Estudio del fenómeno de percolación 2D optimizando bloques de código mediante slices en Python

Study of the 2D Percolation Phenomenon by Optimizing Code Blocks Using Slices in Python

Gustavo Medina-Ángel*[1, 3, 4], Gennadiy Burlak[1]
Erika Martínez-Sánchez[2] y Anahí Yelitza De la Cruz Abarca[1, 4]

[1]CIICAp, Universidad Autónoma del Estado de Morelos
Av. Universidad 1001, Cuernavaca, Morelos 62209, México
[2]Facultad de Ingeniería, Universidad Autónoma de Coahuila
Blvd. Fundadores Km. 13 Ciudad Universitaria, Arteaga, Coahuila 25354, México.
[3]FCAeI, Universidad Autónoma del Estado de Morelos
Av. Universidad 1001, Cuernavaca, Morelos 62209, México
[4]EESM, Universidad Autónoma del Estado de Morelos
Av. Subida al Calvario s/n, Col. Justo Sierra, C.P. 62230, Morelos, México
*gustavo.isc@hotmail.com

PALABRAS CLAVE:

Slices, Python, Percolación, Optimización

RESUMEN:

Realizamos dos códigos para analizar el fenómeno de percolación en su dimensión 2D, en el cual uno de ellos se desarrolló de manera clásica utilizando ciclos for para analizar los arreglos bidimensionales, y otra en donde utilizamos sentencias slices de python remplazando bloques de código a una forma tipo slice. Se realizaron varias pruebas para diferentes tamaños de rejilla en de diferentes dimensiones (L) para ambos códigos (con ciclos for y con slices) y se encontró una considerable mejora en la optimización de tiempos cuando se utilizan sentencias slices en comparación a los ciclos for convencionales.

KEYWORDS:

Slices, Python, Percolation, Optimization.

ABSTRACT:

We developed two codes to analyze the percolation phenomenon in its 2D dimension. One of them was implemented in the classical manner using for loops to analyze the two-dimensional grids, while the other employed Python slice statements, replacing blocks of code with a slice-like structure. Several tests were conducted for different grid sizes (L) and dimensions for both codes (using for loops and slices). A significant improvement in time optimization was observed when using slice statements compared to conventional for loops.

• **Recibido: 24 de noviembre de 2024** • **Aceptado: 5 de julio de 2025** • **Publicado en línea: 2 de junio de 2026**

1. INTRODUCCIÓN

La percolación asociada generalmente con el fenómeno de filtración [1] es un tema de estudio presente en la construcción de materiales, pues de ello depende si un material carece o no de la suficiente porosidad para filtrar o no un fluido determinado por un clúster infinito [2, 3]. El fenómeno de filtración se puede encontrar también de manera natural por ejemplo en la construcción de estalactitas o llamadas también espeleotemas [4] en la cual las crecientes rocosas son guiadas por las diferentes rutas de filtración y que por efecto de la gravedad tienden a bajar al suelo, formando con dirección vertical estructuras rocosas que se pueden admirar

DOI: 10.30973/progmat/2026.18.2/2

en un entorno arquitectónico natural, tal como se muestra en la figura 1. La ciencia como en todo momento de la historia ha tratado de replicar los fenómenos naturales, entre ellos el fenómeno de percolación, fenómeno aprovechado para la construcción de diversos materiales con percolación, como por ejemplo; cuando se necesita saber cuál es la porosidad suficiente de un filtro [5] de un sistema de purificador para que solo las partículas de H2O puedan filtrarse a través del material. Por otro lado, en los sistemas donde se involucra el ahorro de materiales, en los cuales los materiales para la construcción de una vivienda habitacional necesitan ser ligeros para ahorrar y optimizar la mayor cantidad de posible de material, pero evitando filtraciones de fluidos y humedad, es presión conocer la probabilidad critica o nivel de porosidad que se necesita para evitar tales filtraciones [6,7]. Sin embargo, tal fenómeno no solo es aplicable al procesos de los fluidos si no que también puede asociarse a otros fenómenos como las guías de luz en el direccionamiento de partículas [8, 9], o en la conducción de energía mediante nano estructuras CNTs [10, 11].

Figura 1. Formaciones rocosas de estalactitas con filtraciones naturales en Cacahuamilpa Guerrero, México.

En suma, se puede deducir que la utilización y aplicación del efecto de percolación es relevante para generar procesos importantes que ayudan a cubrir necesidades humanas relevantes. Los modelos para optimizar fenómenos como el efecto de percolación, se han desarrollado eficazmente mediante algoritmos computacionales utilizando arreglos bidimensionales y tridimensionales, que al simularlos y dependiendo del tamaño del modelo, el nivel de porosidad y los recursos computacionales (como la RAM y procesador) será la rapidez a la que se simule el modelo. A menudo estos arreglos son analizados con ciclos for, ya que se tiene que analizar un sistema bidimensional o tridimensional en cada una de sus posiciones, puesto que esta albergará o no un poro(que puede unirse a otro en sucesivo para formar un clúster) utilizando lenguajes de programación modernos como C++, C# o Java. Sin embargo tanto los lenguajes de programación se han modernizado no solo para brindarnos una mejor comprensión, facilidad y manejo del lenguaje sino que también que se han equipado y agregado bibliotecas para la invocación rápida y fácil de sus métodos como lo es el lenguaje de programación Python. En este artículo se han desarrollado un sistema utilizando el método Slice de Python para optimizar el fenómeno de percolación medido en tiempo de ejecución.

