

# INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

INGENIERÍA EN NANOMATERIALES I

PRIMAVERA 2019



ITESO, Universidad  
Jesuita de Guadalajara

PRÁCTICA DE LABORATORIO 04

## SIMULACIÓN DE SPUTTERING MEDIANTE MONTE CARLO

### EQUIPO

NT704804 CHIÑAS FUENTES, KARINA

NT702357 HERNÁNDEZ MOTA, DANIEL

NT696464 MORA TRUJILLO, KIM ALEJANDRO

NT702281 PADILLA GODÍNEZ, FRANCISCO JAVIER

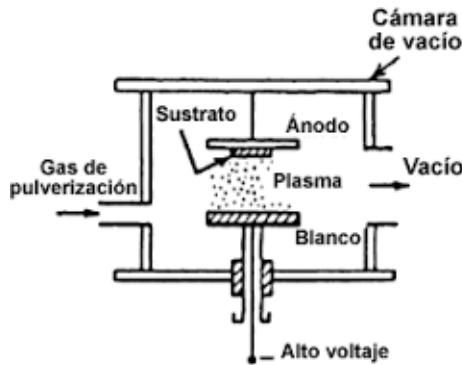
PROFESOR: JULIO HERIBERTO MATA

TLAQUEPAQUE, JALISCO

FECHA DE ENTREGA: 14 DE MAYO DE 2019

# 1. INTRODUCCIÓN

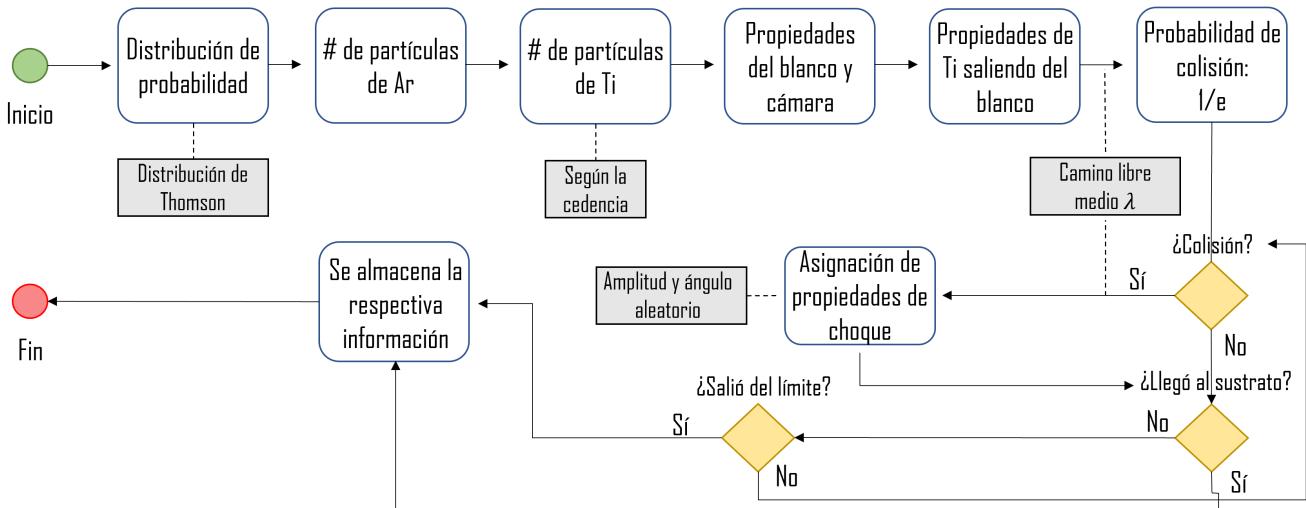
El proceso de *sputtering* es uno de los métodos principales de deposición física para la generación de partículas delgadas nano-estructuradas. Este proceso se basa en la generación de plasma (usualmente de gas inerte o una combinación de inerte con reactivo) el cual, puede o no ser controlado mediante un campo magnético, para bombardear y así pulverizar un blanco para que el material se deposite sobre un substrato [1]. Gracias a la naturaleza del proceso, este se puede simular mediante el método de Monte Carlo, el cual se basa en la aleatoriedad de números para poder hacer: integrales, optimización de algoritmos o distribuciones de probabilidad [2]. Para este caso, se va a utilizar para generar una distribución de probabilidad que se acomode a las observaciones numéricas del fenómeno de *sputtering*. La figura 1 muestra un simple diagrama del proceso.



