Nose-Hoover Chain

1 Forces

1.1 Calculate Physical Forces

- Formula

  For a harmonic oscillator, the force exert on them is

  $$F = -m\omega^2 q$$

- Code

```
void physic_force(void)
{
    double omega = 1;
    for (int mi = 0; mi < sys.num_mole; mi++)
        for (int ai = 0; ai < sys.molecules[mi].num_atom; ai++)
            for (int di = 0; di < 3; di++)
                sys.molecules[mi].atoms[ai].F[di]
                = -1 * sys.molecules[mi].atoms[ai].m * pow(omega, 2)
                * sys.molecules[mi].atoms[ai].q[di];
}
```

1.2 Calculate Thermostat Forces

- Formula

$$\Gamma_1 = \sum_i^N \frac{\mathbf{p}_i^2}{m_i} - dNkT$$

$$\Gamma_j = \frac{\theta_{j-1}^2}{\mu_{j-1}} - kT \quad (j = 2, \ldots, M)$$

- Code

```
void thermo_force(void)
{
    double tmp_2_kinetic_energy = 0.0;

    for (int mi = 0; mi < sys.num_mole; mi++)
        for (int ai = 0; ai < sys.molecules[mi].num_atom; ai++)
            for (int di = 0; di < 3; di++)
                tmp_2_kinetic_energy +=
                pow(sys.molecules[mi].atoms[ai].p[di], 2.0)
                / sys.molecules[mi].atoms[ai].m;

    tmvs[0].Gamma = tmp_2_kinetic_energy - 3 * N
    * Phy_Const::Boltzmann_const * T;

    for (int j = 1; j < M; j++)
        tmvs[j].Gamma = pow(tmvs[j - 1].theta, 2.0) / tmvs[j - 1].mu
        - Phy_Const::Boltzmann_const * T;
```

```
}
```

2 Propagations

2.1 Physical Propagation

- Formula

$$p_i \to p_i + \frac{\Delta t}{2} F_i$$
$$q_i \to q_i + \Delta t \frac{p_i}{m_i}$$
$$p_i \to p_i + \frac{\Delta t}{2} F_i$$

- Code

```
void physic_propagate(void)
{
    physic_force();
    for (int mi = 0; mi < sys.num_mole; mi++)
        for (int ai = 0; ai < sys.molecules[mi].num_atom; ai++)
            for (int di = 0; di < 3; di++)
                sys.molecules[mi].atoms[ai].p[di]
                += bsp.step_size * sys.molecules[mi].atoms[ai].F[di] / 2;

    for (int mi = 0; mi < sys.num_mole; mi++)
        for (int ai = 0; ai < sys.molecules[mi].num_atom; ai++)
            for (int di = 0; di < 3; di++)
                sys.molecules[mi].atoms[ai].q[di]
                += bsp.step_size * sys.molecules[mi].atoms[ai].p[di]
                / sys.molecules[mi].atoms[ai].m;

    physic_force();
    for (int mi = 0; mi < sys.num_mole; mi++)
        for (int ai = 0; ai < sys.molecules[mi].num_atom; ai++)
            for (int di = 0; di < 3; di++)
                sys.molecules[mi].atoms[ai].p[di]
                += bsp.step_size * sys.molecules[mi].atoms[ai].F[di] / 2;
}
```

2.2 Thermostat Propagation

- Formula ($\delta_\alpha = w_\alpha \Delta t / n_{\text{ff}}$)

1.
$$\theta_M \to \theta_M + \frac{\delta_\alpha}{4} \Gamma_M$$

2. $j = M-1, \ldots, 1$
$$\theta_j \to \theta_j \exp\left(-\frac{\delta_\alpha}{8} \frac{\theta_{j+1}}{\mu_{j+1}}\right)$$
$$\theta_j \to \theta_j + \frac{\delta_\alpha}{4} \Gamma_j$$
$$\theta_j \to \theta_j \exp\left(-\frac{\delta_\alpha}{8} \frac{\theta_{j+1}}{\mu_{j+1}}\right)$$

3.

$$\eta_j \to \eta_j - \frac{\delta_\alpha}{2}\frac{\theta_j}{\mu_j}$$

4.

$$p_i \to p_i \exp\left(-\frac{\delta_\alpha}{2}\frac{\theta_1}{\mu_1}\right)$$

5. $j = 1, \ldots, M-1$

$$\theta_j \to \theta_j \exp\left(-\frac{\delta_\alpha}{8}\frac{\theta_{j+1}}{\mu_{j+1}}\right)$$
$$\theta_j \to \theta_j + \frac{\delta_\alpha}{4}\Gamma_j$$
$$\theta_j \to \theta_j \exp\left(-\frac{\delta_\alpha}{8}\frac{\theta_{j+1}}{\mu_{j+1}}\right)$$

6.

$$\theta_M \to \theta_M + \frac{\delta_\alpha}{4}\Gamma_M$$

- Code

```
void thermo_propagate(void)
{
    for (int wi = 0; wi < sy_info.n_sy; wi++) {
        for (int ni = 0; ni < sy_info.n_ff; ni++) {
            double tmp_delta = sy_info.weight[wi]
            * bsp.step_size / sy_info.n_ff;

            thermo_force();
            tmvs[M - 1].theta += tmp_delta * tmvs[M - 1].Gamma / 4;

            for (int j = M - 2; j >= 0; j--) {
                tmvs[j].theta *= exp(-1 * tmp_delta * tmvs[j + 1].theta
                / (8 * tmvs[j + 1].mu));
                tmvs[j].theta += tmp_delta * tmvs[j].Gamma / 4;
                tmvs[j].theta *= exp(-1 * tmp_delta * tmvs[j + 1].theta
                / (8 * tmvs[j + 1].mu));
            }
            for (int j = 0; j < M; j++)
                tmvs[j].eta -= tmp_delta * tmvs[j].theta / (2 * tmvs[j].mu);

            for (int mi = 0; mi < sys.num_mole; mi++)
                for (int ai = 0; ai < sys.molecules[mi].num_atom; ai++)
                    for (int di = 0; di < 3; di++)
                        sys.molecules[mi].atoms[ai].p[di]
                        *= exp(-1 * tmp_delta * tmvs[0].theta / (2 *
tmvs[0].mu));

            for (int j = 0; j < M - 1; j++) {
                tmvs[j].theta *= exp(-1 * tmp_delta * tmvs[j + 1].theta
                / (8 * tmvs[j + 1].mu));
                thermo_force();
                tmvs[j].theta += tmp_delta * tmvs[j].Gamma / 4;
                tmvs[j].theta *= exp(-1 * tmp_delta * tmvs[j + 1].theta
                / (8 * tmvs[j + 1].mu));
            }
            thermo_force();
```

```
                tmvs[M - 1].theta += tmp_delta * tmvs[M - 1].Gamma / 4;
            }
        }
    }
```

3 NHC Numerical Evolution

```
for (double t = 0; t <= bsp.run_time; t += bsp.step_size) {
    thermo_propagate();
    physic_propagate();
    thermo_propagate();
}
```