Este articulo está distribuido de la siguiente manera: En el capítulo 2, se Analiza el fenómeno de percolación desde una perspectiva típica, utilizando bucles for, en el capítulo 3, se optimiza el simulado de percolación utilizando métodos slices, en el capítulo 4, se muestra los resultados de comparación entre los bucles for y los slices en python, nuestras ultimas secciones contienen nuestra discusión, conclusiones y trabajo a futuro.

2. ANÁLISIS DEL FENÓMENO DE PERCOLACIÓN 2D ASISTIDA POR BUCLES FOR

Para la construcción del sistema clásico se

ha desarrollado un código utilizando bucles For, donde cada bucle lleva el control de las columnas, filas (auxiliadas por las variables i, j) El ciclo interno maneja las columnas, el ciclo externo controla las filas del sistema. Los métodos principales para simular el fenómeno de percolación 2D son; 1) generar el tamaño del arreglo, así como los valores aleatorios para los radios de los poros y generar dos capas de entrada y de salida del filtro que representan las capas de vacío o aire por donde puede filtrarse el fluido, 2) encontrar la conglomeración de poros, es decir la unión de varios poros formando clústeres (utilizando la lógica del vecino cercano con sus cuatro posibilidades, arriba, abajo, izquierda y derecha, es decir los 4 vecinos que rodean al poro (ver Ec. 1 y 3) encontrar la respuesta de percolación en el sistema simulado. Estos procesos los podemos observar en la figura 2(a-c), donde se observan los procesos de simulación (1-3) antes descritos.

Cabe señalar que el procesamiento numérico para encontrar los clústeres de poros fue necesario la utilización de la siguiente ecuación:

$$C=\sum_{i=0}^{L-1}\sum_{j=0}^{L-1} M(i,j)+M(i,j-1) \quad (1)$$

Donde cada sumatoria representa un bucle dimensional para 2D, donde las variables i, j son las responsables de llevar el control de cada repetición, j-1, representa que se está analizando el vecino que se encuentra arriba del poro. L, representa el tamaño del arreglo, M la matriz o el arreglo y C la respuesta entera que se tendrá al analizar el vecino, si este es mayor a 1, entonces significa que se ha encontrar un vecino.

En la figura 3, se define la función Fill Array2D, el cual recibe los parámetros: fi, co (filas y columnas) que constituyen el tamaño del arreglo LxL. Se prepara una matriz [] y una lista llamada fila que se utilizaran para el llenado de valores binarios utilizando la anidación de dos ciclos con parámetros de control row y colum, que repiten las instrucciones hasta que se alcancen los límites del tamaño fi y co. Posteriormente se genera los números aleatorios utilizando randint con valores de 1 hasta 100 para ser evaluados posteriormente. Se crean las capas superiores etiquetando con 2 la primera fila (fila.append(2)) utilizando la condicional: if row==0 y la última fila (fila.append(1)) con la condicional: elif row==fi-1. Mientras que para todo el espacio sobrante del arreglo se llena ya sea con un 0 ó 1.

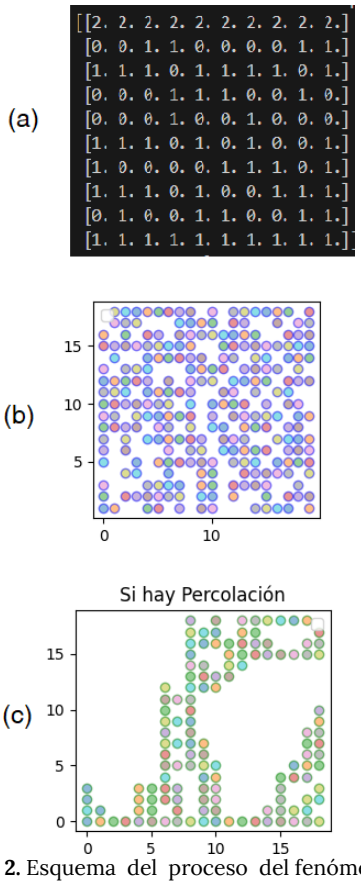


Figura 2. Esquema del proceso del fenómeno de percolación: (a), muestra los valores binarios a partir de los números aleatorios generados, (b), muestra los clústeres (poros aglomerados) encontrados en la matriz y (c), muestra el clúster de percolación infinito, indicando si existe o no percolación en el sistema.

Para el caso en donde el valor del arreglo toma el calor de 0, se debe de cumplir la condición r > 100 - Probabilidad,