**Figura 1:** Idea general del proceso de *sputtering* [3].

Hacer una simulación mediante el método de Monte Carlo ayuda a obtener una buena aproximación sobre lo que sucede en una cámara de depósito de *sputtering* ya que esto simplifica por mucho el problema comparado a intentar hacer una reproducción determinista; no obstante, aunque se quisiera hacerlo determinista, se necesitaría muchísimo poder de cómputo, ya que el fenómeno físico conlleva a un análisis de más de  $10^{23}$  grados de libertad [4].

La motivación para hacer una simulación de *sputtering* es que ayuda a desarrollar y sostener ideas de depósito sin que necesariamente se utilice el equipo; esto es importante porque algunos de los blancos y substratos que se necesitan para depósitos en específico son muy caros. Además de dinero, también se ahorra tiempo. La figura 2, muestra un diagrama de flujo de la lógica seguir para desarrollar el código.



**Figura 2:** Diagrama de flujo para el código.

## 2. OBJETIVO

El objetivo es llegar a un mejor entendimiento del proceso de *sputtering* mediante la implementación de la simulación del mismo, en PYTHON, haciendo énfasis paso a paso del funcionamiento y principio utilizado en este método para tener una mejor aproximación a la realidad. Para ello, se considera un proceso hecho con un gas de argón y un blanco de titanio.

## 3. CÁLCULOS MATEMÁTICOS

Previo a la simulación, es importante calcular los parámetros que delimitan el proceso, tal como el camino libre medio  $\lambda$  y la cedencia  $Y(E)$ . Para dichos cálculos, vamos a tomar valores con los que hemos trabajado en las prácticas de laboratorio; eso es: presión  $p = 1,5 \times 10^{-2}$  Torr (que es equivalente a casi 2 Pa) y temperatura  $T = 300K$ .

Para el camino libre medio  $\lambda$ :

$$\lambda = (\pi n d^2)^{-1}. \quad (1)$$

Siendo  $d$  la suma del radio del Ti y el ,  $d = 0,23$  nm 6 y  $n$  el número de moles determinado por:

$$n = \frac{p}{kT} = \frac{2\text{Pa}}{1,38064 \times 10^{-23}\text{JK}^{-1}300\text{K}} = 4,8287 \times 10^{20}. \quad (2)$$

Haciendo que  $\lambda = 1,25$  cm.

En cuanto a la cedencia  $Y(E)$ , se tiene que [4]:

$$Y(E) = \frac{1,8\alpha}{U} \cdot \left(\frac{M_t}{M_i}\right) \cdot \left(\frac{Z_t Z_t M_t}{(M_i + M_t) \cdot (Z_t Z_i)^{2/3}}\right)^{1/2} \cdot \left(1 - \frac{E_{TH}^{1/2}}{E^{1/2}}\right)^2 \cdot E^{1/2} \quad (3)$$

Siendo:  $U$  la energía de cohesión superficial  $U = 3,3$  eV [5],  $E$  es la energía con la que inciden los iones  $E = 350$  eV,  $E_{TH}$  es el valor de *threshold* (de umbral)  $E_{TH} = 5$  eV,  $Z_i$  y  $M_i$  es el número atómico y peso atómico del áromo incidente (en este caso, Ar), respectivamente; siendo ambos:  $Z_i = 18$  y  $M_i = 39,948$  (g/mol),  $Z_t$  y  $M_t$  es el número atómico y peso atómico del blanco (*target*, de Ti), respectivamente; siendo ambos:  $Z_t = 22$  y  $M_t = 47,867$  (g/mol) y, por último,  $\alpha$  está dado por [6]:

$$\alpha = 0,08 + 0,164 \left(\frac{M_t}{M_i}\right)^{0,4} + 0,0145 \left(\frac{M_t}{M_i}\right)^{1,29} = 0,2741095. \quad (4)$$

Por lo que  $Y(E) = 5,904171$ .

