Nook'iin Overview

Ossiel Aguilar-Spíndola

2024

Índice general

Introducción

En la naturaleza existen sistemas a escala atómica[1] cuya razón entre superficie y volumen es $\geq 10^3$ lo que permite clasificarlos como materiales bidimensionales (2D). Uno de los materiales 2D más conocidos y estudiados desde su síntesis por exfoliación mecánica en 2004 por Geim y Novoselov es el grafeno [1]. El grafeno es un sistema compuesto de átomos de carbono dispuestos en un arreglo hexagonal en forma de panal de abeja con grosor de un átomo. El grafeno se puede encontrar como una lamina atómica apilada verticalmente formando grafito, el cual es aquel que encontramos en la punta de un lápiz, figura 1.1(a). Vale la pena mencionar que las fuerzas que mantienen unidas a las láminas de grafeno es mucho más débil que aquellas fuerzas que mantienen unidos a los átomos en el plano.[2]

Como consecuencia del trabajo de Geim y Novoselov, surgió el interés por el estudio teórico y/o experimental (o la combinación de ambos) de otros materiales 2D; experimentado un aumento notable del interés en la comunidad científica. Debido a que sistemas como el grafito están compuestos por láminas de grafeno débilmente unidas, es posible pensar que otros sistemas al igual que el grafeno como el disulfuro de molibdeno (MoS_2) sean utilizados para construir, usando un mecanismo de apilamiento vertical, un sistema heterogéneo de dos o más láminas con propiedades físicas distintas, creando así lo que se denomina heteroestructuras. Estas heteroestructuras cuentan con potenciales aplicaciones en la industria electrónica y optoelectrónica [2, 3].

La formación de estas heteroestructuras es posible debido, nuevamente, a las fuerzas de interacción entre las capas atómicas, que son significativamente más débiles que las fuerzas que mantienen unidos a los átomos dentro de cada capa. Para ser preciso, el tipo de fuerzas débiles son de carácter dispersivo de tipo van der Waals (vdW).[3] A este tipo de sistemas se les conoce como homo- y hetero-estructuras de van der Waals. El apilamiento de las capas se puede vislumbrar cómo algo semejante a un juego de LEGO,

[1] En escala de 10^{-10} metros.

[2] Las fuerzas que mantienen unidas a las láminas son de carácter dispersivo, mientras que aquellas que unen a los átomos son por enlaces químicos.

[3] Interacciones dipolo-dipolo son las más representativas.

donde las láminas atómicas se ensamblan como bloques de construcción capa por capa, ver figura 1.1(b).

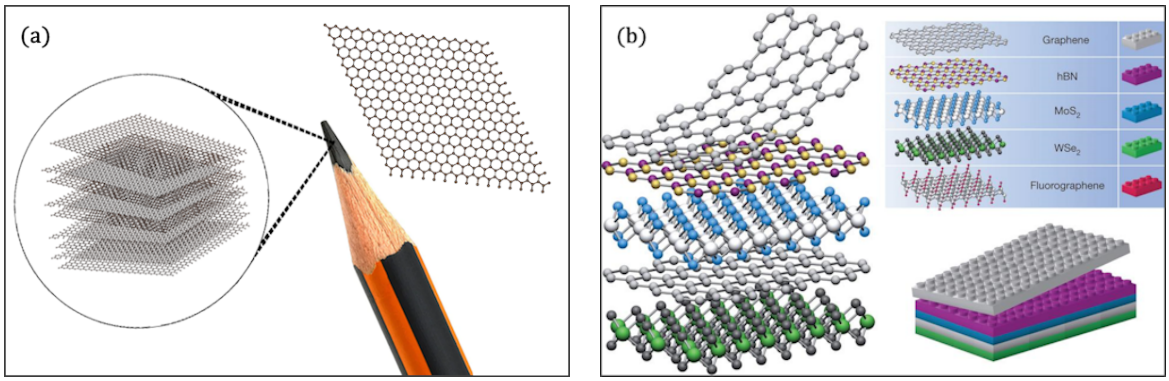


Figura 1.1: (a) Grafeno extraído del grafito usado en lápices. (b)Panel derecho: Láminas atómicas que pueden ser apiladas verticalmente donde cada esfera de diferente color representa una especie química diferente. Panel izquierdo: Analogía entre sistemas laminares atómicos con bloques de construcción de Lego y su correspondiente apilamiento vertical. Figura tomada del artículo "*Van der Waals heterostructures*"[4].

Mediante el apilamiento vertical de materiales 2D, es posible crear nuevas estructuras que, en relación con las propiedades físicas de los materiales individuales, pueden conservar dichas propiedades, mejorarlas o incluso exhibir características completamente nuevas no observadas en los materiales aislados. Además, los sistemas formados por capas de los mismos materiales pueden mostrar propiedades diferentes si presentan variaciones como el orden de apilamiento o distintas rotaciones en sus capas. Todo esto dota al apilamiento vertical de materiales 2D de un gran potencial para crear materiales con diversas propiedades físicas, lo que hace que estas estructuras generadas sean de gran interés para la Física y las Nanociencias, con posibles aplicaciones en Nanotecnología [5].