donde si el valor es mayor que la probabilidad asignada la posición de arreglo evaluado tomara un valor de 1, mientras que si el valor fuese menor, se agregara un valor de 0 en esa posición. Por último se agrega cada fila a la matriz y una vez completados todos sus valores se retorna a la función que a la vez es almacenada en una variable result. El siguiente paso es encontrar los clústeres (el clúster infinito si es que lo hay), y la respuesta para encontrar si existe o no percolación dentro del sistema. Por lo que generamos una función FindPercol2D, donde se aceptan como parámetros el arreglo binario generado por el método anterior (def Fill_Array2D(fi, co)), y las dimensiones del mismo sobre las variables fi y co. Posteriormente se declaran dos bucles anidados que llevan el control sobre la matriz result y para todos aquellos valores que en su posición exista un poro con el valor de 1 (if result[row][colum]>0), y se analizan sumando los 4 vecinos que se encuentran alrededor(con la instrucción: m=result[row][colum] + result[row+1][colum] + result[row-1][colum] + result[row][colum+1] + result[row][colum-1]), el resultado de esta sumatoria se almacena en la variable "m". A la par analizan también estas mismas posiciones pero ahora verificando si existe algu´n vecino que tenga algún valor de 2(b=(result[row+1][colum]==2) or (result[row-1][colum]==2) or (result[row][colum+1])==2 or (result[row][colum-1]==2)), esto es necesario ya que la fila 0 se determinó como una capa de aire, por lo que el siguiente poro que aparezca con el valor de 1 se considera como un hueco dentro del sistema por lo tanto tendrá que convertirse en 2 (if m>1 and b==True: result[row][colum]=2) para combinarse con la capa de aire antes declarada(if m>1 and b==True: result[row][colum]=2), de contrario este permanecerá en 0 (else: result[row][colum]=0)(solido del material). Esta última instrucción va determinado el camino de los poros y clu´steres combinados con la fila de inicio =0, formando un clúster que alcanza o no el otro extremo de la rejilla.

```
def Fill_Array2D(fi,co):
    matriz=[]
    for row in range(fi):
        fila=[]
        for colum in range (co):
            r=randint(1,100)
            if row==0:
                fila.append(2)
            elif row==fi-1:
                fila.append(1)
            elif r>=100-Probabilidad:
                fila.append(1)
            else:
                fila.append(0)
        matriz.append(fila)
    return matriz
result=Fill_Array2D(Filas, Columnas)
print(result)
```

Figura 3. Imagen del código para la generación de la matriz de bordes y valores binarios aleatorios, correspondiente a la imagen 2(a).

El Resultado de analizar estos 4 vecinos se almacena en la variable "b", si alguno de los 4 vecinos es verdadero, significa que se ha encontrado al menos un poro unido al poro analizado. El último paso para este procedimiento es encontrar la respuesta de percolación, para ello es necesario analizar la penúltima fila del sistema(for colum in range(co-1):), si en esta penúltima fila se encuentra un valor de 2, significada que el clúster se ha combinado desde la base y ha alcanzado la última capa del arreglo logrando una exitosa percolación(if result[fi-2][colum]==2: isper='Si hay Percolación').La respuesta de este análisis sobre la penúltima capa se guardara en la variable isper para posteriormente ocuparla para etiquetar nuestros gráficos (como en el ejemplo de la figura 5). Lo restante es devolver la variable result que contiene los valores de los clústeres encontrados etiquetados con 2(return result). Ver código figura 4.

En la figura 5, se muestra el proceso completo del fenómeno de percolación 2D

con vista de un clúster infinito creciente atravesando de la rejilla de un extremo al otro, con percolación exitosa (Si hay percolación) y con parámetros de dimensión L=100 (donde los valores de fi=co=L del código de las figuras 3 y 4) y una probabilidad de ocupación de P=0.7. Sin duda el punto protagónico aquí es la optimización de tiempos, considerando este paradigma de programación descrita y una forma auxiliada por los métodos slices de Python.

```
def FindPercol2D(result, fi, co):
    m=0
    row=1
    colum=1
    for row in range(fi-1):
        for colum in range (co-1):
            if result[row][colum]>0:
                m=result[row][colum]+
                result[row+1][colum]+
                result[row-1][colum]+
                result[row][colum+1]+
                result[row][colum-1]
                b= (result[row+1][colum]==2)
                or (result[row-1][colum]==2)
                or (result[row][colum+1])==2
                or (result[row][colum-1]==2)
                if m>1 and b==True:
                    result[row][colum]=2
                else:
                    result[row][colum]=0
    for colum in range(co-1):
        if result[fi-2][colum]==2:
            isper='Si hay Percolación'
            break
        else:
            isper='No hay Percolación'
    return result
```

Figura 4. Imagen del código para la búsqueda de los clústeres en el sistema 2D, y respuesta de percolación correspondientes a las imágenes de la figura 2(a) y 2(b), respectivamente. Las instrucciones subrayadas en rojo indican que esta es una sola instrucción, pero en este ejemplo fueron divididas para una mejor visualización (Frecuentemente juntar instrucciones para formar instrucciones largas suelen tener una ejecución más eficiente [12]).

3. ANÁLISIS DEL FENÓMENO DE PERCOLACIÓN 2D AUXILIADO POR SLICES

El método silice() es un método eficiente y corto cuando se trata de trasladar datos de una lista a otra [13], para entender este concepto y como funciona se presenta un ejemplo con las principales formas en las que se pueden aplicar dicho método. En la imagen de la figura 6 se observa un ejemplo en donde se genera una lista llamada números de valores aleatorios, con el método random.rand(6), de seis elementos con un límite de valor de 100.