## 4. CÓDIGO EN PYTHON

Se decidió hacer el código en PYTHON ya que es el lenguaje de programación que todos los integrantes del equipo conocemos, además de que éste es un lenguaje de código libre que cuenta ya con muchas funciones y librerías que permiten facilitar el procedo de codificación. Para hacer el código, primero se importan todas las librerías necesarias o principales que permiten al código trabajar con matemáticas *avanzadas* y que al mismo tiempo pueda graficar los resultados.

```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects

from numpy import sin, cos, sqrt, pi
from matplotlib.pyplot import savefig

# Para hacer graficas
plt.style.use("seaborn-poster")

plt.rc('text', usetex = True)
plt.rc('font', family = 'serif')

```

Se consideró primero la distribución de probabilidad dada por la relación de la función de Thompson [4]

$$F(E) \propto \frac{E}{(E + U)^3} \rightarrow F(E) = \frac{\alpha E}{(E + U)^3}, \quad (5)$$

donde  $E$  es la energía,  $\alpha$  es una constante de escalamiento y  $U$  es la energía de unión de superficie. Para este, caso, vamos a considerar  $\alpha = 100$  y  $U = 3,3$ . Con ello, se genera un histograma normalizado y se utilizan dichos valores para generar un peso estadístico que posteriormente se va a utilizar para decidir la velocidad de una partícula de Ti saliendo del blanco. Para incorporarlo en el código, se consideró lo siguiente.

```

points=100000

y=100*x/ (x+4)**3

a=np.random.uniform(0,max(x),points)
b=np.random.uniform(0,max(y),points)

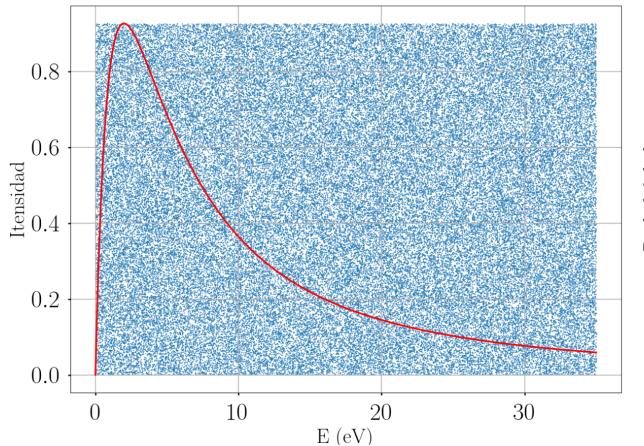
for p in range(points):
    flag=False
    if b[p]<=100*a[p]/(a[p]+4)**3:
        flag=True
    for i in range(n):
        if i == 0:
            if a[p]<bars[i] and flag==True:
                histogram[0]+=1 #Para categorizar
                break
        else:
            if a[p]>bars[i-1] and a[p]<bars[i] and flag==True:
                histogram[i]+=1
                break

histogram=histogram/sum(histogram)

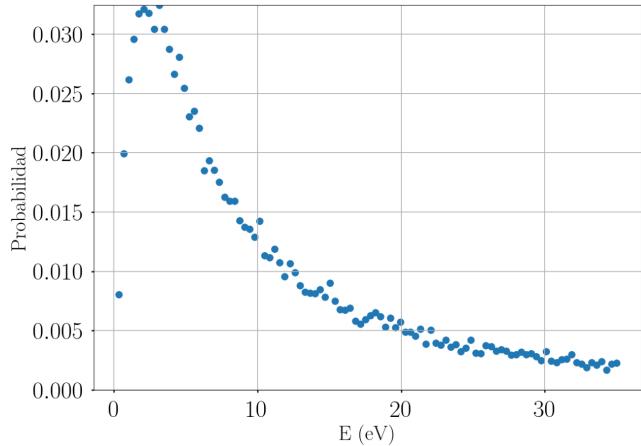
# Comprobar normalizacion
sum(histogram)

```

La figura 3 muestra la función de distribución y el histograma normalizado, para los pesos estadísticos.



a) Función de Thomson y valores random



b) Histograma normalizado.

**Figura 3:** Creación de pesos estadísticos a partir de la función de Thomson

Luego, sigue inicializar un número de partículas de Ar que van a incidir contra el blanco de Ti para saber cuántas partículas de Ti van a despegarse del material.

```
y=5.904171

Ti = 0
for i in range(Ar):
    ty = np.random.uniform(0,1,1)
    if ty > y:
        pass
    elif ty <= y:
        Ti += 1
```

Para caracterizar a las partículas que salen del blanco de Ti, se hace que las partículas de argón incidan sobre todo el blanco (para evitar el caso de *sputtering* puntual).

```
def create_particle(bars,histogram,lenght,blancomin,blancomax):
    #Seleccionar la velocidad del histograma
    velocity=np.random.choice(bars,p=histogram)

    #Dar un valor específico de la velocidad
    particle_v=np.random.uniform(low=(velocity-lenght),high=velocity)

    #Dar un valor específico del angulo de la particula
    particle_a=np.random.uniform(0, np.pi)

    posx=np.random.uniform(blancomin,blancomax)
    posy=0    #Siempre sale de la superficie
    pos =[posx,posy]
    particle=[particle_v,particle_a,pos]
    return particle
```

