Pseudo-código para dinámica molecular

17 de Junio de 2021

Pseudo-código

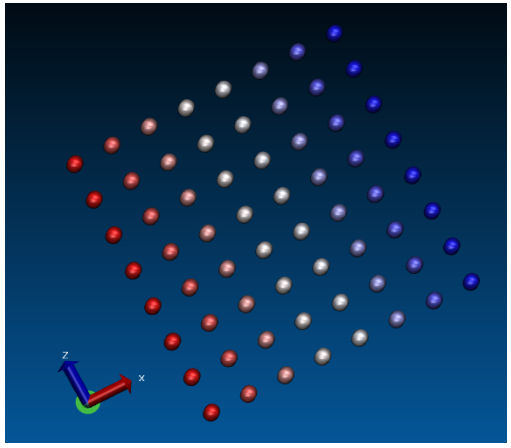
```
main()
{
    condiciones_iniciales();
    tablas();

    fuerzas_iniciales();

    for (nt = 0; nt < nt_maximo; nt++)
    {
        termalizacion();
        T_actual = calculo_temperatura();

        //mediciones();

        T_deseada = T_actual - T_max/(double)nt_maximo;

        reescaleo_velocidades();
    }
}
```

Loop de temperaturas

Más data acá: Understanding molecular simulations, Frenkel-Smit, capítulo 4.

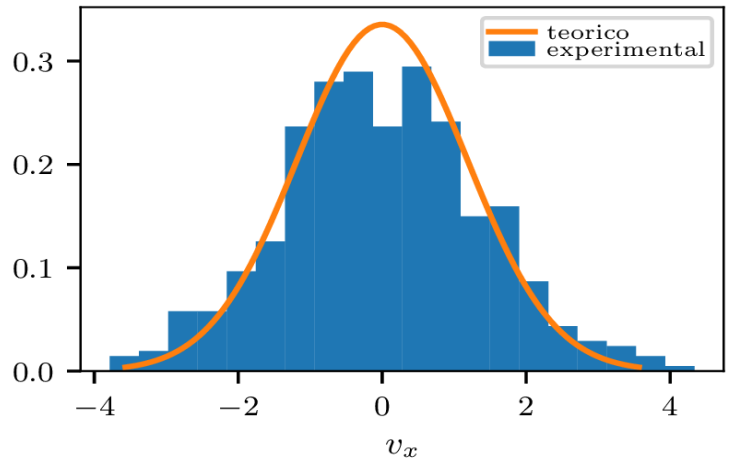
Condiciones iniciales

- Posiciones iniciales: arreglo cúbico simple (sc).



- Velocidades iniciales: distribución gaussiana centrada en cero y con dispersión √T

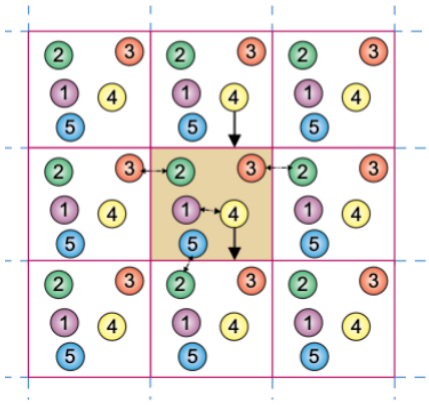
$$P(p) = \left(\frac{\beta}{2\pi m}\right)^{3/2} \exp\left[-\frac{1}{kT}\frac{p^2}{2m}\right]$$



