Parcial I: Computación Científica Avanzada
Laura Arboleda Hernández
CC 1017220781

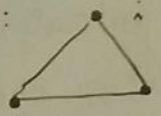
1. Construir triángulos

- Búsqueda de los 2 vecinos más cercanos:
  - ✓ se debe calcular la distancia entre las partículas (x: con todas las demás)

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

Para ello debemos cargar (leer) del archivo, las posiciones de las partículas.

- ✓ Almacenar las distancias en un arreglo: | d_{ij} | j-ésimo ID | } distancia de la i a todas las demás
- ✓ Organizar el arreglo indexado de menor a mayor. Para ello podemos usar el algoritmo burbuja o GSL-sorting. Como el algoritmo burbuja es del orden de N^2 operaciones, usamos GSL.
- ✓ Con el arreglo organizado procedemos a construir los triángulos. Al seleccionar las 2 primeras líneas del arreglo habremos encontrado los 2 vecinos a una partícula dada. El triangulo formado por la partícula y sus 2 vecinas tendrá vértices indexados así:



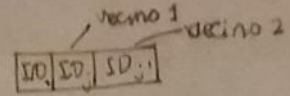
vértices (ID_i, ID_j, $ID_$

Escribimos un archivo con los vertices: "vertices.dat"

2. Contar los triángulos asociados a una partícula

- ✓ Buscamos en las tripletas que definen cada triángulo cuántas veces aparece una partícula dada. Cada vez que encontramos ~~en cual caso~~ si la partícula i es vecina de alguna de las demas partículas, se actualiza un contador de ~~#~~ de triángulos:

- leemos el archivo con los vértices



↳ almacenamos la información de los vecinos 1 y 2 en arreglos V1 y V2 y luego buscamos en cada arreglo:

```
j = 0; j ... Nparticles, j++ → inicializamos (#Δs)j = 1;
    i = 0; .... Nparticles, i++.

        Leer archivo y guardar V1[i], V2[i]

    if (V1[i] == j)
        { (#Δs)j += 1 } → se actualiza el contador de triángulos
    if (V2[i] == j)
        { (#Δs)j += 1 }
```

Se inicializa a 1 para contar el primer Δ que forma la partícula con sus vecinos

3. Número de Triángulos vs Distancia al centro.

- Encontrar el centro del sistema:

→ Ecuaciones del centro de masa: $\vec{r}_{cm} = \frac{\sum_i m_i \vec{r}_i}{\sum_i m_i}$

La masa de todas las partículas es la misma: $m_i = m = 1$:

$$\vec{r}_{cm} = \frac{1}{N_{particles}} \sum_i \vec{r}_i \quad \begin{cases} \vec{r}_{cx} = \frac{1}{N_{partic.}} \sum_i \vec{r}_{xi} \\ \vec{r}_{cy} = \frac{1}{N_{parti}} \sum_i \vec{r}_{yi} \end{cases}$$

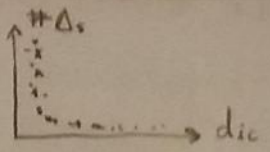
✓ Encontrar la distancia de cada partícula al centro:

$$d_{ic} = \sqrt{(c_x - x_i)^2 + (c_y - y_i)^2}$$

↳ dist. radial.

✓ guardamos en un archivo: | distancia al centro (de cada partícula) | $(\#\Delta_s)_i$ |

→ graficando se encuentra

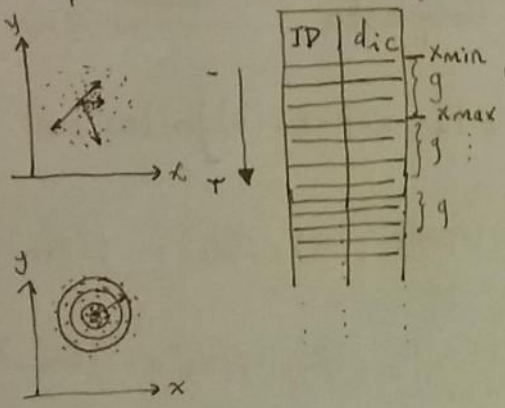


4. Densidad Superficial de Masa de los anillos:

⊛ Para calcular $S(x,y)$

✓ debo crear anillos concentricos de ancho variable, pero todos con igual número de partículas. Para ello usare las distancias al centro calculadas en el punto anterior.