Ejemplos del uso de las heteroestructuras de van der Walls en la actualidad se observan en:

- La fabricación de semiconductores, lo que ha favorecido en la miniaturización de circuitos integrados, sustituyendo las conexiones alámbricas de berilio por capas de aluminio y mejorando su funcionalidad al emplear elementos compuestos por diferentes tipos de semiconductores [6, 7].
- Experimentos donde se busca limpiar aguas residuales mediante fotocatálisis utilizando su banda energética [8].
- La creación de fotodetectores de alta sensibilidad, capaces de operar en un amplio rango espectral, desde el visible hasta el infrarrojo [3].

- La creación de láseres ultradelgados y flexibles muy prometedores para aplicaciones en pantallas, comunicaciones ópticas y tecnologías de realidad aumentada [9].
- La fabricación de LEDs y celdas solares ultradelgadas aplicables en dispositivos portátiles y en tecnología flexible, una tendencia creciente en la industria [10].

Como se mencionó antes, el grosor de los materiales 2D utilizados está en el orden de angstroms (Å)[4] y sus interacciones físicas ocurren a escala atómica, por lo que el estudio de estos sistemas recae en el área de la Física del Estado Sólido. Esta área de la física tiene como objetivo principal el estudio de las propiedades físicas de estructuras cristalinas, basándose en el hecho de que dichas estructuras presentan una periodicidad en el espacio real, ya sea en una (1D), dos (2D) o tres dimensiones (3D)[11] (figura 1.2). Basados en su periodicidad, estas estructuras cristalinas, conocidas simplemente como *cristales*, pueden describirse mediante una **celda primitiva**, que se compone de dos elementos fundamentales: *una red de Bravais* y una *base atómica.*

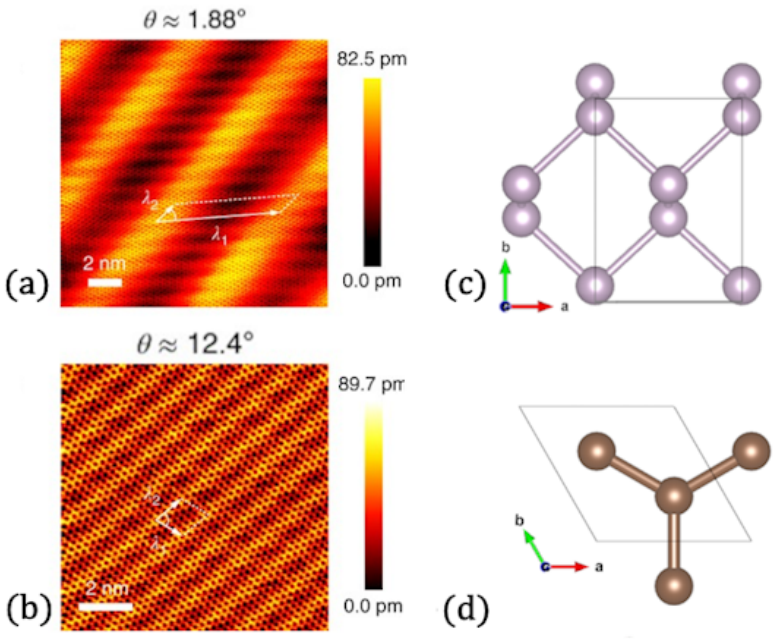


Figura 1.2: Imágenes tomadas con microscopio de efecto túnel (STM por sus siglas en inglés) de heteroestructuras formadas por dos capas de fosforeno negro y una de grafeno con una rotación de (a)1.88^o y (b)12.4^o donde se puede apreciar la periodicidad que presentan estas [12]. Representación gráfica de las celdas unitarias del (c)fosforeno negro y (d)grafeno.

La red de Bravais representa la repetición traslacional de los átomos en el espacio real mientras que la base atómica indica la disposición específica de los átomos dentro de la

[4] $1\text{Å} = 0.1\,nm = 1 \cdot 10^{-10}\,m$

celda (figura 1.3). La naturaleza cuántica de estos sistemas y sus características a escala atómica hacen que la determinación y comprensión de sus celdas primitivas sean de vital importancia para el estudio de sus propiedades físicas y químicas como son la estructura electrónica y reactividad química.

