The following text contains practical step-by-step user guides for how the functions contained in `RPA_polyampholytes.py` and `FTS_polyampholytes.py` can be used to calculate phase diagrams in RPA and FTS for the polyampholyte solution model. These codes are published along with [1], and follows the methods and notations described therein.

Random phase approximation (RPA)

When executing `RPA_polyampholytes.py`, the script produces (1) a plot depicting the self-intersection behaviour of the $(\beta\mu(\rho), \beta\Pi(\rho))$ curve at an example value of l_B, and (2) the full phase diagram (i.e. the binodal curve) in the (ρ, T^*)-plane. The `main` function of this script is structured as follows:

1. A `PolySol_RPA` object is initialized with values of the excluded volume parameter ($v \to$ `v`), Bjerrum length ($l_\mathrm{B} \to$ `lB`), smearing length ($\bar{a} \to$ `a`) and charge sequence ($\sigma_\alpha \to$ `seq`) specified by the user. The charge sequence is pulled from the `CL_list.py` script. The `PolySol_RPA` class is defined at the top of the script, along with its associated functions.

2. The function `PolySol_RPA.calc_mu_Pi` is then used to calculate the chemical potential and osmotic pressure for a set of polymer bead densities, contained in the array `rho`. The chemical potential is then plotted against the osmotic pressure. It is useful to look at this plot before computing the actual phase diagram to get a rough idea of what the critical temperature is, or what the densities of the dense and dilute bulk phases are for certain input temperatures.

3. For the subsequent phase diagram calculation, the user first specifies the values of $T^* = 1/l_\mathrm{B}$ for which the binodal curve is to be calculated. The T^* values are contained in the array `T_all`. The maximum T^* should be higher than (or equal to) the critical temperature T^*_c to assure that the low- and high density branches of the binodal curve merge in the final phase diagram.

4. We now loop through each temperature in `T_all` starting from the lowest temperature. The temperature is set using the `PolySol_RPA.setl_v_lB` function. At each temperature, the we find the densities of the low- and high density phases using the function `PolySol_RPA.find_binodal`. This function takes two arguments, `rho_L_in` and `rho_H_in`, that represent initial guesses for the respective bulk densities which are used by `scipy.optimize.fsolve` to numerically solve the coexistence conditions $\mu(\rho_\mathrm{L}) = \mu(\rho_\mathrm{H})$, $\Pi(\rho_\mathrm{L}) = \Pi(\rho_\mathrm{H})$. It is important that `rho_L_in` (`rho_H_in`) is set sufficiently low (high) for the first temperature, in order to avoid the trivial solution $\rho_\mathrm{L} = \rho_\mathrm{H}$. At subsequent temperatures, the initial guesses are taken to be the bulk densities found at the previous temperature, which ensures a fast convergence of `scipy.optimize.fsolve` if the temperature values in `T_all` are sufficiently dense.

Field theory simulation (FTS)

The accompanying FTS code `FTS_polyampholytes.py` contains the definition of a `PolySol` object, representing a field picture description of a polyampholyte solution, which has methods necessary for computation of chemical potential μ and osmotic pressure Π through Complex-Langevin (CL) evolution. The `main` function of this script, which is run upon the execution of the script, contains a usage example for generating the CL time trajectories $\mu(t)$ and $\Pi(t)$, whose asymptotic averages correspond to thermal averages. A `PolySol` object is initialized for given charge sequence ($\sigma_\alpha \to$ `sig`), bead bulk density ($\rho \to$ `rho`), Bjerrum length ($l_\mathrm{B} \to$ `lB`), excluded volume parameter ($v \to$ `v`), smearing length ($\bar{a} \to$ `a`) and lattice dimension ($M \to$ `Nx`) specified by the user. The bead- and charge density conjugate fields $w(\mathbf{r})$ and $\psi(\mathbf{r})$ (represented as `PolySol.w` and `PolySol.psi`) are

the initialized as random fluctuations around $\psi = 0$ and $w = \mathrm{i}v\rho$. The fields are then evolved in CL time using the function `CL_step_SI`, which performs a single time-step using the semi-implicit integration scheme described above. At every 50th step, the chemical potential and osmotic pressure are calculated using `PolySol.get_chem_pot()` and `PolySol.get_pressure()`, respectively, which are then written to a trajectory file (with file name ending with "_traj.txt").

Our suggested method for computing the phase diagram in FTS follows closely that of the above described RPA method. That is, we use FTS to compute $\mu(\rho)$ and $\Pi(\rho)$ at many different values of the bulk density ρ, and use interpolated versions of $\mu(\rho)$ and $\Pi(\rho)$ to numerically find the bulk densities for which the curve $(\mu(\rho), \Pi(\rho))$ self-intersects. This method is well-suited for computation on a cluster since the calculations at each ρ value can be performed in parallel by assigning each run to a separate core. An alternative, highly efficient Gibbs ensemble method exists in the literature [2], and we encourage the reader to also explore this method.