✓ Organizo el arreglo en orden creciente y empiezo a contar partículas en el arreglo. Esta selección nos dará en ancho de los anillos.



✓ Guardamos la partícula con mayor distancia al centro y la de menor distancia, estos serán el radio externo e interno de cada anillo

(sus distancias al centro)

$$\text{Ancho} : (dic)_{max} - (dic)_{min}$$

$(dic)_{max}$: de la última partícula contada

$(dic)_{min}$: de la 1ra partícula contada.

✓ Conociendo los límites de cada anillo, podemos calcular su radio como:

$$r = \frac{(dic)_{max} + (dic)_{min}}{2}$$

✓ la densidad superficial será:

$$S(x,y) = \frac{g \cdot \text{masa}}{\pi\left(|dic|^2_{max} - |dic|^2_{min}\right)_{anillo}}$$

(# de partículas en cada anillo (dados por el usuario))

= densidad superficial de cada anillo.

- Las propiedades: densidad superficial, ancho, radio y radios interno y externo, de cada anillo se guardaran en una estructura

```
struct rings
{
    double width;
    double density;
    double radius[2];
    double r;
};
struct rings *ring
```

5. Masa Total contenida en cada anillo

- Usando un método de Simpson Compuesto:

$$\int_a^b f(x)\,dx \simeq \frac{h}{3}\left[f(a) + 4I + 2P + f(b)\right]$$

donde: $h = \frac{b-a}{n}$; $I = \sum_{i=1\to impar}^{n-1} f(x_i) = f(x_1) + f(x_3) + \dots f(x_{n-1})$

$$P = \sum_{i=2\to par}^{n-2} f(x_i) = f(x_2) + f(x_4) + \dots f(x_{n-2})$$

en nuestro caso $b = R_f$ (radio del anillo mas externo) $> h = R_f/n$

$a = 0$

y lo que queremos encontrar es: $M_{tot} = \int_S \sigma(R)\,dS$ (diferencial de superficie)

$$\therefore \quad M_{tot} = \int_0^{2\pi}\int_0^{R_f} \sigma(R)\,R\,dR\,d\theta = 2\pi \int_0^{R_f} \underbrace{\sigma(R)R}_{f(x)}\,dR$$

Luego usando simpson compuesto....

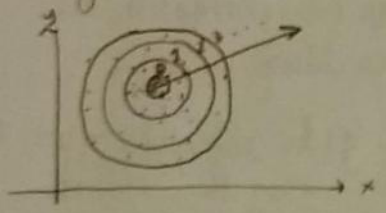
$$M_{tot} = 2\pi \int_0^{R_f} \sigma(R)R\,dR \simeq \frac{2\pi h}{3}\left[\sigma(0)\,\emptyset + 4I + 2P + \sigma(R_f)R_f\right]$$

$h = \frac{R_f}{n}$ ^ $I = \sum_{i=1,n}^{n-1} S(R_i) R_i$ ^ $P = \sum_{i=2-n}^{n-1} S(R_i) R_i$

✓ hay que tener en cuenta que la densidad en $R=0$ no la conocemos. Luego hacemos una extrapolación (trivial) y asignamos a ~~tu densidad~~ del anillo ø un radio cero a mano.

ring[0].r = 0.0

(los anillos creados serán un total de Nparticles · #anillos) y nuestra forma de llenar cada anillo con partículas, como el arreglo es creciente será desde el centro hacia afuera, es decir:



✓ Para calcular las sumas I, P y final/. la masa total:

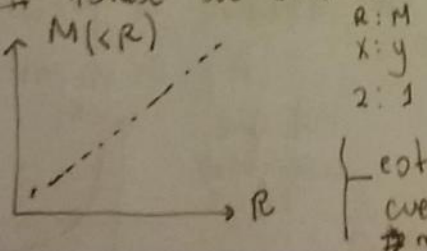
hasta qué anillo quiero ver la masa acumulativa.

```
i=0 .... Npoints/2 -1 ; i++
{
  P += S[2*i] * R[2*i]
  I += S[2*i+1] * R[2*i+1]
}
```

$$M_{bl} = 2\pi * \frac{h}{3} * (4 * I + 2 * P + S(R_f) * b)$$

✓ Para el perfil de masa acumulativa, hacemos el procedimiento anterior pero variando Npoint desde 1 hasta el # total de anillos. Con esto obtendremos

