120
```

Listing 40: Generating total system density profile with `gmx density`

When prompted, select the group for the entire System (usually group 0 or 1).

Output `.xvg` Files

Each command produces an `.xvg` file (plain text):

- Lines starting with `#` or `@` are comments/metadata.
- Data lines:
  - Column 1: Z-coordinate (nm) of the slice center.
  - Column 2: Density (kg/m^3) in that slice, averaged over analyzed frames.

Visualizing Density Profiles (e.g., with Gnuplot)

Use Gnuplot to visualize profiles and save images.

```
# Start gnuplot:
# gnuplot

# Plotting individual profiles and saving:
set terminal pngcairo size 800,600 enhanced font "arial,10"
set output 'density_profile_MOH.png'
set title "Density Profile of Methanol (MOH) along Z-axis"
set xlabel "Z-coordinate (nm)"
set ylabel "Density (kg/m^3)"
set grid
set yrange [0:*] # Ensure y-axis starts at 0, auto-scale max
plot 'density_Z_MOH.xvg' using 1:2 with lines linewidth 2 title 'Methanol'
unset output

set output 'density_profile_PRH.png'
set title "Density Profile of Propanol (PRH) along Z-axis"
# (xlabel, ylabel, etc. are usually retained)
plot 'density_Z_PRH.xvg' using 1:2 with lines linewidth 2 title 'Propanol'
unset output

set output 'density_profile_total.png'
set title "Total System Density Profile along Z-axis"
plot 'density_Z_total.xvg' using 1:2 with lines linewidth 2 title 'Total System
    '
unset output

# Overlaying MOH and PRH (most informative):
set output 'density_profile_overlay_MOH_PRH.png'
set title "Component Density Profiles along Z-axis"
plot 'density_Z_MOH.xvg' using 1:2 with lines linewidth 2 title 'Methanol (MOH)
    ', \
     'density_Z_PRH.xvg' using 1:2 with lines linewidth 2 title 'Propanol (PRH)
        '
unset output