Tablas

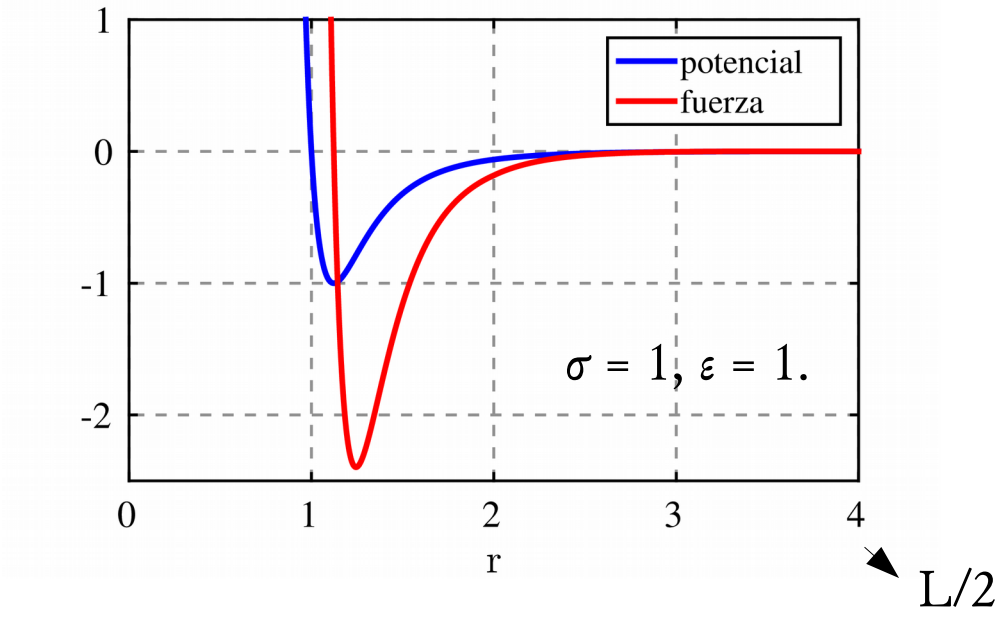
El potencial de Lennard-Jones es de rango infinito → lo truncamos.

$$u(r) = 4\epsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}]$$



Usamos un radio de corte de L/2. Es decir, a partir de dicho punto despreciamos la interacción entre las partículas (U'(r=L/2)=0).

Pero, U(r=L/2) no es estrictamente cero. Entonces, debemos hacer algo "suave" para evitar una divergencia en la fuerza (golpecitos).



Tablas: truncamiento

Dos posibles formas de hacer el truncamiento:

$$u(r) = 4\epsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}]$$

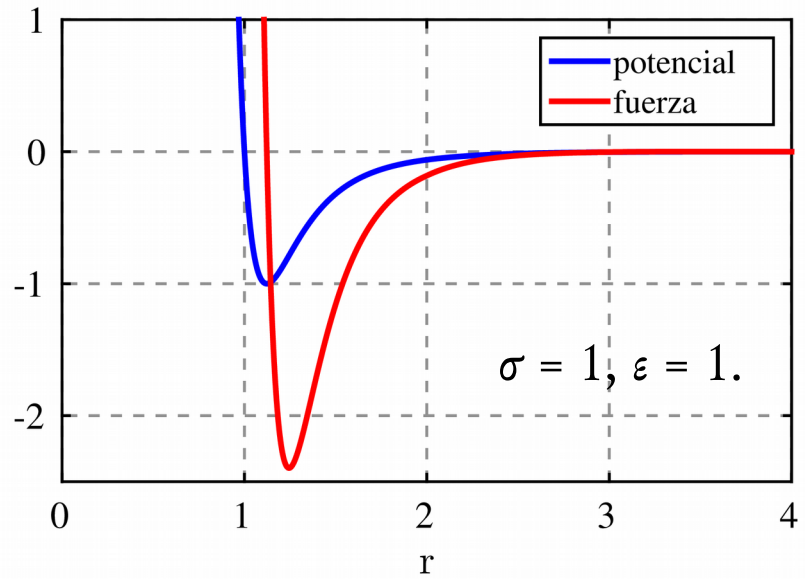
1) Desde r=L/4 hasta r=L/2 hacemos un "spline" tal que U'(r=L/4) = U(r=L/4) y U'(r=L/2) = 0.