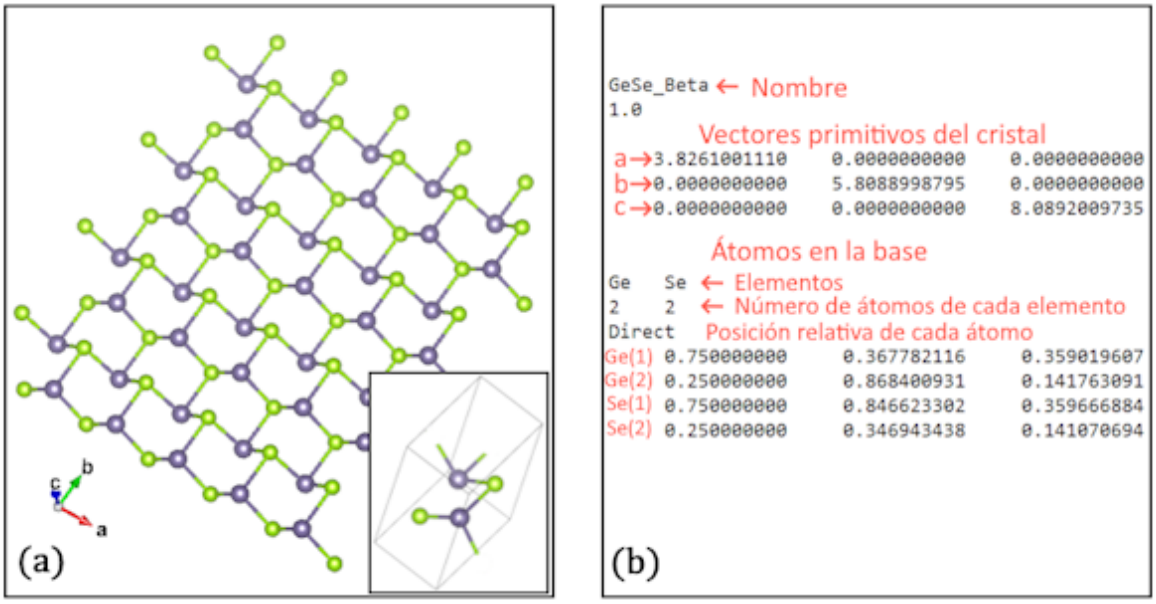


Figura 1.3: (a) Lámina de Germanio-Selenio fase Beta (*β-GeSe*) y su celda primitiva. (b) Imagen de un archivo en formato *POSCAR* donde se observan los vectores que generan la Red de Bravais y la base atómica del *β-GeSe*

En la actualidad, para el estudio teórico de estos sistemas, estructuras compuestas de dos o más materiales 2D, se utilizan herramientas computacionales como la *Teoría de los Funcionales de la Densidad* (DFT, por sus siglas en inglés)[13]. En estos casos es necesario identificar y caracterizar el *"bloque de construcción"* más pequeño del sistema, la celda primitiva. Una vez hecho lo anterior si se pretende estudiar al sistema perturbado es necesario construir otro sistema periódico más grande, a partir de la celda primitiva, que es conocido en el contexto de Estructura Electrónica y Estado Sólido como *supercelda*.

La obtención de estas superceldas no es un problema trivial a diferencia de lo que hace parecer la visualización de estos sistemas con piezas de LEGO apiladas en la figura 1.1(b), dado que la celda primitiva de cada una de las capas del sistema puede tener diferente forma, tamaño y orientación. Así, determinar una supercelda (común) que tenga la misma periodicidad en todas las láminas atómicas no es sencillo y su tamaño llega a ser muy grande, con varios miles de átomos en esta. Es importante mencionar que, en ciertas condiciones, la existencia de una supercelda común en un sistema puede no lograrse si

no se aplica en este un efecto de tensión sobre las capas que lo conforman, obteniendo así un nuevo sistema aproximado al original, el cual sí presenta una supercelda que mantenga la periodicidad de todas las capas en todo el espacio real como en la figura 1.4.