# Exit gnuplot:
# exit
```

Listing 41: Gnuplot script for visualizing density profiles

Download these `.png` files to view them.

Interpreting the Density Profile Plots

- **Identify Phases:** Clearly see a high-density region (liquid slab) in the center and two low-density regions (vapor phases) at the ends. Example: If Z-box is 0-12 nm, liquid might be Z=3-9 nm, vapor Z=0-3 nm and Z=9-12 nm.
- **Check for Symmetry:** Ideally, two vapor phases (and interfaces) should be roughly symmetrical if well-equilibrated.
- **Flat Plateaus:** Density within bulk liquid and vapor regions should be relatively flat. A consistent slope might indicate incomplete equilibration or non-bulk region analysis.
- **Interface Width:** The transition region width is also a property of interest.

This visual inspection is crucial. If no clear phase separation, the simulation might need to run longer or other issues exist. Assuming clear phases, proceed to quantitative analysis.

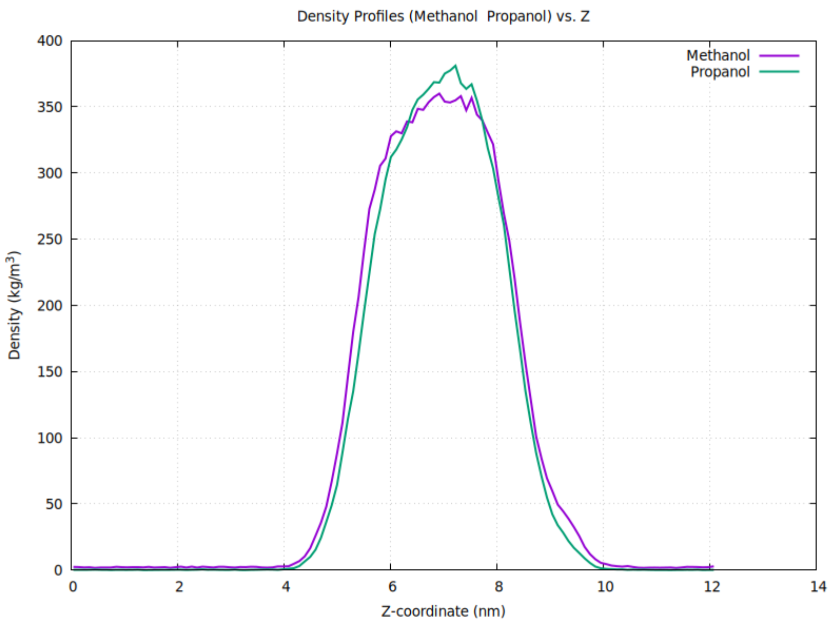

Figure 2: Example density profiles for Methanol and Propanol along the Z-axis, clearly showing the higher density liquid slab in the center and the low-density vapor phases at the ends of the box. The Z-axis represents the dimension perpendicular to the liquid-vapor interfaces.

8.2 5.2. Calculating Phase Compositions & Vapor Pressure

From density profiles and the energy file, calculate VLE properties: compositions of coexisting liquid/-vapor phases, and vapor pressure.

1. Determining Equilibrium Densities and Phase Boundaries

Examine density profile plots (e.g., `density_Z_total.xvg`):

- Identify Z-coordinates for the bulk liquid phase (center, flat plateau). Example: $Z_{\text{liquid\_start}} = 4.0$ nm, $Z_{\text{liquid\_end}} = 8.0$ nm.
- Identify Z-coordinates for bulk vapor phases (ends, low density, flat plateaus). Example: $Z_{\text{vapor1\_start}} = 0.5$ nm, $Z_{\text{vapor1\_end}} = 2.5$ nm AND $Z_{\text{vapor2\_start}} = 9.5$ nm, $Z_{\text{vapor2\_end}} = 11.5$ nm. Average over both vapor regions if symmetrical.
- Choose regions away from rapidly changing interfaces.