2) "Shifteamos" el potencial

$$U'(r) = \begin{cases} U(r) - U(r_{corte}) & r \leq r_{corte} \\ 0 & r > r_{corte} \end{cases}$$

L/2 – 0.1

4000 valores⟶ dr ~ 0.001



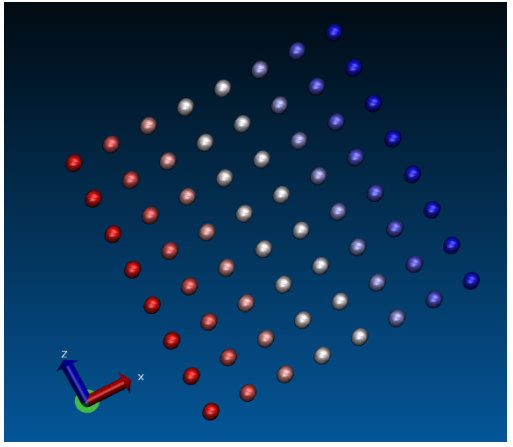
*Understanding molecular simulations, Frenkel-Smit, sección 3.2.

Termalización

Inicialmente nuestro sistema no está en equilibrio.

Hacemos evolucionar el sistema cierta cantidad de pasos (esto habrá que determinarlo).

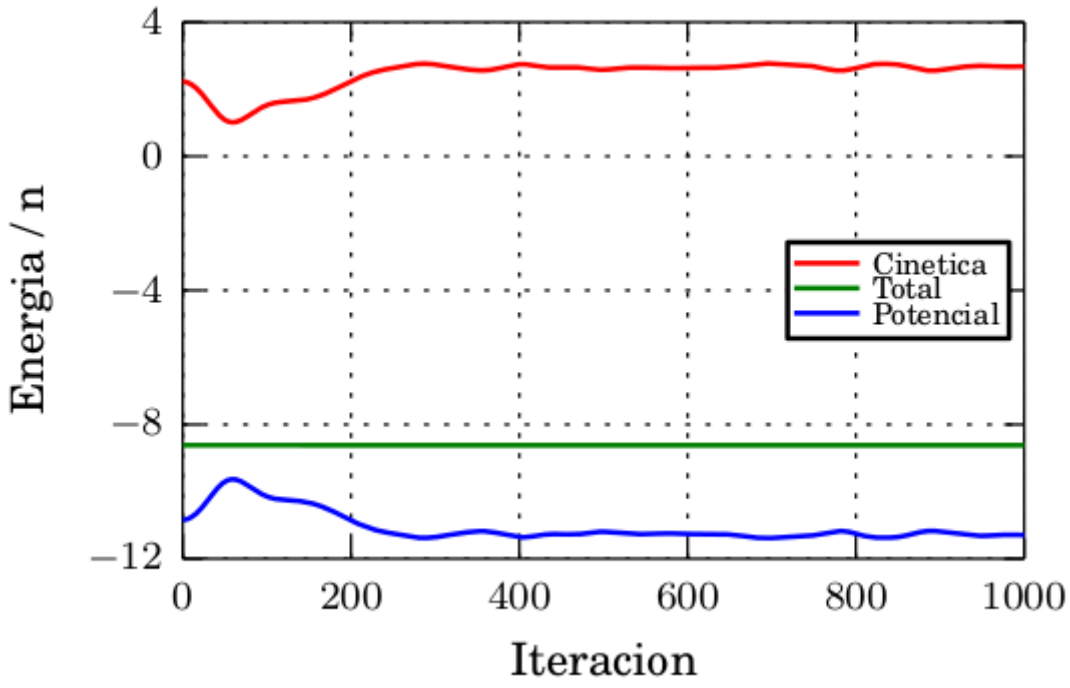
```
void termalizacion()
{
    ciclo pasos temporales verlet();
}
```



Pareciera que el sistema estuviese “congelado”

¿Qué podemos medir para determinar si nuestro sistema termalizó? ¿Nos sirve la energía?

Energías



T = 0.728 y ρ=0.8442.