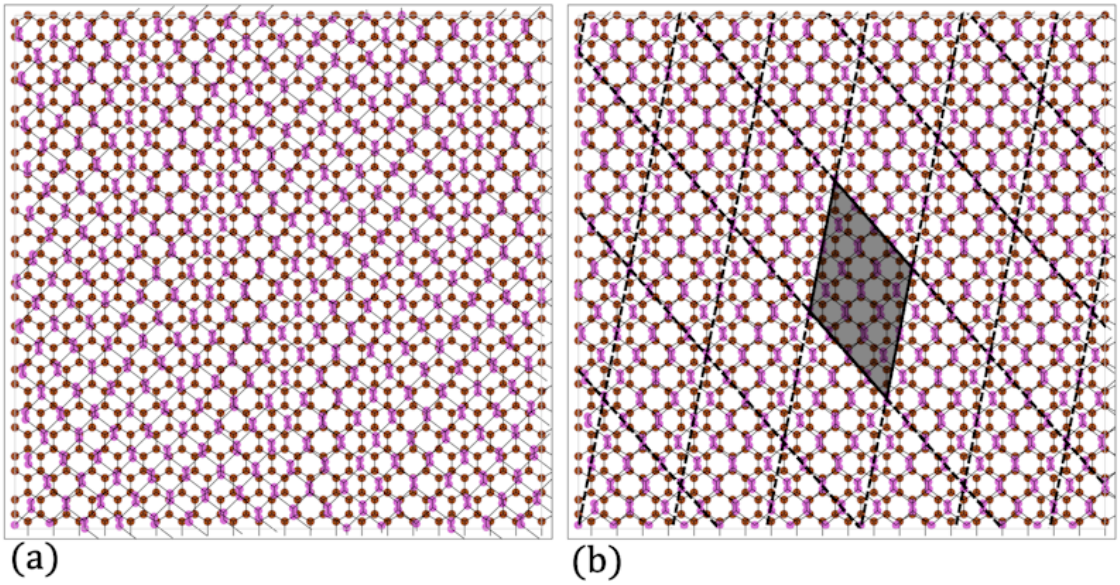


Figura 1.4: Sistema bicapa de grafeno-fosforeno negro con un ángulo relativo entre capas de 3.5^o. En (a) se muestra el sistema sin alteración en las capas y no se encuentra una supercelda que pueda mantener la periodicidad completa dentro del área observada, sin embargo en (b) se muestra el sistema optimizado, donde la capa de fosforeno negro sufrió una deformación aumentando el tamaño de sus vectores primitivos en un $-1.94\,\%$ y $+1.90\,\%$ respectivamente, en este sí existe una supercelda que mantiene la periodicidad en todo el sistema.

Por todo lo anterior, la obtención de una celda mínima para una red cristalina que pueda describir estructuras formadas por dos o más capas de materiales 2D, es el primer gran problema a resolver antes de poder estudiar las propiedades físicas que puedan presentarnos dichas estructuras. Existen implementaciones publicadas actualmente que atacan este problema, tres de estas son los códigos *CellMatch* [14], *Supercell-core software* [15] y *Twister* [16].

Sin embargo, pese a que el problema se ha pensado solucionar con estas herramientas, aún nos topamos con algunas complicaciones, por un lado, en la implementación *CellMatch* se hace uso de fuerza bruta para encontrar su respuesta, lo que afecta su eficiencia computacional, además de que esta implementación solo trabaja con estructuras bicapa[5].

[5]Al iniciar este trabajo la página señalada por el artículo, esta era un "enlace roto", por lo que no era posible acceder a su código, impidiendo su uso o revisión. Sin embargo en una revisión posterior el enlace ya era válido, permitiendo su análisis.

Por otra parte, en la implementación de *Twister*, el principal problema a resolver es la obtención de ángulos de rotación óptimos, exclusivamente para sistemas de 2 capas, limitando el tipo de sistemas que se pueden resolver con este y dejando en segundo plano la obtención de la supercelda para el sistema, así que para esto, dicho código utiliza un algoritmo de *"fuerza bruta"*, que conlleva a tener una solución con complejidad en $O(n^4)$, teniendo como n el límite en el espacio de búsqueda de la supercelda.

Por último, en *Supercell-Core* se tiene un programa más completo y robusto que los dos anteriores; sin embargo, mientras que sus resultados en sistemas específicos son buenos, en otros sistemas (sistemas más complejos que presentan superceldas muy grandes) aún muestra problemas, teniendo resultados con átomos faltantes o extras en su base atómica, esto posiblemente por problemas en el redondeo a la hora de efectuar las operaciones o por los métodos de deformación empleados, estos problemas no son fácilmente detectables en sistemas muy grandes lo que puede llevar a errores.

Por lo tanto, el objetivo general de este proyecto como una Tesis de Licenciatura, es desarrollar un código computacional abierto (*open source*)[17], en lenguaje Python[18], que sea flexible, robusto e intuitivo para el usuario. Este código tiene como fundamentos el uso de métodos geométricos[19, 20] para la determinación de una supercelda 2D (pequeña) conmensurable para sistemas compuestos por dos o más láminas atómicas con diferentes orientaciones relativas y quiralidad, asegurando que presente una tensión acotada en cada capa.

La información resultante del código incluirá los vectores de la red en su representación matricial, así como las posiciones de los átomos en coordenadas relativas a los vectores de la red, con la posibilidad de exportar dichos resultado a un archivo POSCAR [6]. Esta información puede ser utilizada en estudios de primeros principios para caracterizar la estructura electrónica del sistema o para posibles optimizaciones estructurales mediante el uso de herramientas de dinámica molecular con recursos de supercómputo.

Con este código, además de abordar las limitaciones observadas en trabajos previos, también se proporcionará información sobre el espacio recíproco correspondiente a los sistemas calculados, información que no ha sido considerada en los resultados arrojados por ninguna de las implementaciones mencionadas anteriormente.

Conocer la imagen de un cristal en el espacio recíproco es fundamental en la física del estado sólido y en la teoría de estructura electrónica de materiales a escala atómica. El

[6]Un archivo POSCAR es un formato estándar para describir la estructura cristalina de un material en términos de los vectores de red y las posiciones de los átomos en una celda unitaria.

espacio recíproco es crucial para comprender la dispersión de las ondas y los fenómenos de difracción, como los observados en la difracción de rayos X y la microscopía electrónica de transmisión de alta resolución (HRTEM, por su siglas en ingles). Estos fenómenos permiten obtener información detallada sobre la estructura cristalina y la disposición de los átomos en un material.

