Hyperdynamics Lab Exercises

The goal of this lab is to gain some familiarity with basic aspects of hyperdynamics, both how it is implemented and how it performs for various problems and bias potentials. For clarity, only simple model potentials and crude bias potentials are currently incorporated.

First create a directory and `cd` to it. Then `cp ~germann/public/* .` to copy three Fortran files and three input files. The code consists of these Fortran files:

- `hypernd.f` –main driver, handling I/O and molecular dynamics loop
- `hyperndpots.f` –potential-specific subroutines, both the n-dimensional model potential energy hypersurfaces and the bias potential functions
- `hyperndsubs.f` –less interesting subroutines, primarily random number generation

Compile these with `f77 -O *.f -o hypernd` and run by redirecting a file to standard input: `./hypernd < 1.in`

The input files should be relatively self-explanatory; at present three different potentials are implemented, so `icase` is 1, 2, or 3. If you look at `hypernd.f` you will see that there is an inner loop of `nstep` MD iterations, after which the kinetic energy is monitored and information about the current configuration is printed to `fort.9` if `iprint` is 1 or greater. This outer loop is executed `nouter` times, so the total number of timesteps is given by `nstep` $\times$ `nouter`. We next specify a Langevin friction coefficient β, which for no particular reason is called α in JCP Eqs. (21) and (22), and ξ by Allen and Tildesley, *Computer Simulation of Liquids*. The temperature k_BT and two bias potential parameters are specified next, z_{max} and a for JCP Eq. (31), or a negative value and v_{flat} for a flat bias potential. Finally, `iprint` = 0 generates no output other than that to stdout, 1 dumps state information to `fort.11` and periodic trajectory points to `fort.9`, while a value of 2 in addition prints a subset of trajectory information *every timestep* to `fort.13`.

The first n-dimensional model potential has the form

$$V(\mathbf{x}) = \cos(2\pi x_1)(1 + \sum_{i=2}^{n} d_1 x_i) + \frac{d_2}{2}(2\pi)^2 \sum_{i=2}^{n} x_i^2 + d_3 \cos(2\pi x_1/d_4) + d_5 x_1 \qquad (1)$$

with 5 d_i parameters. With $n = 2$ and $d_5 = 0$, this reduces to the two-dimensional model potential Eq. (20) of the background article (*J. Chem. Phys.* **106**, *4665 (1997)*, hereafter "JCP"), except for a typo in that equation: the factor of 2π in the y^2 term should really be $(2\pi)^2$. The initial exercises below will focus on the two d_i parameter sets used in that paper (Table I; models I and II correspond to `icase` = 1 or 2 in the input file). After reproducing some of the results quoted there, you should try varying some parameters on your own (model and bias potential parameters, temperature, ...) to see whether the effects are what you expect.

You then will carry out some runs with $n > 2$ to see the effect of additional "y-like" degrees of freedom on the computational boost obtained using the flat bias potential. To simplify this comparison, a n-dependent constant $-1 - 2(n-1)/\pi^2$ is added to the potential (1) to shift the minima

(using model I d_i parameters) to zero potential energy. (The saddle point energy is +2 after this shift.) A more general such shift is left as an optional exercise, as is the modification of the simple 2x2 Hessian diagonalization used to implement the analytic bias potential of JCP Eq. (31).

A nonzero d_5 parameter may be used to bias the diffusion towards increasing or decreasing x_1, depending on the sign of d_5. You should experiment with this on your own, although parts of the code will need to be modified (specifically, the simple state determination used to count transitions). The analytic bias function JCP Eq. (31) may be used, but the flat bias is clearly inappropriate (*how could you modify it?*).

Before doing any runs it will be useful to have a graphics plotting program such as `gnuplot` (in a separate xterm window) or `xmgr` always running for simultaneous data visualization. For instance, gnuplot will be assumed here, so start it by:

```
% gnuplot
gnuplot> set data style l
```

1. Run input file 1.in. This will run ordinary molecular dynamics (i.e., zero bias potential everywhere) on model potential I at $k_BT = 0.20$ (the first entries in JCP Tables II and III). It should finish in well under a minute on a relatively unloaded machine. Note the crossing rate and errorbars, computed from the number of observed crossings during the given MD time. Take a look at the transition time distribution which occurs by
   `gnuplot> p 'fort.9' u 2:8`
   (for this case column 1 = MD time and 2 = hypertime are equivalent, but for the later examples column 2 should be used to compare with this unbiased dataset).

2. Run input file 2.in. This will run hyperdynamics using the JCP Eq. (31) bias potential. First plot the minimum-energy path on both the original and biased potential surfaces by
   `gnuplot> p 'fort.15','fort.15' u 1:3`
   and compare with JCP Fig. 3. Compare the boost and crossing rates with those quoted in the first entry of JCP Table III. Look at the advance in hypertime vs. MD time by
   `gnuplot> p 'fort.9' u 1:2`

3. As in JCP Table III, lower the temperature and run hyperdynamics to verify the dramatic increase in boost as temperature is lowered. Transitions may occur so rapidly that you will need to reduce the number of steps in the inner MD loop, `nstep`, to see them in the fort.11 output. (The counting of dividing-surface crossings in `hypernd.f` occurs in the innermost loop, so `nstep` only affects the fort.11 output frequency.)

4. Run input file 3.in. This is identical to the 2.in input (before you modified it in the previous exercise), except that a flat bias potential `vflat` = 1.6 is used instead of JCP Eq. (31). Check that the same crossing rate is obtained (within statistical errors). What boost is obtained? How did the execution time compared to 2.in? Vary `vflat` and k_BT to check that the behavior is what you expected.

5. Using a flat bias potential (try at least two different values, say `vflat` = 1.0 and 1.8), run model I with $n = 2$, 4, and 8 degrees of freedom. What happens to the resulting computational boost?