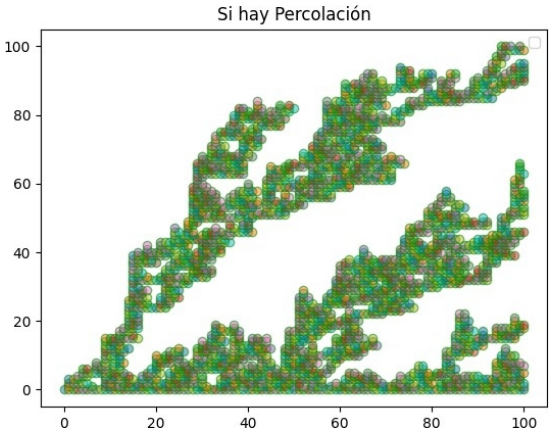


Figura 5. Simulación de fenómeno de percolación 2D, con tamaño de la matriz L=100, y probabilidad P=0.7, con percolación verdadera.

Para imprimir estos números generados utilizamos la instrucción de la impresión 1; números[:], donde indicamos la separación de inicio y fin con corchetes es decir de la siguiente manera: [inicio:fin], donde el inicio denota que por defecto ser ‘a el ´índice 0, mientras que el fin, por defecto denota el final de la lista que por defecto será el límite del tamaño del arreglo, la instrucción y el resultado de esta impresión se muestran en las imágenes de la figura 6(a, b) en el numeral “1.”.

Por otro lado, el equivalente de la instrucción numeros[:], también se puede escribir como: numeros[0:6], donde el primer elemento tomara como índice el valor de cero y terminara hasta un valor de 6 indicando que el slice debe parar su refiriéndose al límite de nuestra lista e indicando al slice que debe parar su rutina cuando llegue a tal valor (ver figura 6(a, b) en el numeral “2.”). Se tiene un tercer ejemplo en la figura 6(a, b) en el numeral “3.”, en

donde se tiene la instrucción: números [0:6:2], la cual indica el recorrido de la lista des del índice 0 hasta el término de la lista 6, con un salto de 2. Un cuarto ejemplo en la figura 6(a, b) en el numeral "4." ımprime los valores de la lista (un rango) pero comenzando en el ´índice 1 hasta un tope de -1, indicando que debe parar una posición antes del último elemento de la lista. En el quinto ejemplo de la figura 6(a, b) en el numeral "5." se invierten los valores de la lista (numeros[-1::-1) indicando que se debe comenzar en la última posición de la lista(-1), recorrer todas sus posiciones (::) y en un decremento de -1. Después de ello se hace una asignación de tales valores al arreglo para invertir dichos números; numeros[:] = numeros[-1::-1] y se imprime la lista para verificar su inversión y asignación en la lista original, como se muestra en figura 6(a, b) en el numeral "6.".

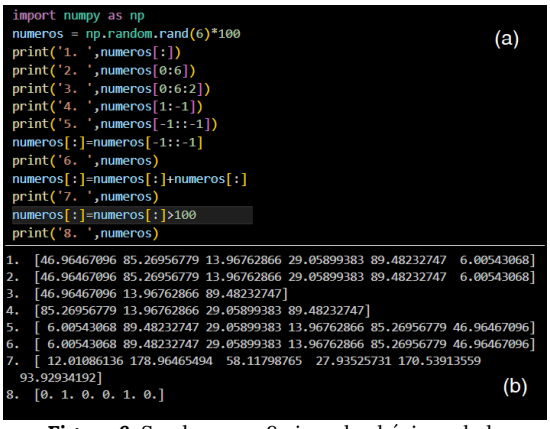


Figura 6. Se observan 8 ejemplos básicos de la utilización de slices en python. La sección (a), muestra las instrucciones de los ejemplos, mientras que en la sección (b) se muestran los resultados de las ejecuciones para cada una de las instrucciones (1-8).

Por otra parte se pueden sumar valores a los elementos de la lista siempre y cuando compartan la misma dimensión como se muestra en el ejemplo 7 de la figura 6(a, b), donde el valor de cada posición de la lista se le suma el valor de su misma posición con la instrucción; numeros[:] = numeros[:] + numeros[:]. Por último se realiza una condición para los valores de la lista numeros[:], si el valor de la lista en cada posición es mayor que 100, entonces el valor será verdadero y se asignara un 1, de lo contrario se asignara un 0. Ver figura 6(a, b) en el numeral de impresión "8.".

Una vez comprendido el funcionamiento del método slice(), podemos implementar esta propiedad al fenómeno de percolación pero ahora para 2 dimensiones (2D). Para ello se define el mismo método: Fill_Array2D(fi, co), utilizado en la lógica anterior con bucles For, en donde en primera instancia se utiliza el método random.rand para construir la matriz con las dimensiones fi y co, y que a su vez dichos números aleatorios no rebasen un límite de valor de 100.

```
def Fill_Array2D(fi,co):
    matriz =(np.random.rand(fi,co)*100)
    matriz[0:1,:]=2
    matriz[-1:,:]=1
    matriz[1:-1,:]=matriz[1:-1,:]>=100-Probabilidad
    return matriz
```

Figura 7. Se muestra el código reducido utilizando el método slice(), equivalente al funcionamiento del código de la imagen 3.

Una vez realizado el llenado de la matriz con valores aleatorios, se consideran la parte inferior y superior de arreglo considerados como los extremos donde terminan la rejilla de simulación (aire) y por lo tanto el final del material. Para la parte inferior del material se llenaran con un valor de 2 la primera fila, utilizando la instrucción; matriz[0:1,:]=2, donde los valores [0,1,:], representan el inicio de la fila(0), hasta la primera fila(1) y para todas las columnas (:) de la matriz en su primera fila.