Es importante recalcar que en PYTHON la identación de código es vital para tener un código que funcione bien. En cuanto a las delimitaciones del blanco y de la cámara, se tomaron los siguientes parámetros (en centímetros).

```

#Propiedades del target
blancomin = 3 #pos en x
blancomax = 5 #pos en x longitu total: 2cm
particlesTi= [create_particle(bars,histogram,lenght,blancomin, blancocomax) \
    for i in range(Ti)]
countgood = 0 #contador
countbad = 0 #contador
#Dimensiones de camara
camxmin = -5
camxmax = 11
camymin = 0
camymax = 20

```

Lo siguiente es establecer el comportamiento principal del programa, el cual básicamente evalúa si una partícula de Ti se sale o no de la región que evita que la partícula termine en el substrato o no; así como su alteración de trayectoria tras una colisión. Todo este análisis se hace en un `for`, el cual tiene dura según el número de átomos de Ti que salen por bombardeo.

```

for particle in particlesTi:
    objective=False #bandera: revisa si pego al blanco.
    operations=True #bandera: revisa si sigue dentro de la camara.
    #Definir la posicion
    vectorposx=[]
    vectorposy=[]
    posx=particle[2][0]
    posy=particle[2][1]
    vectorposx.append(posx)
    vectorposy.append(posy)
    recorrido = 1.2 #camino libre medio
    thresh = 1/np.exp(1) # valor "thresh" de umbral

```

Luego, se describe la mecánica la colisión de partícula o de determinar si se sale o no del área para que llegue al sustrato.

```

while operations==True:
    #Determinar si colisiona la particula
    collision=np.random.uniform(0,1)
    if collision < thresh:
        collisiontrue=True
    else:
        collisiontrue=False
    #Determinar cuando no colisiona
    if collisiontrue == False:
        posx=posx+recorrido*np.cos(particle[1])
        posy=posy+recorrido*np.sin(particle[1])
        pos =[posx, posy]
        if abs(posx)>camxmax:
            operations=False
        elif posy<camymin:
            operations=False
        elif (posy>camymax) and (abs(posx)>5): #Si se sale del sustrato
            operations=False
        elif (posy>camymax) and (abs(posx)<5): #Por si no colisiono
            operations=False
        objective=True

```

Si la partícula colisiona, entonces su amplitud de desvío se verá afectado por un valor uniformemente aleatorio que va entre 1 a 7; junto con un ángulo aleatorio que va de 0 a  $2\pi$  radianes.

```

elif collisiontrue == True:
    modificacion=np.random.uniform(1,7)*recorrido
    posx=posx+recorrido*np.cos(particle[1]) + \
        modificacion*np.cos(np.random.uniform(0,2*np.pi))
    posy=posy+recorrido*np.sin(particle[1]) + \
        modificacion*np.sin(np.random.uniform(0,2*np.pi))
    pos=[posx,posy]
    if abs(posx)>camxmax:
        operations=False
    elif posy<camymin:
        operations=False
    elif (posy>camymax) and (abs(posx)>5):
        operations=False
    elif (posy>camymax) and (abs(posx)<5):
        operations=False
        objective=True
    vectorposx.append(posx)
    vectorposy.append(posy)
    particle[2] = pos

```

Por último se evalúa si las partícula en cuestión llegó o no al sustrato, para entonces así seguir con el resto de las partículas.

```

if objective==True:
    #Cuenta de partículas que llegan al sustrato
    countgood+=1
    plt.plot(vectorposx,vectorposy,c='aquamarine', linewidth=.1)
elif objective==False:
    #Cuenta de partículas que no llegan al sustrato
    countbad+=1
    plt.plot(vectorposx,vectorposy,c='k', linewidth=.1)
    #plt.plot(vectorposx,vectorposy, )

