Note about Julia Code

Haolei Wang

January 3, 2019

1 Introduction

In this note, I will try to introduce how I write this Julia code to calculate the energy in a granular system as soon as possible. This code was based on Ortner's `'JuLIP'` package.

We will compare the performance of this code with my own Matlab code. It is known that Matlab deals the vector manipulation pretty well while loop rather bad. Julia can deal with loop rather well. In our project, it is rather common to use loop, so it is better to design a Julia code.

2 Problem Setting

In this section, we will introduce our problem. We need to calculate the inner potential energy of a granular system. The system is a square (2D) with periodic boundary condition, which is filled with two kind of disks. The number of particle of each are the same, while their radius ratio is 1.4. Particles will interact with each other only when they overlap. We give the following energy model.

$$E_{ij} = \begin{cases} k\dfrac{(r_i + r_j - |\vec{x}_i - \vec{x}_j|)^\alpha}{\alpha} & \text{for} \quad |\vec{x}_i - \vec{x}_j| < r_i + r_j, \\ 0 & \text{for} \quad |\vec{x}_i - \vec{x}_j| \geq r_i + r_j, , \end{cases} \quad i \neq j \tag{2.1}$$

where k represents the particle stiffness, r_i and r_j are the radii of particle i and j and $\vec{x}_i$ and $\vec{x}_j$ are the positions of the their centers. With different α, we can get different kinds of pair potentials. Thus the total potential energy of the system is:

$$E_{total} = \sum_{i<j} E_{ij}. \tag{2.2}$$

Since the particles only interact with each other unless they contact, we can rewrite the potential energy for a single contact c as:

$$E_c = k\frac{\delta_c^\alpha}{\alpha}, \tag{2.3}$$

where δ_c is the overlap associated with the contact c and according to equation (2.1):

$$\delta_c = r_i + r_j - |\vec{x}_i - \vec{x}_j|\,. \tag{2.4}$$

Thus, the total potential energy of the system is:

$$E_{total} = \sum_c E_c. \tag{2.5}$$

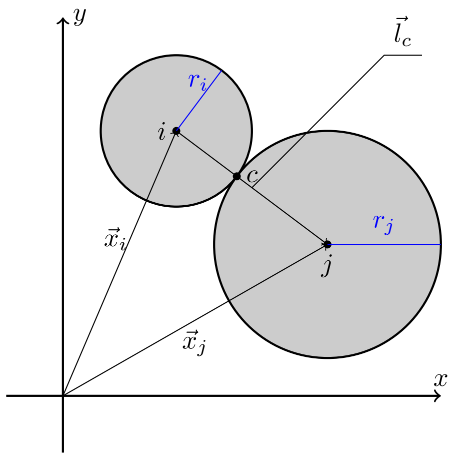

Figure 1: Two particles contact

3 Programming

In this section, we will introduce the programming approach.

3.1 Calculating the distance between two points in periodic cell

From section 2, we can calculate the total potential energy by summing the potential energy of all contacting particles. Thus, we need to find all the contacts. From equation (2.4), it's the key to calculate the distance of all particles with each other.

If the cell is not periodic, it's easy to calculate the distance between particles, which is the 2-norm of $|\vec{x}_j - \vec{x}_i|$. However, in our project, the cell is periodic, so we need to reconsider this when the particles are near the boundaries. Since we don't consider long-range interaction, so the distance between two points in periodic cell is the smallest of the distance of their periodic projections with each other.

Firstly, we will try to calculate the periodic distance in a unit square. In Figure 2, we have two points i and j, with their coordination X_i and X_j respectively. If we set

$$d\bar{X} = X_j - X_i = (d\bar{X}_1, d\bar{X}_2), \tag{3.1}$$

we know that $-1 < d\bar{X}_k < 1, k = 1, 2$. For a periodic unit square, it is important to note that the real difference of two points' coordination

$$dX = (dX_1, dX_2)$$

should satisfy the following condition:

$$|dX_k| \leq \frac{1}{2}.$$

The relationship between dX_k and $d\bar{X}_k$ is:

$$dX_k = \mathrm{mod}(d\bar{X}_k + \frac{1}{2}, 1) - \frac{1}{2}. \tag{3.2}$$

The distance between points i and j is:

$$d_{ij} = \sqrt{dX \cdot dX} \tag{3.3}$$

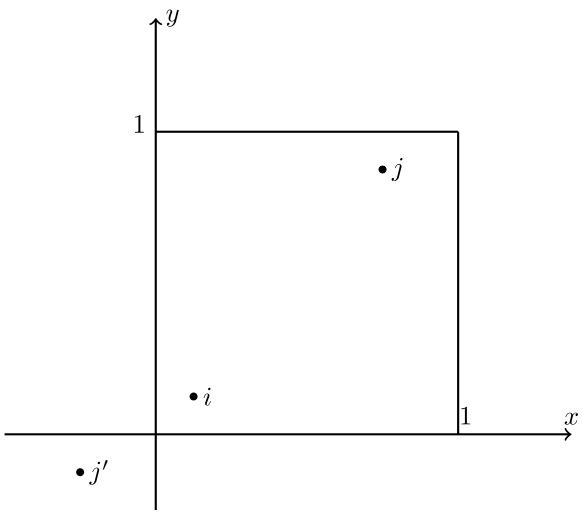

Figure 2: Points in periodic unit square

Secondly, we will consider a common deformed (simple shear) cell with periodic boundary condition. In Figure 3, the cell can be denoted using a matrix:

$$M = \begin{bmatrix} L & Le \\ 0 & H \end{bmatrix}$$

The points i and j in this cell can be projected in the unit square reference cell using

$$\hat{X} = M^{-1}x \tag{3.4a}$$

$$X = \hat{X} - \mathrm{floor}(\hat{X}) \tag{3.4b}$$

where X and x are the coordination in the reference cell and real cell respectively.

We can calculate the difference between two reference point, dX, using equation (3.1) and (3.2). Then we can calculate the real coordination difference using

$$dx = MdX \tag{3.5}$$

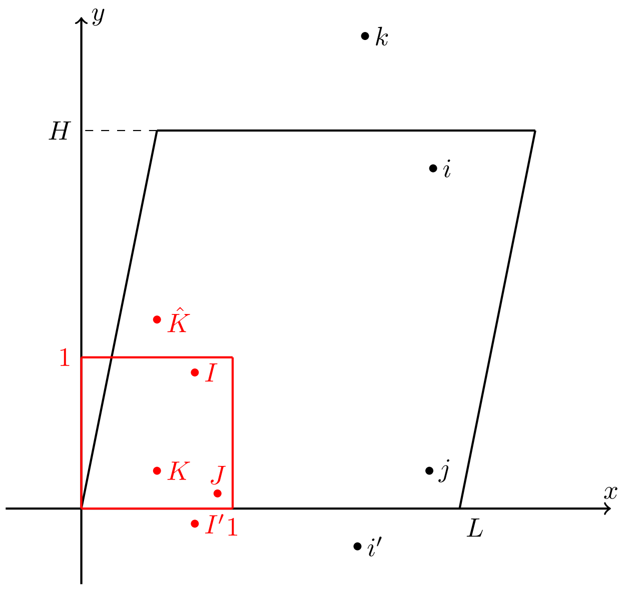

Figure 3: Points in deformed cell

The distance between points i and j can be calculated by

$$d_{ij} = \sqrt{dx \cdot dx} \tag{3.6}$$

3.2 Calculate neighbourlist of granular system

Now we know how to calculate the distance between points in periodic cell, it's identical to calculate the distance between the centers of two particles. Thus, we can determine whether two particles overlap.

Noting that the particles interact with each other only when they overlap, we can firstly find the neighbourlist of the system. The neighbourlist stores the pairs of indexes of every two possibly overlapping particles. Thus when we calculate the total potential energy, we only need to compute the potential energy in the neighbourlist and then sum up.

Ortner provide a package called '`NeighbourLists`' using Julia language, which is rather fast to determine the neighbourlist based on a cutoff r_{cut}. Matlab also provides several functions to calculate the distance and knn (k-nearest neighbour), the relative functions are **`pdist, knnsearch, rangesearch`**, etc. However, the Matlab functions cannot compute the periodic distance either, so the above functions are not suitable for our project. I programmed a Matlab code to calculate periodic distance, but it couldn't manipulate large system because the space complexity is $O(N^2)$. While Ortner's code is efficient and with $O(N)$ space complexity (not sure if this is correct), this can be a big advantage.

Package '`NeighbourLists`' computes the distance between particles and then determine whether the distance is smaller than r_{cut}. This package can provide the following information: `nlist.i, nlist.j, nlist.r, nlist.R`, here `nlist` means neighbourlist, `nlist.i` and `nlist.j` are two vectors of particles' index, `nlist.r` is a vector storing distance between particles, `nlist.R` is a vector that stores dx in equation (3.5). Meanwhile, `nlist.r[k]` is the distance between particle `nlist.i[k]` and `nlist.j[k]`, `nlist.R[k]` is dx associated with particle `nlist.i[k]` and `nlist.j[k]`.

Even though package '`NeighbourLists`' is very efficient, we still find it is a waste to calculate the total neighbourlist every time when we calculate the energy, since the displacement of particles in our system is so small that the neighbourlist is almost unchanged. We only need to update `nlist.r` and `nlist.R`. If the maximum displacement reaches a tolerance r_{tol}, we will update the total neighbourlist.