Además, la imagen en el espacio recíproco proporciona información sobre la periodicidad y la simetría de la red cristalina, información esencial para comprender las propiedades físicas y electrónicas de los materiales. Por ejemplo, los picos en un patrón de difracción de rayos X o en una imagen de HRTEM corresponden a vectores de red recíproca que están relacionados con la estructura cristalina del material. Estos picos pueden proporcionar información sobre la estructura cristalina, la orientación de los cristales y las deformaciones en la red.

En resumen, esta información adicional que será proporcionada por el programa es esencial para caracterizar la estructura y las propiedades de los materiales a nivel atómico, lo que tiene implicaciones significativas en áreas como la ciencia de materiales, la nanotecnología y la electrónica.

El enfoque interdisciplinario de esta Tesis combina conceptos de física de materiales, cristalografía, programación y simulación computacional, lo que permitirá abordar sistemas complejos y utilizar métodos avanzados de análisis, además de que el que el código sea abierto es especialmente importante, ya que permitirá que otros investigadores utilicen y contribuyan al desarrollo del software en sus propios proyectos científicos.

Estructura general del programa

El programa que aquí se presenta, está formado por 6 archivos de python, *Basics.py, Atom.py, Lattice.py, Functions.py, System.py* e *Interface.py*. Estos archivos son importados cada uno por el siguiente archivo, haciendo una especie de encapsulado (figura 2.1). De tal manera que las funciones descritas en cada uno de ellos son ocupadas por los siguientes archivos, manteniendo un orden y jerarquía, permitiendo al usuario interactuar solamente con la capa más externa, Interface.py, donde se guía al usuario paso a paso desde la consola para crear y analizar un sistema[1]. A continuación se describirá, brevemente, cada uno de los archivos antes señalados[2].

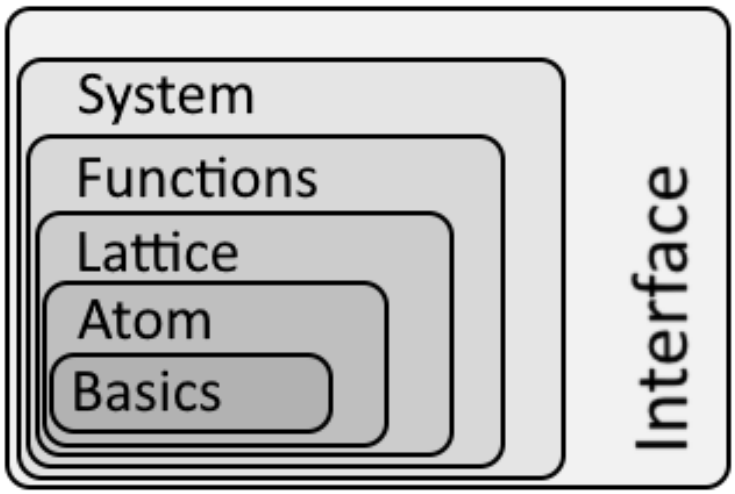


Figura 2.1: Representación esquemática de la estructura general del programa. Esta figura muestra como cada nivel encapsula al anterior hasta llegar a la interface del usuario.

[1]En caso de que el usuario quiera usar las herramientas del programa de forma directa y manual basta con importar el archivo System.py para tenerlas todas esas herramientas a disposición

[2]Todas las funciones pertenecientes a cada archivo se tratan con profundidad en el apéndice A.

2.1. Basics.py

Este archivo agrupa funciones esenciales que constituyen los cimientos del programa. Las funciones pertenecientes a este bloque pueden clasificarse en tres grupos: funciones que operen sobre vectores, funciones que operan sobre matrices y funciones misceláneas. Estas últimas son funciones usadas por algunos métodos que resuelven problemas específicos. Ejemplos de estas funciones son las utilizadas para encontrar los puntos de red de vecinos más cercanos a uno específico; o la función que calcula los vértices de una Celda de Wigner-Seitz.

El propósito de escribir manualmente estas funciones básicas aquí es para tener una independencia de bibliotecas externas a la biblioteca estándar de Python, proporcionando portabilidad al programa y eliminando la necesidad de que los usuarios instalen bibliotecas de Python de terceros. Además de asegurar un control más detallado de las operaciones internas del programa.

2.2. Atom.py

En este archivo se describe la clase “Atom”, clase que sirve de molde para crear los objetos ocupados en este programa para modelar los átomos que conforman la base atómica de los cristales.

Esta clase es muy básica y maneja solamente la información mínima de los átomos en un cristal requerida por el programa, lo que, aunque pasa por alto diversas características en el modelado, es suficiente para los requerimientos del programa.

La robustez de la clase “Atom” radica en su capacidad para representar y clasificar átomos pese a su simplicidad, proporcionando una estructura coherente y organizada para construir estructuras más complejas.

2.3. Lattice.py

Esta clase es usada para crear objetos que se erigen como la representación integral de los cristales en este programa. Son definidos por sus vectores primitivos de red y la base atómica proporcionada por una lista de objetos “Atom”.