The Python script `FTS_trajectories_MPI.py` may be used to calculate a phase diagram in a cluster setting where many cores are available. The script imports `FTS_polyampholytes.py` and calculates the CL trajectories of μ and Π at several densities, each trajectory assigned to a separate CPU utilising the Python `multiprocessing` module. The script `FTS_trajectories_MPI.py` works as follows:

1. In the `main` function, the number of CPUs to be used (which equals the number of densities considered) is specified by the parameter `ncpus`. To fully utilise the cluster resources available, we recommend setting `ncpus` equal to the number of CPUs available on a node.

2. The densities are contained in the array `all_rho`. We recommend densities evenly spaced on a logarithmic scale. It is important that the density interval covered by the entries of `all_rho` is wide enough to cover the densities of the coexisting high- and low density bulk phases.

3. The Bjerrum lengths are specified in the array `all_lB`. The script `FTS_trajectories_MPI.py` is executed with an integer parameter specifying the entry of `all_lB` to be taken as the l_B value. For example, running the script through the command line as

   ```
   python FTS_trajectories_MPI.py 0
   ```

   indicates that the first entry of `all_lB` is taken to be the value of the Bjerrum length.

4. The script then proceeds to generate a CL time evolution trajectory $\{\mu(t), \Pi(t)\}$ for each density in `all_rho`, similarly to the usage example contained in the `main` function of `FTS_polyampholytes.py`. The trajectory files are saved in the subdirectory `data/` and are named according to Bjerrum length and bead density.

One instance of `FTS_trajectories_MPI.py` needs to be executed for each desired l_B. The resulting trajectory files can then be analysed using the script `FTS_analyze_trajectories.py` in order to yield the thermally averaged chemical potentials and osmotic pressures, and the resulting phase diagram. This script needs to be run from the same folder as `FTS_trajectories_MPI.py` and is used as follows:

1. To monitor the calculation of the self-intersection points of the $(\mu(\rho), \Pi(\rho))$ curve, set `show_plot = True` at the top the script. This will produce one plot per temperature displaying the $(\mu(\rho), \Pi(\rho))$ curve along with all found self-intersection points.

2. Make sure that the variables `ncpus`, `all_rho` and `all_lB` are set to the same values as in `FTS_trajectories_MPI.py`.

3. The variable `t_equil` defines defines length of the equilibration CL time period to be discarded from the thermal averaging. We recommend plotting a few of the trajectory files and estimate how long `t_equil` is by observing at what t the curves $\mu(t)$ and $\Pi(t)$ start to flatten out.

4. The script then reads in each trajectory file, calculates the thermal averages of μ and Π at each density, and outputs the result to files ending with "_mu_Pi.txt". Figure 5 in [1] shows the data from one of these files.

5. At each l_B, the program then proceeds to find the densities of of the coexisting phases by calling the function `find_intersection(rho, mu, Pi)`, where `mu` and `Pi` are arrays containing the FTS computed chemical potentials and osmotic pressures, respectively, at the densities `rho`. This function works as follows:

   - The data is first slightly smoothed using a Savitzky-Golay filter (through `scipy.signal.savgol_filter`), which makes the calculation more robust against fluctuations (due to finite sampling) in the vicinity of the self-intersection point.
   - Neighboring values in `mu` and `Pi` are used to construct `sympy.geometry.Segment` objects in the (μ, Π) plane. For each pair of segments, the `sympy.geometry.intersection` function is used to check if they intersect.
   - Each intersection found is then used as an initial guess for `scipy.optimize.fsolve` to more accurately find the densities corresponding to the self-intersection point. In this step, interpolated versions of `mu` and `Pi`, as functions of `rho`, are constructed using `scipy.interpolate`.
   - In FTS, it often happens that several self-intersections at the same temperature are found, out of which only one corresponds to the physical phase separation point. The other self-intersection points tend to be located around the spinodal turning points inside the (μ, Π) loop, and will disappear for larger systems. The function `find_intersection` therefore selects the solution to $\mu(\rho_L) = \mu(\rho_H)$, $\Pi(\rho_L) = \Pi(\rho_H)$ for which $\rho_H - \rho_L$ is the largest. We nevertheless recommend plotting all found intersection points to ensure that the correct one is selected.

6. The Bjerrum length, the two bulk densities and the coexistence chemical potential and osmotic pressure are written to the file "data/...phase_diagram.txt". This data can be used to plot phase diagrams such as Fig. 6 and Fig. 7 in [1].

References

[1] Yi-Hsuan Lin, Jonas Wessén, Tanmoy Pal, Suman Das, and Hue Sun Chan. (in preparation).

[2] Robert A. Riggleman and Glenn H. Fredrickson. Field-theoretic simulations in the gibbs ensemble. *The Journal of Chemical Physics*, 132(2):024104, 2010.