Para la parte superior de la matriz realizamos un procedimiento similar con el slice utilizando la instrucción; matriz[-1:,:]=1, en donde se establece que ahora se deberá comenzar a llenar en la penúltima fila (-1:) hasta el término del arreglo y para todas las columnas (:) en su última fila. Los valores restantes del arreglo (matriz[1,-1] se deberán llenar con un valor binario que se

valida con: matriz[1,-1]>=100-Probabilidad, el valor tomara un valor de 1 si la condición se cumple, de lo contrario se asignara un 0 por defecto.

Un punto importante de recalcar es que aunque los slices permiten condiciónales no se permiten realizar más de una comparación, es por ello que la siguiente construcción para encontrar los clústeres en el arreglo utilizamos los bucles for para este paso (código de la imagen 4). Sin embargo la última fase de para identificar si existe o no un clúster infinito de percolación se redujo a la instrucción; res=result[-2:-1,1:-1].any(), en donde se examina la penúltima fila del arreglo(-2), y si existe por lo menos un valor diferente de cero el método any(), devolverá un True, de lo contrario se devolverá un valor booleano False que se guardara en la variable res. Esta relación la podemos observar en la figura 8, donde la parte comentada se reduce posteriormente a una sola instrucción. Por ultimo toca retornar a la matriz (return matriz) para su llamado a la función.

```
# for colum in range(co-1):
#     if result[fi-2][colum]==2:
#         isper='Si hay Percolación'
#         Percol=True
#         break
#     else:
#         isper='No hay Percolación'
#         Percol=False
res=result[-2:-1,1:-1].any()
```

Figura 8. Implementación del método slice con la instrucción: res=result[-2:-1,1:-1].any(), que sustituye las instrucciones comentadas anteriores (en verde).

4. RESULTADOS

Una vez realizada la programación pertinente, la siguiente tarea fue comparar los tiempos de ejecución para ambos casos; primero cuando el fenómeno es programado únicamente con bucles for y la segunda cuando cambiamos y remplazamos fragmentos de código con slices(Para el primer y último paso del fenómeno del esquema 2(a, c)). Para ello se consideraron dimensiones cuadradas del arreglo de LxL=20, 40, 60, 80, 100, 120, 140, 160, 180 y 200, con un rango de probabilidad de P =0.35 hasta P =0.7, graficando 36 puntos por cada una de las 10 L o dimensión de nuestras pruebas, dando un total de 360 ejecuciones. Para tomar el tiempo utilizamos el método time.time() al inicio y al final de cada prueba ejecutada teniendo como resultados los valores de la figura 9, donde cada línea de color representa una dimensión que es graficada en un rango de probabilidad 0.35 a 0.7(rango amplio en donde se encuentra la probabilidad critica Pcr).

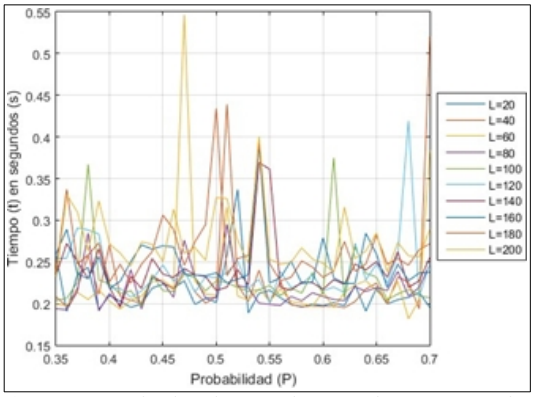


Figura 9. Resultados de corrida para el caso con ciclos For para las dimensiones LxL=20,40,60 ..200, con un rango de probabilidad P=0.35 hasta P=0.7.

Se puede observar que para todas las dimensiones existe un rango de tiempo de ejecución entre el tiempo t=0.2 hasta t=0.3 aproximadamente y de manera regular. Se realizó el mismo procedimiento para el caso cuando se implementan slices en el estudio del fenómeno y con los mismos parámetros (P=0.35-0.7, LxL=20...200). Los resultados para esta prueba muestran una mejora considerable en los resultados, ya que para las dimensiones probadas se tienen un rango de tiempo de t=0.0009789466857910156 segundos hasta t=0.21742582321166992 segundos.

En la figura 10 se observan tales resultados en donde se puede apreciar claramente que para valores de dimensiones pequeñas existe

un tiempo bajo de calculación pero mientras el arreglo aumenta su tamaño, el tiempo de calculación también incrementara.

Para tener una apreciación más clara de los tiempos de ambas calculaciones se realiza una comparación de los tiempos de ambos procedimientos en una sola gráfica, esta relación se puede observar en la Figura 11, donde las líneas graficadas en color rojo son las pertenecientes a las prueba realizadas con bucles for (valores de la figura 9), mientras que las líneas graficadas en color azul, corresponden a las pruebas implementando slices (valores de la figura 10). Se puede observar una disminución importante de tiempos en segundos(s) de las líneas azules en relación a las líneas rojas.

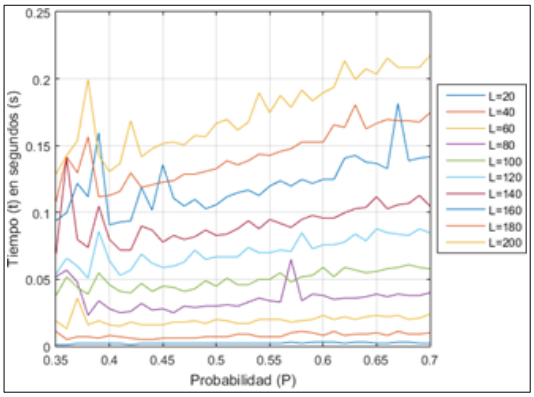


Figura 10. Resultados de corrida para el caso con Slices, cada línea representa una dimensión del arreglo L.