Además de la inicialización del objeto “Lattice”, esta clase alberga funciones que operan directamente sobre él. Estas funciones desempeñan un papel crucial en la mani-

pulación y análisis de los cristales.

En esta clase se describen desde funciones para modificar el objeto mismo, hasta funciones con el propósito de exportar la información del cristal hacia un archivo POSCAR o desplegar una representación del cristal en pantalla.

2.4. Functions.py

El archivo functions.py contiene diversas funciones que trabajan mediante la interacción de objetos tipo "Lattice" principalmente. La principal función de las funciones en este archivo es la de encontrar superceldas de dimensiones similares comunes en todas los objetos de esta lista y conocer el ajuste que estos objetos requieren para lograr la coincidencia sin ser deformados más allá de un valor límite específico.

Además de las funciones anteriores, en este archivo también se tienen funciones que crean de forma rápida objetos tipo "Lattice" hexagonales y rectangulares, donde solo se dan como parámetros de entrada el tamaño de el o los periodos (en Å) de la red.

También se cuenta con funciones que inicializan "Lattices" específicas para algunos materiales recurrentes como por ejemplo: *grafeno*, *fosforeno negro*, y las fases α y β del compuesto GeSe, por simplicidad α-GeSe y β-GeSe.

2.5. System.py

En este archivo se describe la clase "System", en esta se crean objetos los cuales actúan como el modelo para las heteroestructuras con las que se está trabajando. La base fundamental de estos objetos es una lista con las "Lattices" correspondientes a los materiales que apilados verticalmente conforman a la heteroestructura por modelar.

Las funciones de esta clase se encargan de operar sobre el objeto System, encontrando posibles celdas primitivas para este, analizando los resultados obtenidos o se encargan de presentar estos resultados al usuario de forma que el usuario haga uso de lo que más le convenga dado el problema que quiere resolver.

2.6. Interface.py

Este archivo viene siendo la capa más externa del programa. Es la que interactúa directamente con el usuario dándole acceso a la creación y manipulación de los objetos "System", para modelar las estructuras cuya celda primitiva se desea conocer.

Algoritmo principal

En el análisis del software presentado, se modelará a la estructura bajo estudio como S, un objeto de clase *"System"* definido por una lista de objetos de tipo *"Lattice"*.[1] Estos objetos son la representación de las redes periódicas 2D asociadas a las estructuras que son apiladas para formar la heteroestructura. Una vez obtenida la celda primitiva para S, se define un nuevo objeto *Lattice* llamado **SuperRed** asociado a S ($S.SuperRed$), de tal forma que S pueda ser visto como una red periódica 2D.

Internamente, el programa identificará al primer elemento de la lista de redes que definen a S ($S.Layers[0]$) como la **capa sustrato**, esta es la capa sobre la cual se realizan los principales análisis dado que esta capa no tendrá deformaciones al momento de resolver el problema.

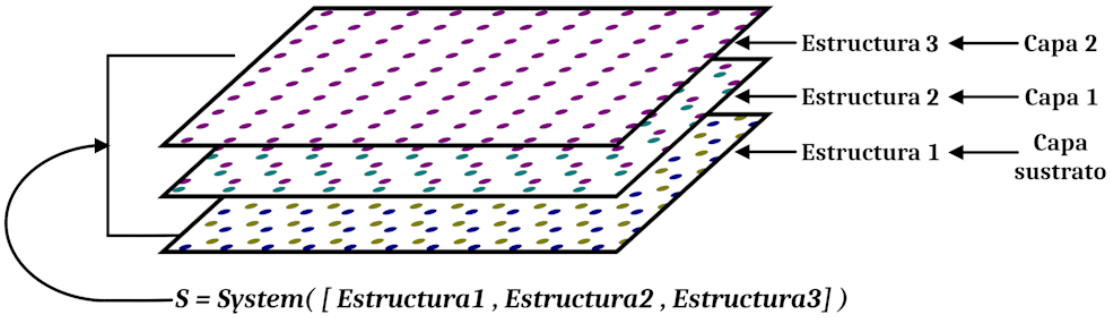


Figura 3.1: Representación general sobre cómo el software abstrae el sistema a partir de la lista de estructuras iniciales.

Fue mencionado anteriormente que, si encontramos superceldas para las redes en cada una de las capas de un sistema que coinciden al proyectarlas sobre el plano, entonces tenemos una posible celda primitiva para el sistema (ver figura 3.2). De manera inversa, una celda que cumple con ser una celda primitiva para el sistema S coincidirá con alguna supercelda de cada una de las estructuras que conforman dicho sistema. Además, los vectores que describen estas superceldas serán vectores de traslación válidos para estas

[1] Ambas clases, *System* y *Lattice*, se verán a detalle más adelante en este capítulo.

redes.