Extract Average Densities from .xvg Files

The `.xvg` files contain Z-coordinate (col 1) and density (kg/m^3, col 2). Calculate average density for each component (`MOH`, `PRH`) and total system within identified liquid/vapor regions.

Method A: Manual/Spreadsheet (for a few points): Open `.xvg` (e.g., `density_Z_MOH.xvg`), identify lines for chosen Z-ranges, copy density values (col 2), and average.

Method B: Scripting (Recommended): Use Python, Awk, etc., to read `.xvg`, filter lines by Z-range, and average densities. Conceptual Python Example:

```python
def get_average_density(xvg_filepath, z_start, z_end):
    densities_in_range = []
    with open(xvg_filepath, 'r') as f:
        for line in f:
            if line.startswith('#') or line.startswith('@'):
                continue # Skip comments
            parts = line.split()
            if len(parts) < 2: continue # Skip malformed lines
            try:
                z_coord = float(parts[0])
                density_val = float(parts[1]) # Corrected variable name
                if z_start ≤ z_coord ≤ z_end:
                    densities_in_range.append(density_val)
            except ValueError:
                continue # Skip lines where conversion to float fails
    if not densities_in_range:
        return 0.0 # Or raise an error, or return None
    return sum(densities_in_range) / len(densities_in_range)

# Define your Z-ranges based on visual inspection of plots
z_liq_start, z_liq_end = 4.0, 8.0
z_vap1_start, z_vap1_end = 0.5, 2.5
z_vap2_start, z_vap2_end = 9.5, 11.5

# Calculate average densities
avg_rho_MOH_liq = get_average_density('density_Z_MOH.xvg', z_liq_start,
    z_liq_end)
avg_rho_PRH_liq = get_average_density('density_Z_PRH.xvg', z_liq_start,
    z_liq_end)

avg_rho_MOH_vap_1 = get_average_density('density_Z_MOH.xvg', z_vap1_start,
    z_vap1_end)
avg_rho_MOH_vap_2 = get_average_density('density_Z_MOH.xvg', z_vap2_start,
    z_vap2_end)
avg_rho_MOH_vap = (avg_rho_MOH_vap_1 + avg_rho_MOH_vap_2) / 2.0 if
    avg_rho_MOH_vap_1 is not None and avg_rho_MOH_vap_2 is not None else 0.0