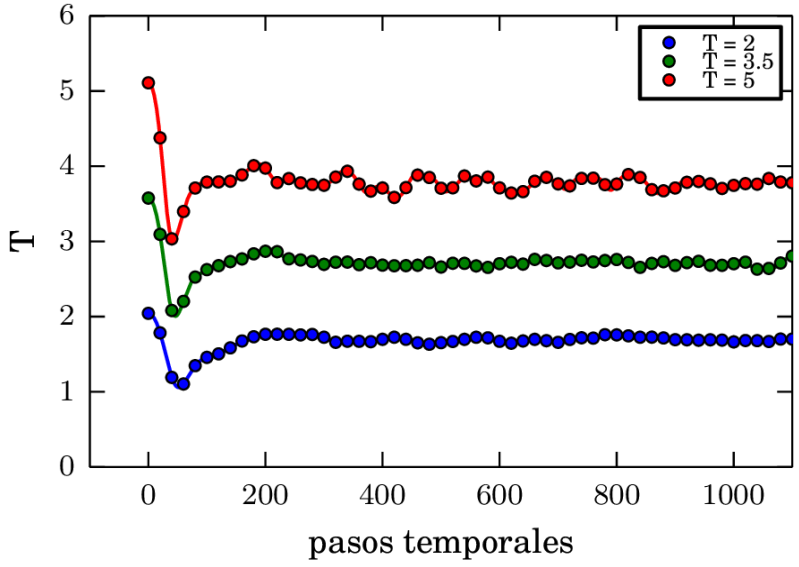
Termalización: temperatura

Estamos usando el ensamble microcanónico (NVE).

$$T = \frac{1}{3}\frac{m}{k_B}(\langle v_x^2\rangle + \langle v_y^2\rangle + \langle v_z^2\rangle)$$

La velocidad de cada partícula varía en el tiempo

$$T(t) = \frac{1}{3N}\sum_{i=1}^{N} m_i(v_{i,x}^2(t) + v_{i,y}^2(t) + v_{i,z}^2(t))$$



Habrá que promediar T durante un cierto intervalo de tiempo (continuo).

Termalización: temperatura

```
main()
{
    condiciones_iniciales();
    tablas();

    fuerzas_iniciales();

    for (nt = 0; nt < nt_maximo; nt++)
    {
        termalizacion();
        T_actual = calculo_temperatura();

        //mediciones();

        T_deseada = T_actual - T_max/(double)nt_maximo;

        reescaleo_velocidades();
    }
}
```

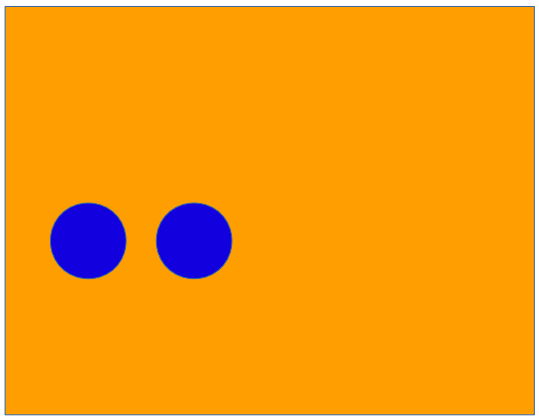
Hacer evolucionar al sistema “V” pasos y medir T(t) en cada paso temporal.

Luego, promediar.