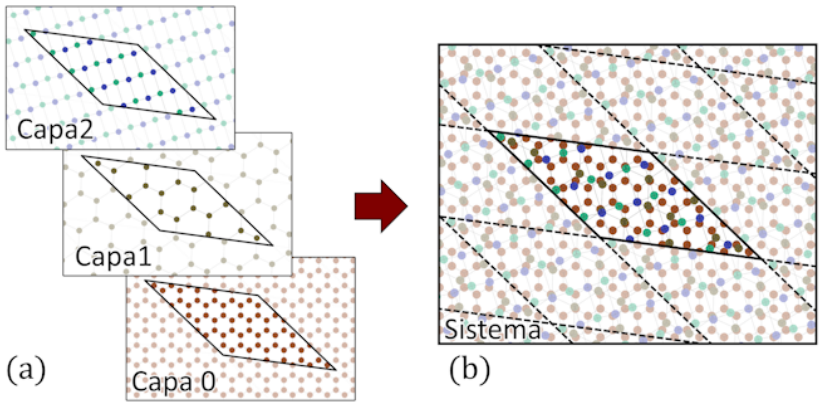


Figura 3.2: La figura en (a) muestra superceldas de 3 redes cristalinas distintas pertenecientes a las capas de un sistema apilado, estas superceldas tienen la cualidad de ser idénticas al proyectarse sobre el plano. En la figura (b) se presentan una celda primitiva para la heteroestructura, esta celda coincide con el apilamiento de las superceldas en (a).

Esta propiedad muestra que los vectores primitivos del sistema coinciden con algún vector de traslación de las redes en cada capa. Por lo tanto, es posible relacionar las celdas primitivas del sistema con una matriz de transformación (MT) para alguna capa del sistema. En el programa esta identificación se hará con la correspondiente MT de la capa sustrato del sistema, esto ya que, como se mencionó, esa capa no será modificada durante todo el proceso.

En estructuras 2D, una MT es una matriz 2×2 formada por los coeficientes enteros de los vectores de traslación que describen la supercelda ligada a la MT (un ejemplo se ve en la figura 3.3). Esta MT se construye de la siguiente forma: Sean $\mathbf{a}$ y $\mathbf{b}$ los vectores primitivos de una red cristalina 2D, L, entonces la MT correspondiente a la supercelda formada por los vectores de traslación $\mathbf{s}_a$=($m\mathbf{a}$+$n\mathbf{b}$) y $\mathbf{s}_b$=($p\mathbf{a}$+$q\mathbf{b}$) es

$$T = \begin{pmatrix} m & p \\ n & q \end{pmatrix}$$

.

Dado lo anterior, lo primero a hacer para resolver el problema principal es encontrar los VTs comunes en todas las capas, que serán posibles vectores primitivos del sistema. Para llevar a cabo la búsqueda de estos vectores, el programa necesita parámetros adicionales. En primer lugar, es necesario delimitar el espacio de búsqueda, lo cual se logra mediante un valor entero n, este valor define un espacio de búsqueda, el cual corresponderá al paralelogramo formado por la supercelda de la capa sustrato de S, desde $-n$

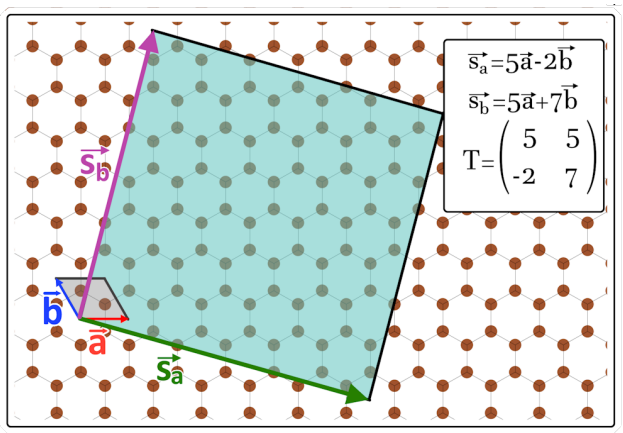


Figura 3.3: La MT, T, hace referencia a la supercelda cuadrada definida por los vectores de traslación $\mathbf{s}_a$ y $\mathbf{s}_b$, en una red hexagonal que tiene como vectores primitivos, **a** y **b**.

hasta n, en ambos vectores primitivos **a** y **b** como se representa en la figura 3.4.

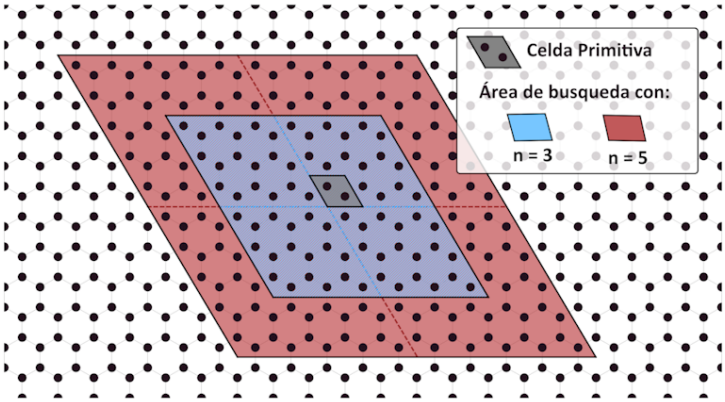


Figura 3.4: En rojo y azul las áreas de búsqueda delimitadas por $n = [-3,3]$ y $n = [-5,5]$ para una red hexagonal. Se puede observar que el área de búsqueda crece de forma cuadrática con respecto al valor de n que la genera.