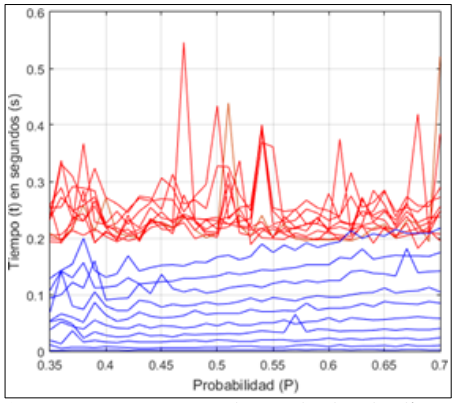


Figura 11. Comparativa de resultados, las líneas graficadas en color rojo son las pertenecientes a las prueba realizadas con bucles for, mientras que las líneas graficadas en color azul, corresponden a las pruebas implementando slices.

5. DISCUSIÓN

Los códigos desarrollados para comparar ambas técnicas (Con bucles for y auxiliadas de slices), se implementaron con poros de radios fijos r = 0,5 [3, 14], parámetro fijo en el cual, el poro puede alcanzar a tocar alguno de los cuatro vecinos analizados (el de arriba, abajo, izquierda y el de la derecha), así como la probabilidad critica (Pcr) ya bien conocidas de este fenómeno con valor de Pcr=0.5927 para la percolación en 2D [15]. Dado que el objetivo de esta investigación es optimizar una mejora en el código remplazando los ya conocidos bucles for por los métodos slices (que provee un lenguaje de alto nivel como lo es Python), los poros de radio aleatorio no se abordan en esta investigación. Una consideración importante a notar es que los slices no son métodos triviales y comunes, el uso de estos significa una compresión detallada, sin embargo cuando se domina tal método, puede ser de gran apoyo en el análisis de información. Las pruebas numéricas realizadas para nuestras simulaciones se implementaron en una laptop Dell i7, Gen. 6, con velocidad de procesador de 2.6 GHZ, con 4 núcleos físicos y 4 lógicos, con 16 de memoria RAM y con un SO Win10, por lo que los resultados pueden variar dependiendo de los recursos de hardware de cada computadora.

Por otra parte podemos analizar los ahorros en tiempo y porcentajes por cada calculación utilizando la ecuación (2).

$$pt = 100 - (ts/tf * 100) \quad (2)$$

En donde *pt* es el porcentaje de tiempo, *ts* el tiempo obtenido de la calculación usando slices y *tf* el tiempo obtenido con el ciclo for. Por ejemplo para el primer punto (p1), cuando el arreglo tiene una dimensión de L=20 y probabilidad P=0.35, se tiene que para el tiempo optimizado usando slices el tiempo ts=0.000978946685791015 y un tiempo tf= 0.258190631866455 con ciclos for, el tiempo de ahorro en porcentaje sería igual a

99.62084345, aplicamos esta regla a todos los puntos de probabilidad P y para todas las dimensiones L, los resultados de esta relación de los ahorros en porcentajes las podemos observar en la figura 12, donde se muestra el valor real de ahorro cuando se utilizan métodos slices para el estudio y simulación de este fenómeno en 2D.

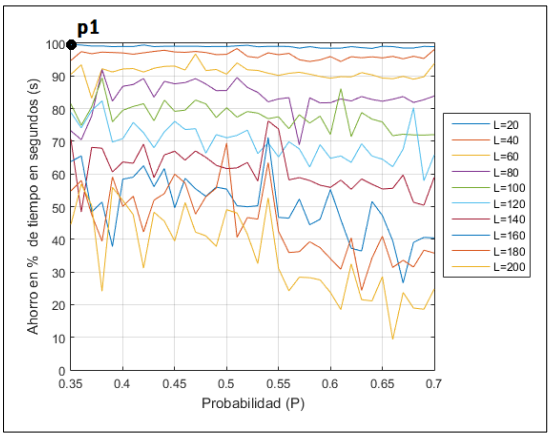


Figura 12. Se muestran los porcentajes de ahorro en segundos cuando se utilizan métodos slices, para un rango de probabilidad de 0.35 a 0.7 para las dimensiones de arreglo L=20, 40, 60,...200. Se observa también que mientras el valor de las dimensiones incrementan el ahorro decrece pero conservando todavía porcentajes de ahorros entre el 50% y el 20% para los rangos de probabilidad 0.35 a 0.7 como lo es para el caso de la dimensión L=200, con el espectro más amplio simulado para este caso de estudio.