avg_rho_PRH_vap_1 = get_average_density('density_Z_PRH.xvg', z_vap1_start,
    z_vap1_end)
avg_rho_PRH_vap_2 = get_average_density('density_Z_PRH.xvg', z_vap2_start,
    z_vap2_end)
avg_rho_PRH_vap = (avg_rho_PRH_vap_1 + avg_rho_PRH_vap_2) / 2.0 if
    avg_rho_PRH_vap_1 is not None and avg_rho_PRH_vap_2 is not None else 0.0


print(f"Avg Liquid MOH density: {avg_rho_MOH_liq:.4f} kg/m^3")
print(f"Avg Liquid PRH density: {avg_rho_PRH_liq:.4f} kg/m^3")
print(f"Avg Vapor MOH density: {avg_rho_MOH_vap:.4f} kg/m^3")
print(f"Avg Vapor PRH density: {avg_rho_PRH_vap:.4f} kg/m^3")
```

Listing 42: Python script to calculate average density from .xvg

You will obtain: $\langle\rho_{\text{MOH,liquid}}\rangle$, $\langle\rho_{\text{PRH,liquid}}\rangle$, $\langle\rho_{\text{MOH,vapor}}\rangle$, $\langle\rho_{\text{PRH,vapor}}\rangle$.

2. Calculate Mole Fractions in Each Phase (x_i, y_i)

Convert average mass densities to mole fractions. **Molar Masses (M):**

- Methanol (CH_3OH): $M_{\text{MOH}} \approx 32.04$ g/mol $= 0.03204$ kg/mol
- 1-Propanol ($CH_3CH_2CH_2OH$): $M_{\text{PRH}} \approx 60.10$ g/mol $= 0.06010$ kg/mol

Calculate Molar Concentrations ($c = \rho/M$) in mol/m^3:

$$\begin{aligned}
c_{\text{MOH,liquid}} &= \langle\rho_{\text{MOH,liquid}}\rangle / M_{\text{MOH}} \\
c_{\text{PRH,liquid}} &= \langle\rho_{\text{PRH,liquid}}\rangle / M_{\text{PRH}} \\
c_{\text{MOH,vapor}} &= \langle\rho_{\text{MOH,vapor}}\rangle / M_{\text{MOH}} \\
c_{\text{PRH,vapor}} &= \langle\rho_{\text{PRH,vapor}}\rangle / M_{\text{PRH}}
\end{aligned}$$

Calculate Mole Fractions:

- Liquid Phase:

$$x_{\text{MOH}} = c_{\text{MOH,liquid}}/(c_{\text{MOH,liquid}} + c_{\text{PRH,liquid}})$$
$$x_{\text{PRH}} = c_{\text{PRH,liquid}}/(c_{\text{MOH,liquid}} + c_{\text{PRH,liquid}})$$
$$(\text{Check: } x_{\text{MOH}} + x_{\text{PRH}} \approx 1)$$

- Vapor Phase:

$$y_{\text{MOH}} = c_{\text{MOH,vapor}}/(c_{\text{MOH,vapor}} + c_{\text{PRH,vapor}})$$
$$y_{\text{PRH}} = c_{\text{PRH,vapor}}/(c_{\text{MOH,vapor}} + c_{\text{PRH,vapor}})$$
$$(\text{Check: } y_{\text{MOH}} + y_{\text{PRH}} \approx 1)$$

You now have equilibrium mole fractions ($x_{\text{MOH}}, y_{\text{MOH}}$) for methanol.

3. Determine the Vapor Pressure (P_{vap})
The VLE production run was NVT. Average pressure in the equilibrated vapor phase is P_{vap}. Use `gmx energy`:

```
gmx energy -f system_vle.edr -o pressure_vle.xvg
```

Listing 43: Extracting pressure data with `gmx energy`

Select "Pressure" (and optionally "Time"). **Analyze `pressure_vle.xvg`:** Plot Pressure vs. Time. Discard initial non-equilibrated portion. Calculate average pressure over the stable, equilibrated portion. This is P_{vap}. GROMACS usually outputs pressure in bar. Conceptual Python Example:

```python
def get_average_pressure(xvg_filepath, time_start_ps):
    pressures_in_range = []
    with open(xvg_filepath, 'r') as f:
        for line in f:
            if line.startswith('#') or line.startswith('@'):
                continue
            parts = line.split()
            if len(parts) < 2: continue
            try:
                time_ps = float(parts[0])
                pressure_val = float(parts[1]) # Corrected variable name
                if time_ps ≥ time_start_ps:
                    pressures_in_range.append(pressure_val)
            except ValueError:
                continue
    if not pressures_in_range:
        return 0.0 # Or raise an error, or return None
    return sum(pressures_in_range) / len(pressures_in_range)

# Example: Start averaging after 5 ns (5000 ps) of VLE run
avg_Pvap = get_average_pressure('pressure_vle.xvg', 5000.0)
print(f"Average Vapor Pressure (Pvap): {avg_Pvap:.4f} bar")
```

Listing 44: Python script to calculate average pressure from .xvg

Summary of Data for One VLE Point
From one VLE simulation at temperature T, you have:

- Liquid phase mole fraction of methanol: x_{MOH}
- Vapor phase mole fraction of methanol: y_{MOH}

- Vapor pressure of the mixture: P_{vap}

This set $(T, P_{\text{vap}}, x_{\text{MOH}}, y_{\text{MOH}})$ is one tie-line.

Uncertainty Estimation (Advanced)
For rigorous work, estimate uncertainties:

- **Block Averaging:** Divide equilibrated trajectory into blocks, calculate property for each block, then standard error of the mean.
- Analyzing fluctuations over time.

This provides error bars for VLE data points.

8.3 5.3. Building VLE Diagrams

Phases 4-7 and Sections 8.1-8.2 detail a single VLE simulation at one temperature and initial composition, yielding $(T, P_{\text{vap}}, x_i, y_i)$. To construct VLE diagrams (T-xy, P-xy, xy), repeat the process, varying temperature or initial composition.

1. Generating Data for a T-xy Diagram
Shows bubble/dew point temperatures vs. composition at fixed external pressure. **Strategy with NVT Slab Simulations:** NVT slab simulations determine their own P_{vap}. To approximate a T-xy diagram at a target pressure (e.g., 1 atm = 1.01325 bar):

1. Choose temperatures $(T_1, T_2, \dots)$ spanning the expected boiling range.
2. For each T_i:
   - Perform full VLE workflow (equilibration, slab, NVT VLE production at T_i, analysis).
   - Obtain $(x_{\text{MOH},i}, y_{\text{MOH},i}, P_{\text{vap},i})$.

Data Collection Table: Plotting T-xy Diagram:

Table 1: Example data for T-xy diagram construction.

Temperature (K) (T_i)	$x_{\text{MOH},i}$	$y_{\text{MOH},i}$	$P_{\text{vap},i}$ (bar)
T_1	x_1	y_1	P_1
T_2	x_2	y_2	P_2
...	...	...	...

- For exactly 1 atm: Find, for several overall compositions, the temperature where $P_{\text{vap}} = 1$ atm (complex, involves multiple simulations and interpolation).
- More direct plot from NVT slabs: Plot T_i (y-axis) vs. $x_{\text{MOH},i}$ and $y_{\text{MOH},i}$ (x-axis).
  - Bubble point curve: T_i vs. $x_{\text{MOH},i}$.
  - Dew point curve: T_i vs. $y_{\text{MOH},i}$.

  These curves represent VLE at self-generated $P_{\text{vap},i}$.

2. Generating Data for a P-xy Diagram (at Constant Temperature)
Shows bubble/dew point pressures vs. composition at fixed temperature. **Strategy:**

1. Fix simulation temperature (T).
2. Vary Overall Initial Composition (in `packmol.inp`, Section 5.1). Examples:
   - Run 1: High Methanol (e.g., 80% MOH)
   - Run 2: Medium Methanol (e.g., 50% MOH)
   - Run 3: Low Methanol (e.g., 20% MOH)

3. For each initial composition:
    - Perform full VLE workflow at fixed T.
    - Obtain coexisting $(x_{\text{MOH},i}, y_{\text{MOH},i})$ and $P_{\text{vap},i}$.

Data Collection Table (fixed T): Plotting P-xy Diagram: Plot $P_{\text{vap},i}$ (y-axis) vs. $x_{\text{MOH},i}$ and

Table 2: Example data for P-xy diagram construction (fixed T).

Overall Start (approx.)	$x_{\text{MOH},i}$ (Equil. Liq.)	$y_{\text{MOH},i}$ (Equil. Vap.)	$P_{\text{vap},i}$ (bar)
80% MOH	x_1	y_1	P_1
50% MOH	x_2	y_2	P_2
20% MOH	x_3	y_3	P_3
...	...	...	...

$y_{\text{MOH},i}$ (x-axis).

3. Generating Data for an xy-Diagram (Relative Volatility Plot)

Plots vapor mole fraction (y_i) vs. liquid mole fraction (x_i) at constant T or P. Useful for separability.
Strategy:

- Use data from T-xy or P-xy generation.
- For each tie-line, you have $(x_{\text{MOH}}, y_{\text{MOH}})$.
- Plot y_{MOH} (y-axis) vs. x_{MOH} (x-axis).
- Often, a diagonal line ($y = x$) is drawn. Curve above diagonal $\Longrightarrow$ component is more volatile.

Visualizing Diagrams (e.g., Gnuplot)

Conceptual Gnuplot for T-xy (data in `vle_data.dat`: Temp x_{MOH} y_{MOH} P_{vap}):