```

Una vez que los datos han sido almacenados, se pide al programa que grafique los resultados. Por último, se le hace saber al usuario qué porcentaje, de todas las partículas que salieron del blanco, se adhirieron al substrato o no.

```

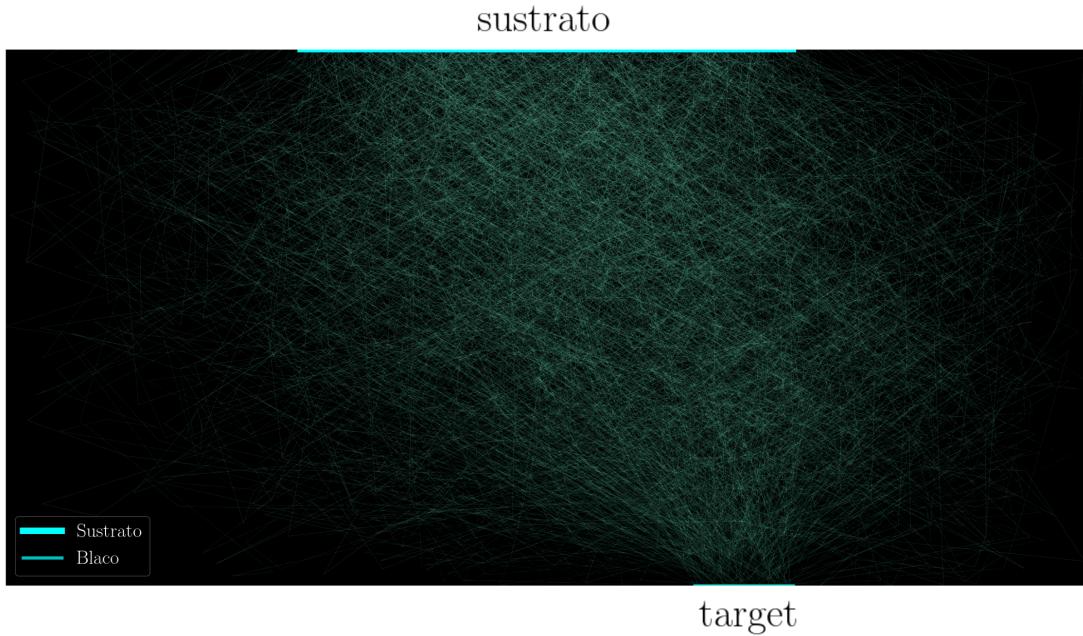
plt.xlim(camxmin,camxmax)
plt.ylim(camymin,camymax)
plt.plot([-5,5],[20,20],c= 'aqua', linewidth=10, label="Susbstrate")
plt.plot([blancomin,blancomax],[0,0] ,c= 'c', linewidth=5, label="Target")
plt.xlabel("$x$-position_(cm)", fontsize = 50)
plt.ylabel("$y$-position_(cm)", fontsize = 50)
plt.legend()
plt.title("Sputtering_chamber_simulation")
plt.show()

counttotal=countgood+countbad
total_particles_sub_hit=countgood/counttotal
print(f"Percentage_of_particles_in_the_substrate:{total_particles_sub_hit*100}%")