6. CONCLUSIONES Y TRABAJO A FUTURO

Se estudió la optimización del fenómeno de percolación en 2D implementado bloques de instrucciones en un código desarrollado con bucles For los cuales fueron remplazados por métodos slices de Python. Se realizaron un conjunto de pruebas para verificar la velocidad en tiempo de ejecución de cada caso, y se obtuvo que el código donde se implementaron slices resulto en un mejor rendimiento en comparación al código que solo se implementó con bucles For para este caso de estudio. Los resultados muestran un 99% de ahorro en tiempo de ejecución cuando se simulan arreglos pequeños (L=20) y hasta un 20 % para arreglos prominentes (L=200) sometidos en un rango P=0.35 a 0.7. Las bondades de los slices son múltiples por lo que en un segundo trabajo futuro se pretende investigar e implementar slices que satisfagan la simulación del fenómeno en su integridad y extender la optimización del fenómeno mediante slices y otros métodos en su versión simulada 3D. La figura 13, muestra la imagen de percolación 3D(utilizando bucles For.), siguiente objetivo a optimizar mediante slices.

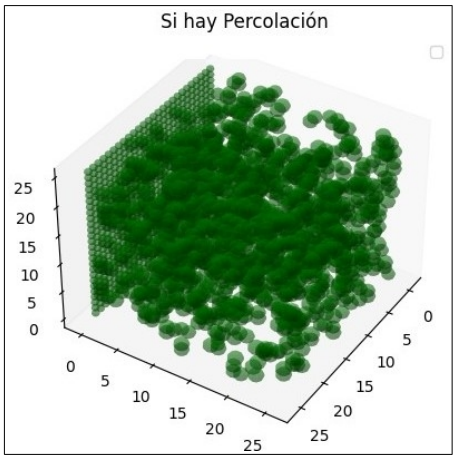


Figura 13. Simulación de fenómeno de percolación 3D en python, utilizando bucles for, con tamaño de la matriz L=25, y probabilidad P=0.4, con percolación verdadera.

Agradecimientos

Este trabajo fue apoyado en parte por CONAHCYT (México) bajo la subvención No. A1-S-9201. G. M-A. Agradece una beca otorgada por el CONAHCYT-México.

Disponibilidad de datos

Los datos que respaldan la investigación de este estudio están disponibles bajo el autor de correspondencia con previa solicitud razonable.

REFERENCIAS

[1] Hunt A, Ewing R, Ghanbarian B. Percolation theory for flow in porous media. Lecture Notes on Physics,

Vol. 880. Berlin: Springer; 2014. doi: 10.1007/978-3-319-03771-4.
[2] Childs EC, Collis-George N. The permeability of porous materials. Proceedings of the Royal Society of London Series A Mathematical and Physical Sciences. 1950;201(1066):392-405. doi: 10.1098/rspa.1950.0068
[3] Medina-Ángel G, Calderón-Segura YY, Burlak G, Hernández-Aguilar JA. Study of the critical probability of percolation in a 3D system with pores of random radius for variable grids. Revista Mexicana de Física. 2020;66(3):315-321. doi: 10.31349/revmexfis.66.315
[4] Vidal Romaní JR, Vilaplana JM. Datos preliminares para el estudio de espeleotemas de cavidades graníticas. Cadernos do Laboratorio Xeolóxico de Laxe, 1984, 7: 305-324. Disponible en: http://hdl.handle.net/2183/5861.
[5] Saberi AA. Recent advances in percolation theory and its applications. Physics Reports. 2015;578:1-32. doi: 10.1016/j.physrep.2015.03.003
[6] Bentz DP. Virtual pervious concrete: Microstructure, percolation, and permeability. ACI Materials Journal. 2008;105(3):297-301. doi: 10.14359/19827
[7] Han B, Chen X, Pan Y, Wang C, Shi M, Chu X. Diffusion Model of Cement Slurry in Porous Media Considering Porosity Variation and Percolation Effect. Applied Sciences. 2023;13(3):1919. doi: 10.3390/app13031919
[8] Martinez Sanchez E, Burlak G, Medina-Ángel G, Magallanes Rivera R. Optical Field Localization in the Three-Dimensional Percolating System with Gaussian Distribution Disorder. International Journal of Optics. 2024;2024:3983413. doi: 10.1155/2024/3983413
[9] Burlak G, Díaz-de-Anda A, Malomed BA, Martinez-Sánchez E, Medina-Ángel G, Morales-Nava R, Martínez-Ocampo JJ, de-Anda-Reyes ME, Romero-López A. Critical properties of the optical field localization in a three-dimensional percolating system: Theory and experiment. Chaos Solitons & Fractals. 2023;173:113734. doi: 10.1016/j.chaos.2023.113734
[10] Burlak G, Medina-Ángel G. The optimization at studying of electrical conductivity in the dielectric nanocomposites with disordered nanotubes. Progress In Electromagnetics Research Letters. 2018;74:77-82. doi: 10.2528/PIERL17120407
[11] Medina-Ángel G, Burlak G. Software para calcular la probabilidad crítica de percolación en un sistema de nanotubos desordenados aplicado al estudio de sus radios en materiales conductores. Programación Matemática y Software. 2019;11(2):24-34. doi: 10.30973/progmat/2019.11.2/4
[12] Medina-Ángel G, Chávez MC, Burlak G. The Time optimization for polynomial series using a long For in C. International Journal of Combinatorial Optimization Problems in Informatics. 2020;11(3):130-135. Disponible en: https://ijcopi.org/ojs/article/view/170
[13] McKinney W. Python for data analysis. Sebastopol, CA, USA: O'Reilly Media Inc; 2022.
[14] Allen R, Hansen JP, Melchionna S. Molecular dynamics investigation of water permeation through nanopores. The Journal of Chemical Physics. 2003;119(7):3905-3919. doi: 10.1063/1.1590956
[15] Scher H, Zallen R. Critical density in percolation processes. The Journal of Chemical Physics. 1970;53(9):3759. doi: 10.1063/1.1674565