```
set title "T-xy Diagram for Methanol-Propanol"
set xlabel "Mole Fraction Methanol (x_{MOH}, y_{MOH})"
set ylabel "Temperature (K)"
set xrange [0:1]
# set yrange [min_temp:max_temp] # Adjust to your temperature range
set grid
set key top right

plot 'vle_data.dat' using 2:1 with linespoints title 'Bubble Point (T vs x_{MOH
  })', \
     'vle_data.dat' using 3:1 with linespoints title 'Dew Point (T vs y_{MOH})'
```

Listing 45: Conceptual Gnuplot script for T-xy diagram

Key Considerations for Building Diagrams

- **Number of Points:** 5-7 points across the range is a good start.
- **Equilibration:** Ensure each VLE simulation is truly equilibrated (stable density profiles, compositions, pressure).
- **Computational Cost:** Each point requires a full, lengthy simulation.
- **Force Field Limitations:** Accuracy depends on force field quality (GAFF2 here). Compare with experimental data if available.

Your Next Steps (Practical)

1. Complete analysis for current 298.15 K simulation:
   - Determine Z-ranges for bulk liquid/vapor.
   - Extract average densities for `MOH` and `PRH`.
   - Calculate x_{MOH} and y_{MOH}.
   - Calculate P_{vap} from `system_vle.edr`.

   You have one VLE data point: ($T = 298.15$ K, $P_{\mathrm{vap}}, x_{\mathrm{MOH}}, y_{\mathrm{MOH}}$).
2. Plan Your Next Simulation:
   - If T-xy: Choose new temperature (e.g., 310 K or 320 K). Update `ref_t` in `nvt_equil.mdp`, `npt.mdp`, and `vle_nvt.mdp`.
   - If P-xy: Keep $T = 298.15$ K. Change overall composition in `packmol.inp` (molecule numbers).

This systematic approach builds VLE diagrams.

9 Troubleshooting & Common Issues

This section is invaluable for anyone (including your future self) who encounters problems. It should be a "first aid" guide. For each issue, we describe the symptom, likely cause(s), and solution(s)/debugging steps.

- **Issue: `antechamber` PDB Formatting Errors**
  - *Symptom:* `antechamber` exits with a fatal error related to PDB coordinate formatting.
  - *Likely Cause(s):* The input `.pdb` file for `antechamber` does not strictly adhere to PDB format, especially the fixed-column layout for atomic coordinates (X in 31-38, Y in 39-46, Z in 47-54, often F8.3). Spaces and justification matter.
  - *Solution(s) / Debugging Steps:*
    - Manually inspect the input `.pdb` file with a text editor, paying close attention to column alignment for all `ATOM` or `HETATM` records.
    - Use a text editor with a column mode or a ruler guide if possible.
    - Ensure atom serial numbers, atom names, residue names, and residue sequence numbers are also correctly formatted and justified as per PDB spec, as misalignment here can shift coordinates.
    - If the PDB was generated by a program, check its export options or try re-generating it. Using a reliable molecular editor (like Avogadro) to save the PDB often ensures correct formatting.
- **Issue: `antechamber` Internal Program Errors (`bondtype`, `atomtype`)**
  - *Symptom:* `antechamber` reports a fatal error from an internal program like `bondtype` or `atomtype`. Subsequently, `parmchk2` (if run in the same script line) fails because the `.mol2` file was never created by `antechamber`.
  - *Likely Cause(s):* Often due to problematic input geometry in the PDB file (e.g., atoms too close, incorrect valencies implied by the structure) that `bondtype` cannot resolve.
  - *Solution(s) / Debugging Steps:*
    - Carefully review the input `.pdb` file in a molecular visualizer.
    - Perform a quick geometry optimization/cleanup in a tool like Avogadro before feeding the PDB to `antechamber`.
    - Ensure all hydrogens are present and the molecule is chemically sensible.
- **Issue: `packmol` MOL2 Input Problems**

  - *Symptom:* `packmol` errors out when trying to read an input `.mol2` file, even if `filetype mol2` is specified in `packmol.inp`.
  - *Likely Cause(s):* This specific issue was encountered with Packmol v20.15.1. It seemed to ignore or misinterpret the `filetype mol2` directive and defaulted to trying to read the file as a PDB, which fails due to format differences.
  - *Solution(s) / Debugging Steps (Workaround Used):*
    * Ensure initial PDB files (fed to `antechamber`) have globally unique atom names for each molecule type (e.g., `C1M, H1M` for methanol; `C1P, H2P` for propanol).
    * In `packmol.inp`, set `filetype pdb`.
    * Provide these uniquely named PDB files as input to `packmol` in the `structure` blocks.
    * `antechamber` must have been run on these same uniquely named PDBs to generate the `.mol2` files that `tleap` will use, ensuring atom name consistency.