```

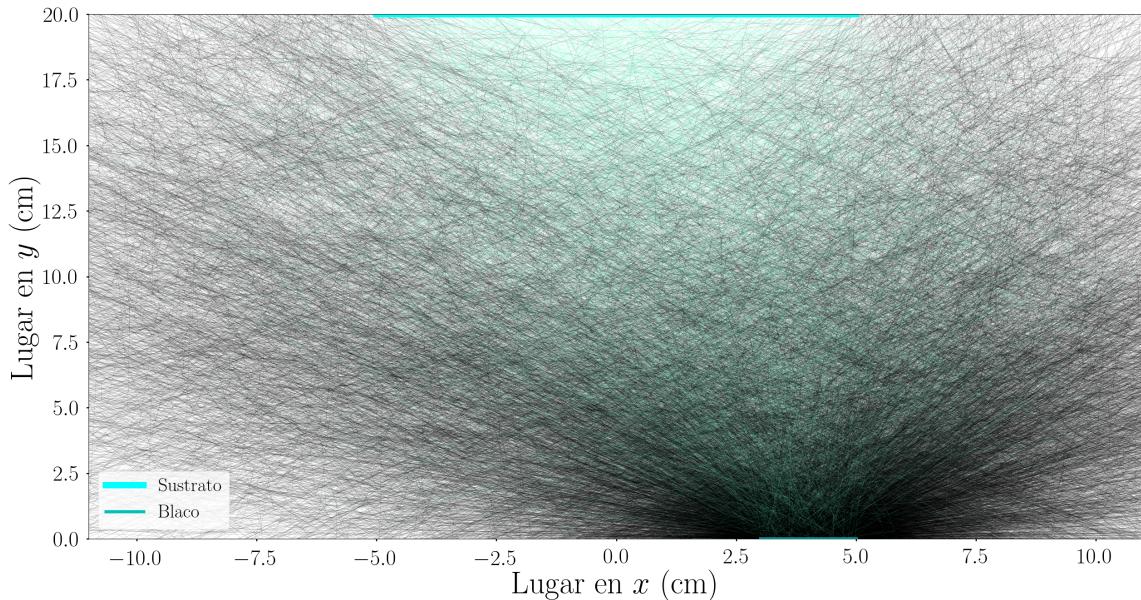
## 5. RESULTADOS DEL CÓDIGO

A continuación, se presentan algunas de las imágenes que fueron resultados del código. La figura 4 muestra los resultados de haber puesto que iban a colisionar 10 mil partículas de Ar sobre el blanco de Ti.



**Figura 4:** Resultado con 10 mil partículas de Ar.

Las todas líneas de color indican que éstas llegaron al sustrato; mientras que las trayectorias de que no lograron llegar al sustrato se pintaron de negro para que se perdieran con el fondo. Según el código, para los resultados de la figura 4, sólo el 9.68 % lograron llegar al sustrato. La figura 5, muestra un resultado de 10 mil partículas de argón, pero ahora sí se puede distinguir en la imagen cuáles sí llegaron y cuáles no, así como su respectiva posición en cm.



**Figura 5:** Resultado con 10 mil partículas de Ar, con distinción de colores y posición relativa.

Los resultados para la figura 5 indican que 9.52 % de las partículas de Ti lograron llegar al sustrato. Cabe mencionar que la computadora empieza a mostrar problemas cuando el número de partículas supera las 12 mil y tarda mucho más para dar un resultado de 11500 partículas. Por lo que, podríamos decir que estos valores nos dan una idea sobre el poder computacional que el código requiere para dar resultados importantes.

## 6. OBSERVACIONES Y CONCLUSIONES

Al ser el proceso de *Sputtering* un modelo probabilístico se podría mejorar la simulación siguiendo la distribución de *Maxwell-Boltzman* para la velocidad promedio de las partículas, ya que este acercamiento debería ser un poco más consistente con las observaciones que se tienen en el proceso real. Se esperaría una respuesta más realista, elevando el número de partículas, sin embargo, debido al tiempo del cálculo, se optó por un número reducido de partículas, esperando que con el apoyo de un GPU más potente o, en última instancia, un servidor de procesadores se cree un ambiente con características más similares a la situación real.

Por último, cabe mencionar, que debido al desconocimiento de algunos procesos físicos del sistema (y por consecuencia no hay ecuaciones que lo describan) se tiene que optar por métodos que suponen aleatoriedad y funciones de distribución para poder determinar el comportamiento del sistema. Por lo que se hacen muchas suposiciones y se consideran en general condiciones ideales (como asumir que cada partícula que llega al sustrato se adhiere a este).

En conclusión, la simulación del proceso de *Sputtering* tiene un peso considerable en la comprensión del mismo, permitiéndonos programar paso a paso los mecanismos que permiten que esta técnica sea tan utilizada y bien comprendida.

---

## Referencias

- [1] Depla, D.; Mahieu, S.; Greene, J.E. (n.d.). "Sputter deposition processes". Ghent University, Department of Solid State Sciences and University of Illinois, Materials Science and Physics Departments.
- [2] Kroese, D. P.; Brereton, T.; Taimre, T.; Botev, Z. I. (2014). "Why the Monte Carlo method is so important today". WIREs Comput Stat. 6 (6): 386–392. doi:10.1002/wics.1314.
- [3] Angarita, G. L. G. (2017). "SÍNTESIS DE PELÍCULAS DELGADAS POR LA TÉCNICA DE MAGNETRÓN SPUTTERING A PARTIR DE BLANCOS DE RENIO Y BORO". Universidad EAFIT, Departamento de Ciencias Físicas.
- [4] Heriberto, M. J. (2019). "Sputtering Processing Simulation". Curso: Ingeniería en Nanomateriales I. Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO).
- [5] Kudriavtsev, Y., Villegas, A., Godines, A., Asomoza, R. (2005). Calculation of the surface binding energy for ion sputtered particles. Applied Surface Science, 239(3-4), 273–278. doi:10.1016/j.apsusc.2004.06.014
- [6] Seah, M. P., Nunney, T. S. (2010). Sputtering yields of compounds using argon ions. Journal of Physics D: Applied Physics, 43(25), 253001. doi:10.1088/0022-3727/43/25/253001