

Collisions and Interacting Particles

OBJECTIVES

In this activity you will use an iterative computer model to observe various types of collisions between a probe and a group of bound particles.

Using the VIDLE editor, open the shell file, "Model4_Shell.py" and save it on your computer as "Model4.py".

BACKGROUND

In this model we will imagine that a group of particles is bound together by some type of interaction that causes repulsion at close distances and attraction past some equilibrium distance. This could represent a cluster of interacting atoms, such as a plasma, or nucleons (protons and neutrons) in a nucleus. Throughout this model, we will specifically focus on the interatomic forces in a collection of atoms.

An example of a potential we know that gives rise to this behavior is the Morse potential, shown in the Figure 7.12. Near its minimum—the equilibrium point—the Morse potential has a similar shape to a parabola. Thus, we expect interactions to be springlike, unless an atom gains sufficient energy to escape from its neighbors. This possibility is realized when an atom has total energy $E > 0$, as is clear from the function's graph.

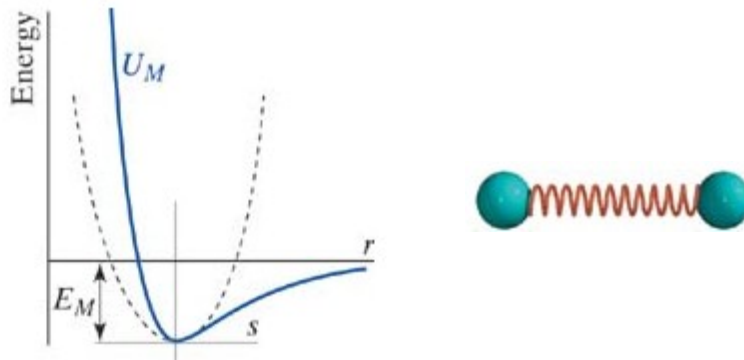


Figure 7.12 The bottom of the interatomic potential energy curve may be approximated by a parabola.

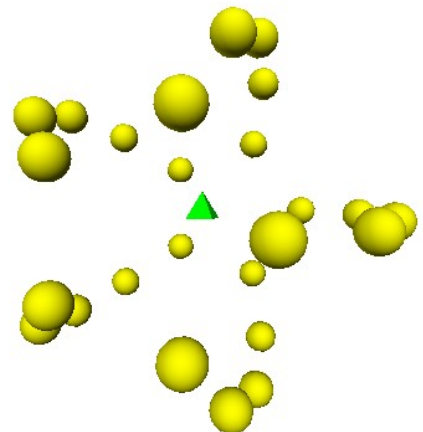
The Morse potential will determine interactions between atoms. In order to represent an incoming probe interacting with the cluster of atoms, we will adopt a flexible function that has very strong interactions when distance is small but whose strength falls off rapidly with distance.

THE PROGRAM

1. Setup

As usual, our program begins with a definition of constants. The constants relevant to this model detail the nature of atom-atom and probe-atom interactions. We will come back to these.

The Objects section initializes the probe and creates the atoms in a spherical lattice. The green pyramid in the center represents the *center of mass* of the cluster of atoms. Time values are initiated before the calculations begin.



The Calculations section begins with a series of outputs detailing the initial state of the system. A loop there sums the Morse potential between all atoms, their kinetic energies, and their momenta. Since the probe begins far away, it has only kinetic energy in the initial state (and also in the final state for the same reason.) These outputs to the shell, especially in combination with the final state outputs, allow you determine information regarding the types of collisions occurring.

2. Calculation loop

Let us first look at a pseudocode outline of the loop to understand what it is doing.

```
while t < 15:
    for i in range(0,nPart): ## choose ith atom for each iteration
        for j in range(0,nPart):
            calculate force due to all other atoms on ith atom
            calculate force due to probe on ith atom
            momentum principle for probe and ith atom
            position update for ith atom
        position update for probe
    calculate center of mass momentum
    position update for center of mass
```

While you might not be familiar with the concept of a loop within a loop, it is something you have probably done before. For example, consider the double sum

$$\sum_{i=1}^2 \sum_{j=1}^3 i+j = (1+1)+(1+2)+(1+3)+(2+1)+(2+2)+(2+3)=21$$

In Python, we could write this as

```
sum = 0
for i in range(1,3):
    for j in range(1,4):
        sum = sum + i + j
```

The value of `sum` would be equal to 21 after this code executes.

In our program, each time step sees an evaluation of the `i` loop and its nested `j` loop. Each iteration of the `i` loop calculates all of the forces on the `ith` atom, then updates the `ith` atom's momentum and position. To calculate the sum of the forces on the `ith` atom, our program uses the `j` loop to add all of the forces due to every other atom on the `ith` atom.

3. Morse Force

The Morse potential, representing a binding force between two atoms, can be written

$$U_M = E_M [1 - e^{-\alpha(r-r_{eq})}]^2 - E_M$$

Since our model is a dynamical simulation that uses forces to update momenta, we need to find the force associated with this potential. Recall that

$$\vec{F} = - \frac{dU}{dr} \hat{r}$$

Derive the Morse force due to the Morse potential. One of the five choices below is the correct answer. You may want to check with your instructor to make sure you have derived the correct result.