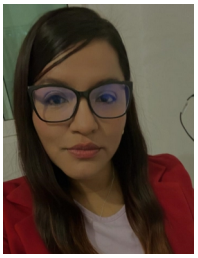
ACERCA DE LOS AUTORES

El Dr. Gustavo Medina Ángel, estudio la Ingeniería en Sistemas Computacionales, egresado de Instituto Tecnológico de Zacatepec en el 2008, en el 2016 se tituló como Maestro en Ingeniería y Ciencias Aplicadas de la Universidad Autónoma del Estado de Morelos y en el año 2020 obtuvo el título de Doctor en Ingeniería y Ciencias Aplicadas en la misma institución, actualmente es docente de la Facultad de Contaduría, Administración e Informática (desde enero 2011), ha impartido 90 cursos referentes a las ciencias básicas y aplicadas, informática e ingeniería. Es desarrollador de Software Independiente y ha impartido talleres de programación avanzada en Java, Programación en Arduino, Bases de datos, circuitos eléctricos y electrónicos, programación móvil, optimización, internet de las cosas e inteligencia artificial. Ha publicado artículos y capítulos de libro como autor y co-autor en revistas internacionales como: Progress in Electromagnetic Research, Revista Mexicana de Fisica, International Journal of Combinatorial Optimization Problems and Informatics, Chaos, entre otros. Se ha desempeñado como árbitro en revistas científicas indexadas internacionales como: Progress in Electromagnetic Research (PIER), Argentine Journal of Science and Technology, SN Computer Science entre otros. Es candidato al sistema nacional de investigadores y sus líneas de investigación a las que se dedica son; la simulación de fenómenos físicos, los métodos numéricos aplicados a la computación, la optimización

de procesos y el procesamiento digital de imágenes.

Dr. Gennadiy Burlak, estudió en la Universidad Nacional de Kiev (KNU), Ucrania, en el Departamento de Física Teórica. El Doctor en Ciencias Físico-Matemáticas. Trabajó como catedrático TC del Departamento de Física Teórica de KNU. A partir de 1998 – a la fecha: Profesor Investigador Titular "C" del CIICAP de la Universidad Autónoma del Estado de Morelos, México, nivel 3. El Dr. Burlak tiene el mérito estatal en categoría Investigación científica 2022. Es Investigador Nacional Emérito. El Dr. Burlak es autor y coautor de 12 libros y capítulos de libros, 175 artículos publicados en revistas internacionales, 186 ponencias en congresos nacionales e internacionales. Bajo de su dirección han graduado: Doctorado: 8, de cuales 6 son miembros de SNI. Líneas principales de investigación son: la percolación óptica, teoría electromagnética, microesferas multicapas, radiación óptica de nano-estructuras, entrelazamiento cuántico, física de solitones e inteligencia artificial.

La Dra. Erika Martínez Sánchez estudió la licenciatura en Matemáticas Aplicadas en la Universidad Autónoma de Coahuila (UAdeC) en 2010. Posteriormente, realizó sus estudios de maestría y doctorado en la Universidad Autónoma del Estado de Morelos, en el CIICAP. Actualmente es Profesor-Investigador Titular C de la Facultad de Ingeniería de la UAdeC. La investigadora pertenece al sistema nacional de investigadores por el CONAHCYT, en el nivel 1. Cuenta con la participación en congresos nacionales e internacionales, coautoría en capítulos de libros y es autor y coautor de artículos arbitrados e indexados. Además de haber asesorado tesis de licenciatura y maestría.



La Mtra. Anahí Yelitza De La Cruz Abarca, es ingeniero en sistemas computacionales e ingeniero industrial por el Instituto Tecnológico de Zacatepec. Cuenta con la Maestría en Psicopedagogía en la Normal Lic. Benito Juárez, así mismo cuenta con la Maestría en Comercialización y conocimientos innovadores del Centro de Investigación en Ingeniería y Ciencias Aplicadas de la Universidad Autónoma del Estado de Morelos. Ha trabajado colaborativamente en diferentes proyectos de investigación cuantitativa y asesoría para el desarrollo de empresas tradicionales al comercio electrónico aplicando el marketing digital, así como el emprendimiento de nuevos negocios, innovación tecnológica de pymes y asesoría de propiedad intelectual, recientemente implemento una propuesta de plan de marketing para la aplicación de un producto de reciente creación, desarrollado vigilancia tecnología, en proyectos en la etapa inicial, así como la asesoría de un plan estratégico y el desarrollo de grupos de trabajo en empresas de Morelos y Puebla. Se ha dedicado al asesoramiento a empresas y negocios estables de grupos de trabajo para el desarrollo de proyectos y servicios innovadores en diferentes áreas del conocimiento. Ha participado en colaboración para el Programa de Estímulos a la Innovación (PEI), en conjunto con equipos de trabajo para la viabilidad de la gestión tecnológica. A impartido en diplomados de gestión empresarial, aplicando las normas de calidad de ISO y Pymes, colaborando en el asesoramiento de herramientas tecnológicas y canales de distribución igual que ha impartido diplomados relacionados en recursos humanos, gestión de proyectos. Sus líneas de investigación son la automatización y optimización de procesos.