    Alternative (if available): Try a different version or build of `packmol` that correctly handles `.mol2` inputs.

- **Issue: `tleap` Parameter Assignment/Residue Recognition Errors**
  - *Symptom:* `tleap` cannot assign parameters because it doesn't recognize an atom, its type, or the residue it belongs to. Or, it reports splitting a known residue into an unknown part (e.g., "POL and OL").
  - *Likely Cause(s) & Solution(s):*
    * **Residue Name Mismatch:** The residue name in `mixture_packed.pdb` (from `packmol`) does not exactly match the name of a unit defined in `tleap` via `RES = loadmol2 res.mol2`. Ensure consistency.
    * **Atom Name Mismatch:** An atom name in `mixture_packed.pdb` (within a specific residue) does not exactly match an atom name within the corresponding `.mol2` file loaded by `tleap`. This was why our unique atom naming strategy was important for the Packmol PDB workaround.
    * **Problematic Residue Name (Name Conflicts/Splitting):** As encountered with `POL`, some 3-letter residue names might conflict with internal `tleap` libraries or parsing rules.
      · *Solution:* Choose a more unique 3-letter residue name (e.g., we changed `POL` to `PRH`). Ensure this new name is used consistently: in the input PDB for `antechamber`, with the `-rn` flag in `antechamber`, as the residue name within the `.mol2` file, as the residue name in the PDB for `packmol`, and when defining the unit in `tleap` (e.g., `PRH = loadmol2 pol.mol2` where `pol.mol2` internally defines `PRH`).
    * **Incorrect `.mol2` File:** The `.mol2` file loaded by `tleap` might be corrupted, not contain proper GAFF2 atom types, or lack charges. Ensure `antechamber` ran correctly.
    * **Missing `.frcmod` File:** If a `.frcmod` file was generated but not loaded in `tleap`, parameters might be missing.
    * **`leaprc.gaff2` not sourced:** If the base force field isn't loaded, no types or parameters will be available.

- **Issue: InterMol/ParmEd GROMACS Topology Issues**
  - *Symptom:* `gmx grompp` fails with "No molecules were defined in the system." Inspection of the `.top` file shows a single large `[ moleculetype ]` or missing `[ molecules ]` section.
  - *Likely Cause(s):* The conversion tool did not correctly interpret the multiple residue types from the Amber `prmtop` as separate GROMACS `moleculetypes`.
  - *Solution(s) (InterMol CLI worked for us):*
    * Use `python -m intermol.convert --amb_in mixture.prmtop mixture.inpcrd --gromacs --oname mixture_intermol`. This invocation correctly generated separate `[ moleculetype ]` sections for `MOH` and `PRH` and the final `[ system ]` and `[ molecules ]` sections.
    * If using ParmEd's Python API or other InterMol API calls, ensure the methods used are appropriate for distinguishing multiple residue types into separate `moleculetypes`. This can be version-dependent.

- **Issue: GROMACS `mdrun` Crashes Immediately (LINCS Warnings)**
  - *Symptom:* Simulation blows up immediately. LINCS warnings show huge constraint deviations.
  - *Likely Cause(s):* Severe steric clashes or strained geometry in the starting structure, leading to enormous forces.
  - *Solution(s):* Perform Energy Minimization. Always run energy minimization (`gmx grompp -f minim.mdp ...` then `gmx mdrun -deffnm system_em`) on the initial structure from `packmol`/conversion before starting any dynamics (NVT, NPT).
- **Issue: GROMACS NPT Simulation Instability (Barostat Choice)**
  - *Symptom:* NPT simulation is unstable, box volume oscillates wildly, then LINCS failures and crash.
  - *Likely Cause(s):* The system is not yet sufficiently equilibrated (density is far from target), and an "aggressive" barostat like Parrinello-Rahman is used too early.
  - *Solution(s):* For initial NPT equilibration, use a more stable, "gentler" barostat like Berendsen (`pcoupl = Berendsen` in `npt.mdp`). Run with Berendsen until the density stabilizes, then optionally switch to Parrinello-Rahman for longer NPT runs if precise NPT ensemble statistics are needed from that stage.
- **Issue: Gnuplot (or other graphical tools) Display Errors on Headless Server**
  - *Symptom:* Plotting commands fail with "could not connect to display" or "cannot open display".
  - *Likely Cause(s):* The program is trying to open a graphical window, but the SSH session to the headless server does not have X11 display forwarding enabled or working.
  - *Solution(s):*
    - **Output to File:** Instruct Gnuplot to render to an image file:

```
set terminal pngcairo size 800,600
set output 'my_plot.png'
plot 'data.xvg' using 1:2 with lines
unset output # Important to close the file
```

Listing 46: Gnuplot output to file

      Then download the `.png` file using `scp` or `sftp`.
    - **Enable X11 Forwarding (More complex):** Configure X11 forwarding on the server (`/etc/ssh/sshd_config`) and use `ssh -X` or `ssh -Y` when connecting from a local Linux/macOS machine (which must have an X server running). For Windows clients, use an X server like Xming or VcXsrv and configure your SSH client (e.g., PuTTY, MobaXterm) for X11 forwarding.
- **Issue: `scp` / `sftp` File Transfer Failures**
  - *Symptom:* Cannot connect to the server from the local machine to transfer files.
  - *Likely Cause(s) & Solution(s):*
    - **Incorrect IP Address or Hostname:** Verify the server's IP/hostname.
    - **Server Firewall:** Ensure the server's firewall (e.g., `ufw` on Ubuntu) allows incoming connections on port 22 (SSH). Command: `sudo ufw allow 22/tcp`.
    - **Local Firewall:** Ensure your local machine's firewall isn't blocking outgoing SSH/SCP connections (less common).
    - **SSH Daemon Not Running on Server:** Check `sudo systemctl status ssh` on the server.
    - **Network Issues:** Problems with your local network, router, or internet connection.

* **Using Cloudflare Tunnel (Specific to your case):** If accessing the server via a Cloudflare Tunnel, `scp` and `sftp` from your local machine must use the SSH host alias defined in your local SSH config file (e.g., `scp toricalc:/path/on/server /local/path`), which uses the `ProxyCommand` to route through `cloudflared.exe`. Direct connection to the server's private IP will fail if you are not on the same local network.

10 Conclusion

This guide has detailed a comprehensive workflow for simulating the Vapor-Liquid Equilibrium (VLE) of a binary mixture, using methanol and 1-propanol as a practical example. By systematically employing tools from the AmberTools suite (namely `antechamber`, `parmchk2`, and `tleap`) for molecular parameterization with the GAFF2 force field, and `packmol` for initial system assembly, we prepared a system suitable for molecular dynamics. A crucial conversion step to GROMACS format was robustly handled using InterMol, overcoming common pitfalls associated with multi-component topology generation.

The subsequent GROMACS simulation protocol involved essential equilibration steps: energy minimization to relax initial strains, NVT equilibration to thermalize the system at the target temperature, and NPT equilibration to achieve the correct liquid density under target pressure. The choice of the Berendsen barostat for initial NPT proved vital for stability. Following equilibration, a slab geometry was created using `gmx editconf`, setting the stage for the NVT VLE production run. Analysis of the VLE trajectory using `gmx density` allowed for the determination of component density profiles, from which liquid and vapor phase compositions (x_i, y_i) can be derived. The equilibrium vapor pressure (P_{vap}) can also be extracted using `gmx energy`.

The successful execution of this multi-stage process yields a single VLE tie-line ($T, P_{\text{vap}}, x_i, y_i$). By systematically repeating this workflow at various temperatures (or overall compositions), users can generate the necessary data to construct complete VLE phase diagrams (e.g., T-xy, P-xy).

While this workflow is powerful, users should be mindful of the computational expense, especially for long VLE production runs and for generating multiple data points. The accuracy of the results is inherently tied to the quality of the chosen force field (GAFF2). Comparison with experimental data, where available, is always recommended for validation. The troubleshooting section provides guidance for common issues encountered during such complex simulation setups.

Ultimately, this documented methodology provides a robust pathway for computational chemists and chemical engineers to investigate and predict VLE behavior of organic mixtures, contributing valuable data for process design and fundamental thermodynamic understanding.

A Appendix: Full Scripts and Input Files

All scripts and key input files discussed in this guide are available on GitHub. Please refer to the repository for the complete files:

`https://github.com/Thyco5/gromacs_vle`

For convenience, the content of the main scripts and input files generated throughout this workflow are also listed below by reference to their definitions in the main text.

Example Methanol PDB (`methanol_unique.pdb`)

The content for this file is shown in Listing 1.

Example Antechamber Command (Methanol)

The command used is shown in Listing 3.

Example Parmchk2 Command (Methanol)

The command used is shown in Listing 5.

Example Packmol Input File (`packmol.inp`)

The content for this file is shown in Listing 6.

Example `tleap` Input File (`tleap_mixture.in`)

The content for this file is shown in Listing 9.

Example InterMol Python Script (`convert_with_intermol.py`)

The content for this file is shown in Listing 12. Alternatively, the command line invocation is shown in Listing 13.

Example Energy Minimization MDP File (`minim.mdp`)

The content for this file is shown in Listing 19.

Example NVT Equilibration MDP File (`nvt_equil.mdp`)

The content for this file is shown in Listing 23.

Example NPT Equilibration MDP File (`npt.mdp`)

The content for this file is shown in Listing 26.

Example VLE Production MDP File (`vle_nvt.mdp`)

The content for this file is shown in Listing 35.

Example `gmx editconf` Command for Slab Creation

The command used is shown in Listing 32.

Example `gmx density` Commands

Commands are shown in Listings 38, 39, and 40.

Example Gnuplot Script for Density Profiles

An example script is shown in Listing 41.

Example Python Snippet for Average Density Calculation

A conceptual script is shown in Listing 42.

Example Python Snippet for Average Pressure Calculation

A conceptual script is shown in Listing 44.

Example Gnuplot Script for T-xy Diagram

A conceptual script is shown in Listing 45.