Next, we will give the pseudocode.

```
function energy(Config, V)
    """
    Function to calculate the total potential energy.
    V(r) is the pair potential.
    """
    nlist_i, nlist_j = update_neighbourlist(Config)
    E = 0.0
    for k = 1:length(nlist_i)
        i, j = nlist_i[k], nlist_j[k];
        xi, xj = Config.X[i], Config.X[j];
        dx = xj - xi;
        dX = inv_bins * dx;                   # using equation (3.4a)
        dX = dX - floor(dX);                  # using equation (3.4b)
        dX = mod(dx + 0.5, 1) - 0.5;          # using equation (3.2)
        dx = bins * dX;                       # using equation (3.5)
        r = norm2(dx);
        E = E + V(r);
    end
    return 0.5*E
end

function update_neighbourlist(Config)
   """
   Function to update the neighbourlist
   has_data(Config, fieldname) judges if the Config structure has this field.
   """
   if has_data(Config, nlist_X)
      X_Old = Config.nlist_X;
      max_displacement = max(norm2.(Config.X - X_Old));
      if max_displacement < r_tol
          return Config.nlist_i, Config.nlist_j;
      end
   end
   nlist = NeighbourList(Config, rcut);
   Config.nlist_X, Config.nlist_i, Config.nlist_j = Config.X, nlist.i, nlist.j;
   return nlist.i, nlist.j;
end
```

One should note that we compute dX using dx instead of X using x in the `energy` function. We will now show this.

In the pseudocode,

$$\begin{aligned}
&X_i = (\varepsilon_i, \eta_i)^T, \quad X_j = (\varepsilon_j, \eta_j)^T, \quad \text{where} \quad 0 \le \varepsilon, \eta < 1 \\
\Longrightarrow &\hat{X}_i = (m_i + \varepsilon_i, n_i + \eta_i)^T, \quad \hat{X}_j = (m_j + \varepsilon_j, n_j + \eta_j)^T, \quad \text{where} \quad m, n \in \mathbb{Z} \\
\Longrightarrow &x_i = M\hat{X}_i, \quad x_j = M\hat{X}_j \\
\Longrightarrow &dx = x_j - x_i = M(\hat{X}_j - \hat{X}_i) \\
\Longrightarrow &d\hat{X} = M^{-1}dx = \hat{X}_j - \hat{X}_i = (m_j - m_i + \varepsilon_j - \varepsilon_i, n_j - n_i + \eta_j - \eta_i)^T \\
\Longrightarrow &d\bar{X} = d\hat{X} - \text{floor}(d\hat{X}) = \boxed{\text{mod}(\varepsilon_j - \varepsilon_i, \eta_j - \eta_i, 1)} \\
\Longrightarrow &dX = \text{mod}(d\bar{X} + \frac{1}{2}, 1) - \frac{1}{2}.
\end{aligned}$$

While according to the introduction in section 3.1,

$$\begin{aligned}
&X_i = (\varepsilon_i, \eta_i)^T, \quad X_j = (\varepsilon_j, \eta_j)^T, \quad \text{where} \quad 0 \le \varepsilon, \eta < 1 \\
\Longrightarrow &d\bar{X} = X_j - X_i = \boxed{(\varepsilon_j - \varepsilon_i, \eta_j - \eta_i)^T} \\
\Longrightarrow &dX = \text{mod}(d\bar{X} + \frac{1}{2}, 1) - \frac{1}{2}.
\end{aligned}$$

The boxed part can reach the same dX which is needed.

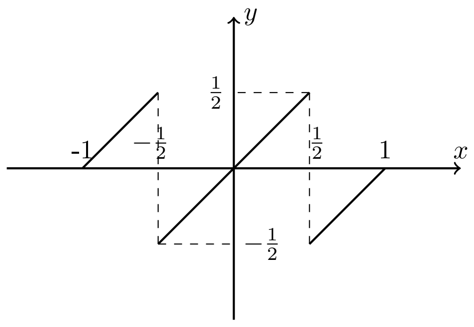

Figure 4: Function $y = \text{mod}(x + \frac{1}{2}, 1) - \frac{1}{2}$

In the real examples, we found that we didn't need to do the periodic correction for each pair of particles in the neighbourlists. In fact, most pairs of particles are very close originally, we can modify the `eneryg` function code.

```
function energy(Config, V)
    """
    Function to calculate the total potential energy.
    V(r) is the pair potential.
    """
    nlist_i, nlist_j = update_neighbourlist(Config)
    E = 0.0
    for k = 1:length(nlist_i)
        i, j = nlist_i[k], nlist_j[k];
        xi, xj = Config.X[i], Config.X[j];
        dx = xj - xi;
        r = norm2(dx);
        if r > 0.5*min(L,H)          # min(L,H) is the length scale of the system
           dX = inv_bins * dx;                  # using equation (3.4a)
           dX = dX - floor(dX);                 # using equation (3.4b)
           dX = dX - (dX > 0.5);                # using equation (3.2)
           dx = bins * dX;                      # using equation (3.5)
           r = norm2(dx);
        end
        E = E + V(r);
    end
    return 0.5*E;
end
```

This is much faster than the original code, and also faster than my own Matlab code. We compute the potential energy of one given granular system (with 1024 particles) for 1000 times using this modified code and the original code and my own Matlab code, the time costed are 0.32s, 2.14s and 0.87s respectively.