Masa Acum	R



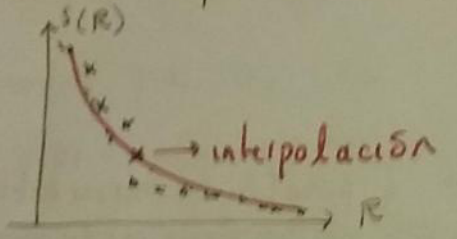
esta gráfica que debe obtenerse da cuenta de la distribución de partículas ⊕ # mas al centro.

- Masa encerrada a un radio $R_i < R_f$

$$M(R_i < R_f) = 2\pi \int_0^{R_i} S(R) R \, dR$$

6 Identificación de R_e tal que $S(R_e) = S_0/e$
($S_0 \rightarrow$ densidad en el centro de la distribución)

- dado que de puntos anteriores ya conocemos la densidad superficial como una función del radio (para ciertos radios) usamos estos puntos para interpolar los datos y así conocer la función $S(R)$ a otros radios:



Con gsl interpolamos los datos usando spline cúbico. De esta forma conocemos la curva roja (la función)

✓ Usando un método de bisección (pues nuestra función es bien comportada) buscaremos las raíces de la ecn:

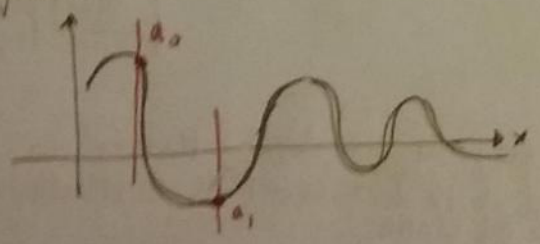
$$S(R) - S_0/e = 0 \quad (1)$$

es decir el radio para el cual se cumple (1).

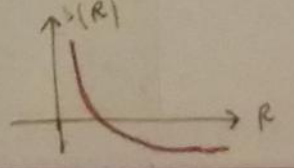
- de la interpolación conocemos el spline:

$$S(R) = \text{gsl\_spline\_eval}(\text{spline}, R_i, \text{acc})$$

como el método de bisección funciona con una función de la forma:



desplazamos la función $S(R)$ en un factor α tal que se cumpla $\rightarrow S_0/e$



el método funciona partiendo el intervalo a la mitad y seleccionando el subintervalo que contendrá la raiz

si inicialy. los extremos del intervalo eran: a_0 y a_1, $a_2 = \frac{1}{2}(a_1 + a_0)$ } esta subdivisión se hace siempre y cuando las funciones en a_0 y a_1 tengan signos diferentes. Así se van reajustando los extremos del intervalo hasta que se encuentra la raiz. La implementación del método la haremos así:

$f(a_0) = gsl\_spline\_eval(spline, a_0, acc) - \alpha$ ← S₀/e
$f(a_1) = gsl\_spline\_eval(spline, a_1, acc) - \alpha$

siempre que $a_1 > a_0$ ...

✓ si $f(a_0) \cdot f(a_1) > 0$

{ las funciones tiren el mismo signo y la raiz puede no ser encontrada }

✓ sino ... $a_2 = \frac{1}{2}(a_1 + a_0)$

$err = \frac{1}{2}|a_0 - a_1|$ } error

$f(a_2) = gsl\_spline\_eval(spline, a_2, acc) - \alpha$

✓ while ($err > \varepsilon$ y $n_{pasos} < n_{máx}$) ← controlará el # de pasos usados para buscar la raiz

{ if ($f(a_0) * f(a_2) < 0$) { $a_1 = a_2$;
$f(a_1) = gsl_\ldots\ldots(spline, a_1, $
}

else { $a_0 = a_2$
$f(a_0) = gsl\_spline\_\ldots\ldots(spline, a_0, acc) - \alpha$
}

n_{pasos} ++
$a_2 = \frac{1}{2}(a_1 + a_0)$
$err = \frac{1}{2}|a_0 - a_1|$
$f(a_2) = gsl_\ldots\ldots(spline, a_2, acc) - $