Heat Engine

Iraj, Michael,Watee
Department of Physics and Center for Soft Matter Research, New York University, 726 Broadway, New York, NY 10003
(Dated: January 17, 2019)

I. EQUATION

We want to simulate the Fokker-Planck equation describing two particles at temperatures $\bar{T} + \Delta/2, \quad \bar{T} - \Delta/2$, connected by a spring with strength k, on a sawtooth potential U, with period l and height u_0.

The equation is as follows:

$$\partial_t P = -\nabla\cdot(\vec{A}P)+\nabla\cdot\vec{\vec{B}}\nabla P = -\partial_i(A_i P)+B_{ij}\partial_i\partial_j P \quad (1)$$

Where the vector $\vec{A}$ and matrix $\vec{\vec{B}}$ are defined as:

$$\begin{aligned} A_r &= -2kr + U'(R+r/2) - U'(R-r/2) \\ A_R &= \frac{1}{2}(U'(R+r/2) + U'(R-r/2)) \\ B_{rr} &= \frac{8\bar{T}}{\gamma} \\ B_{rR} &= B_{Rr} = \frac{\Delta}{\gamma} \\ B_{rr} &= \frac{2\bar{T}}{\gamma} \end{aligned} \quad (2)$$

From here on out, we set $\gamma = 1$ because all it does is rescale the time for relaxation, and we are only interested in steady state anyway.

II. SPATIAL GRID

We want to set up a finite difference scheme for this PDE. First, we establish the spatial grid on which our probability distribution lives. The grid is periodic in R, and needs to be cut off somewhere in r. So we generate a Cartesian mesh in the (R, r) plane, and generate two arrays with respective shapes: $p(np, 2), g(np, 4)$. The former contains the coordinates of each point, and the latter lists the neighbours of each point (up, left, down, right), indicated by the row in which that point lives in p. For points that live on the edge of the space, we set the "empty point" to -1.

This way, the periodicity in R is encoded automatically, and the boundary conditions can be enforced on boundary points by simple pointing.

Finally, we need to choose a cutoff for r. There is a natural thermal length the spring takes, and that is $r_{spr} = \sqrt{\frac{2\bar{T}}{k}}$. So our simulation takes in a scalar parameter r_{mult} and we set $r_{max} = r_{mult} \times r_{spr}$. And then we let our grid in r go from $-r_{max}$ to r_{max}, and split into a sufficient number of points to attain the desired resolution δr.

III. TIME STEPPING

The current version of the code uses full implicity backwards-itme, centred-space time stepping [1].

$$\begin{aligned} &\frac{p^{n+1} - p^n}{\delta t} = f(p^{n+1}) \\ &f(p) = -\nabla\cdot(\vec{A}P) + \nabla\cdot\vec{\vec{B}}\nabla P \\ &\rightarrow (1 - \delta t L)p^{n+1} = p^n \\ &L = -\nabla\cdot\vec{A} - \vec{A}\cdot\nabla + B_{ij}\partial_i\partial_j \end{aligned} \quad (3)$$

The L operator is invertible and independent of P, so time evolution becomes computationally light after the initial building of L, since every time step is only a matrix-vector product. Moreover, the boundary conditions are simple to implement (described in the *boundary conditions* section).

The advantage of this method is that it is unconditionally stable, which is not the case for explicit methods. Attempts at evolving the equation explicitly led to instabilities due to the advective term, which could only be regulated by taking a prohibitively small timestep.

The main disadvantage of implicit methods is that they have poor time resolution. The current method is second order in space but only first order in time. In principle, this means that the time evolution will be inaccurate but since we are only interested in steady state this is okay.

This can be easily improved by changing the time evolution to a mixed implicit-explicit scheme a la Crank-Nicolson [1]. The idea there is as follows:

$$\begin{aligned} &\frac{p^{n+1} - p^n}{\delta t} = \frac{f(p^{n+1}) + f(p^n)}{2} \\ &\rightarrow (1 - \delta t L/2)p^{n+1} = (1 + \delta t L/2)p^n \\ &L = -\nabla\cdot\vec{A} - \vec{A}\cdot\nabla + B_{ij}\partial_i\partial_j \end{aligned} \quad (4)$$

Where we would keep the spatial derivatives centred, i.e. second order. Such a Crank-Nicolson scheme has

second order accuracy in time, which would improve our ability to maintain the normalisation of the probability distribution. The computational complexity of it is also small, since again L is independent of P.

IV. BOUNDARY CONDITIONS

Another advantage of implicit methods is they ease the process of enforcing boundary conditions. At the PDE level, we only have one boundary condition (the periodicity in R is trivially taken care of by our grid):

$$\begin{aligned} &J_r|_{r=\pm r_{max}} = 0 \\ &\rightarrow (A_r(\pm r_{max}) - B_{rr}\partial_r + B_{rR}\partial_R)P(\pm r_{max}) = 0 \end{aligned} \tag{5}$$

In general, linear implicit methods look like $\overleftrightarrow{T} \cdot \vec{P}^{n+1} = \vec{P}^n$. So we can implement our boundary conditions by identifying the row of P which corresponds to $r = \pm r_{max}$, and setting its right-hand-side to 0, and the left-hand-side to the current operator defined above, with centred derivatives. This should lead to $J_{bound} = 0 + \mathcal{O}(\delta r^2)$

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. (Cambridge University Press, New York, NY, USA, 2007).