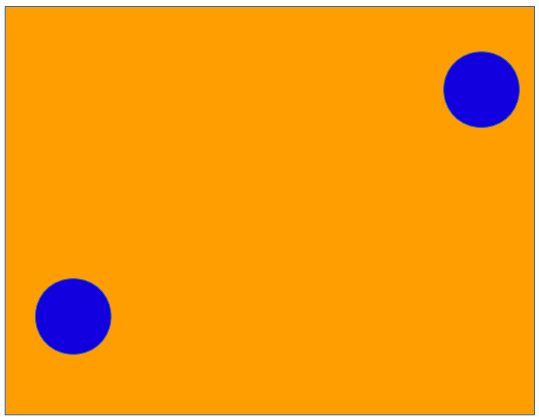
Verlet: rangos

Analicemos cómo calcular las fuerzas

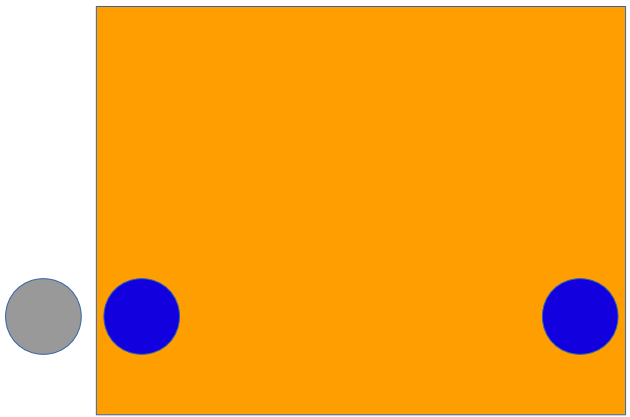
Tendremos alguna de estas tres situaciones:



Interacción “real”



Sin interacción



Interacción “virtual”

Verlet: rangos

```
dx = pos_x[i] - pos_x[j];
dy = pos_y[i] - pos_y[j];
dz = pos_z[i] - pos_z[j];
distancia_cuadrado = dx*dx+dy*dy+dz*dz;

if (distancia_cuadrado <= d_corte_cuadrado) distancia = pow(distancia_cuadrado,1/2.);
else
{
//i está muy lejos de j real, entonces calcula en qué región debería ubicarse la partícula virtual.
//Divide por cuadrantes la caja real y se fija en cada una de las direcciones a que cuadrante va a
//ir a parar la particula virtual

    double dx_abs = fabs(dx);
    double dy_abs = fabs(dy);
    double dz_abs = fabs(dz);

    if(dx_abs >= r_corte) dx = dx - dx*lado/dx_abs;
    if(dy_abs >= r_corte) dy = dy - dy*lado/dy_abs;
    if(dz_abs >= r_corte) dz = dz - dz*lado/dz_abs;

    distancia_cuadrado = dx*dx + dy*dy + dz*dz;
    if (distancia_cuadrado <= d_corte_cuadrado) distancia = pow(distancia_cuadrado,1/2.);
    else
    {
        // Esto sirve para anular la fuerza:
        dx = 0.0;
        dy = 0.0;
        dz = 0.0;
    }
}
```

porque mi tabla usa r y no r^2

Verlet: fuerzas

```
int a = (int) (distancia*nf/r_corte) - 1;
// r = r_corte*(i+1)/(double)nf;

actualizacion = fuerzas[a]*dx/distancia;
f_x_t_h[i] += actualizacion;
f_x_t_h[j] -= actualizacion;

actualizacion = fuerzas[a]*dy/distancia;
f_y_t_h[i] += actualizacion;
f_y_t_h[j] -= actualizacion;

actualizacion = fuerzas[a]*dz/distancia;
f_z_t_h[i] += actualizacion;
f_z_t_h[j] -= actualizacion;
```

Hay que inicializarlos a cero

```
for (int i = 0; i < N; i++)
{
    f_x_t[i] = f_x_t_h[i];
    f_y_t[i] = f_y_t_h[i];
    f_z_t[i] = f_z_t_h[i];
}
```

Guardamos las nuevas fuerzas para usarlas en el siguiente paso temporal (esto lo hacemos a lo último)

$$\begin{cases} x^{(k)}(t+\Delta t) = x^{(k)}(t) + v^{(k)}(t)\ \Delta t + \frac{f^{(k)}(t)}{2m}\Delta t^2 \\ v^{(k)}(t+\Delta t) = v^{(k)}(t) + \frac{f^{(k)}(t+\Delta t)+f^{(k)}(t)}{2m}\Delta t \end{cases}$$

Reescaleo de velocidades: variación de la temperatura

```
main()
{
    condiciones_iniciales();
    tablas();

    fuerzas_iniciales();

    for (nt = 0; nt < nt_maximo; nt++)
    {
        termalizacion();
        T_actual = calculo_temperatura();

        //mediciones();

        T_deseada = T_actual - T_max/(double)nt_maximo;

        reescaleo_velocidades();
    }
}
```

$$v_i^{(k)\prime} = \sqrt{\frac{T_{deseada}}{T_{actual}}}\, v_i^{(k)}$$