1. $\vec{F}_M = -2 E_M [1 - e^{-\alpha(r-r_{eq})}] \hat{r}$
2. $\vec{F}_M = -E_M [1 - e^{-\alpha(r-r_{eq})}]^2 \hat{r}$
3. $\vec{F}_M = -2 E_M [1 - e^{-\alpha(r-r_{eq})}] \alpha e^{-\alpha(r-r_{eq})} \hat{r}$
4. $\vec{F}_M = -E_M [1 - e^{-\alpha(r-r_{eq})}]^2 \alpha e^{-\alpha(r-r_{eq})} \hat{r}$
5. $\vec{F}_M = -2 E_M \alpha e^{-\alpha(r-r_{eq})} \hat{r}$

Carefully code the line defining `FMorse`. `EM`, `alpha`, `rmag`, `req`, and `rhat` have all been defined in the program. Note that e^r would be written in Python as `e**(rmag)`, for example. Here are the corresponding lines of code for the above expressions.

1.
`FMorse = -2*EM*(1-e**(-alpha*(rmag-req)))*rhat`
2.
`FMorse = -EM*(1-e**(-alpha*(rmag-req)))**2*rhat`
3.
`FMorse = -2*EM*(1-e**(-alpha*(rmag-req)))*alpha
 *e**(-alpha*(rmag-req))*rhat`
4.
`FMorse = -EM*(1-e**(-alpha*(rmag-req)))**2*alpha
 *e**(-alpha*(rmag-req))*rhat`
5.
`FMorse = -2*EM*alpha*e**(-alpha*(rmag-req))*rhat`

4. Probe Force

Aside from interatomic forces, we would also like to allow an external probe to interact with the system. Let us define this force as follows:

$$\vec{F}_{probe} = k \frac{e^{-r/r_{range}}}{r^2} \hat{r}$$

where k in our program is `probeStrength`, and r_{range} is `probeRange`. Note that the r in this formula (the distance between the probe and an atom) is not the same as the r in the Morse formulas (the distance between atoms). We define the present r and its associated unit vector in the program as `rPmag` and `rPhat`.

Code the line defining `Fprobe` using the above formula.

5. Center of Mass of the cluster

It will be instructive to see what happens to the center of mass of the cluster of atoms during its interaction with the probe. The momentum of the system is simply the sum of the momenta of all of its constituents,

$$\vec{p}_{sys} = \sum_{i=0}^{N-1} \vec{p}_i$$

which is calculated by the following Python loop:

```
for i in range(0,nPart):
    CoM.p = CoM.p + particles[i].p
```

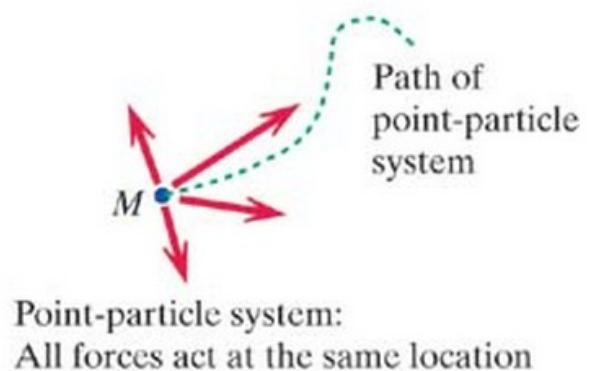
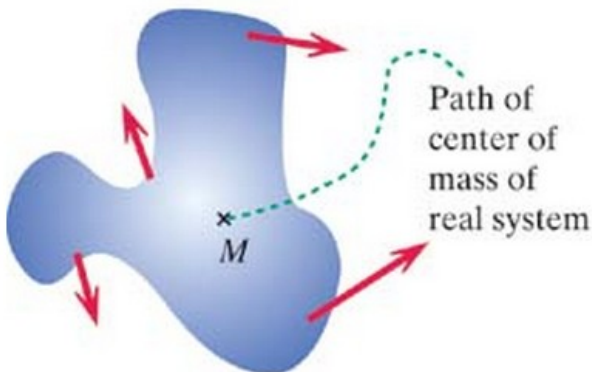
If the center of mass were an actual particle with momentum \vec{p}_{sys} and mass M_{total} , we could divide its momentum by its mass to obtain its velocity, \vec{v}_{CM} . Although the center of mass is not an actual particle, the point particle idealization of the real system does in fact behave this way.

$$\vec{p}_{sys} = M_{total} \vec{v}_{CM}$$

where M_{total} is the total mass of the constituent atoms.

Real system:

Forces act at different locations



Even though the solid objects we interact with on a daily basis are made up of a vast number of constituent atoms, for practical purposes we think of a baseball, for example, as being a single “particle” with a momentum \vec{p}_{sys} , mass M_{total} , and velocity equal to its center of mass velocity, \vec{v}_{CM} .

Write the line of code defining the center of mass velocity using the number of atoms, `nPart`, and the mass of each atom, `mPart`.

6. Reflection

At this point, you may begin answering the reflection questions for Model 4.

Remember that you only get one try per question on a reflection assignment, so be sure discuss with your group or an instructor before answering!