El segundo parámetro a considerar es una constante ε que representa un límite máximo en el error permitido entre los vectores de traslación similares en todas las capas. Un menor valor de ε permite resultados que requieren una menor deformación mecánica (compresión o estiramiento) de las redes para lograr la periodicidad, sin embargo puede requerir una mayor área de búsqueda para tener resultados.

En el algoritmo, los posibles vectores primitivos para S se definen mediante una lista de puntos de red (PRs), R, esta lista se obtiene a partir de una lista de los PRs de la capa sustrato ubicados dentro del área de búsqueda especificada. Esta lista es "limpiada" dejando solo aquellos PRs que presenten un error menor o igual al ε dado con los PRs

correspondientes a cada capa del sistema. Obteniéndose así una lista con solo los PRs que se repiten en todas las capas del sistema con un error acotado.[2]

Posteriormente, a partir de la lista R, se generan las posibles MTs que darán forma a la supercelda de la capa sustrato a partir de la cual se definirá la celda primitiva para S. Estas matrices se construyen seleccionando parejas de vectores de traslación de R, asegurándose que estos vectores generen MT válidas.

Para que una matriz sea considerada una MT válida para una red debe de cumplir que los vectores resultantes de esta sean linealmente independientes, para lo que basta con que su determinante sea diferente a cero.

De las matrices calculadas, se crea la lista $LoMat$ de S, $S.LoMat$, en la cual se almacenan aquellas MTs correspondientes a las superceldas más pequeñas. En esta etapa, el usuario tiene la opción de decidir manualmente una MT, T, de $LoMat$ o permitir que el programa utilice la que ha calculado como la mejor opción.

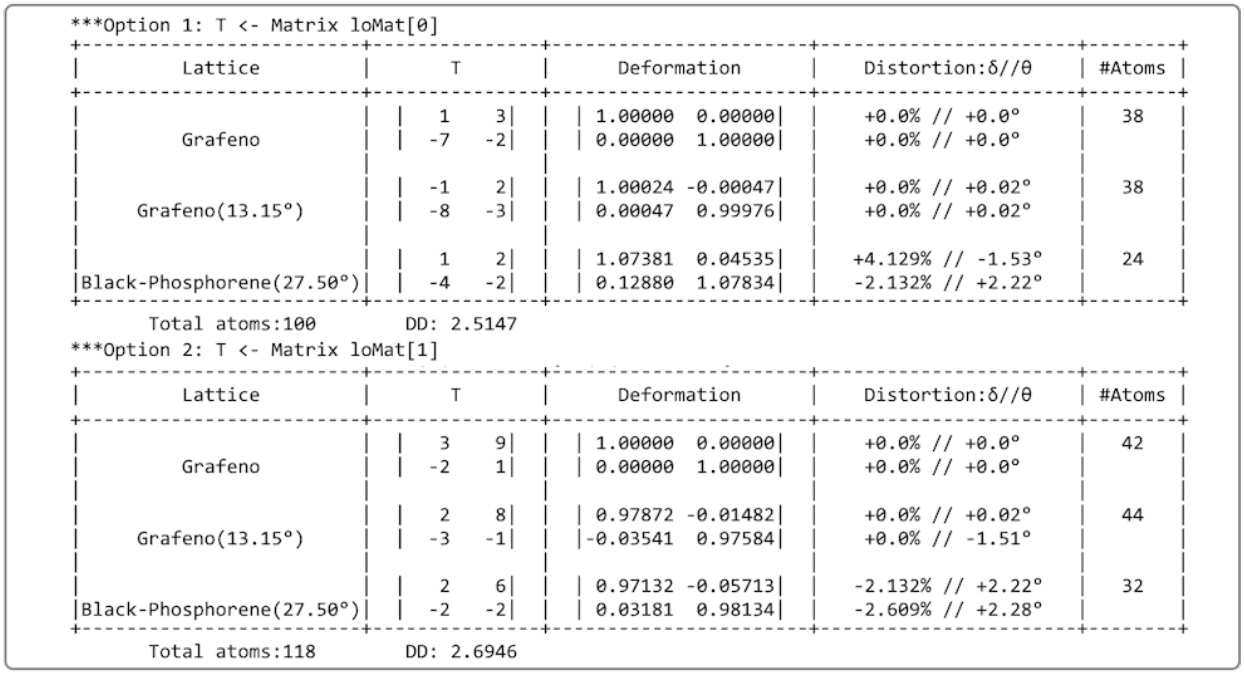
Para poder elegir la MT más conveniente para el usuario se puede hacer uso del método `ShowTMs` de la clase *System*, la cual despliega en pantalla tablas con columnas que muestran las características que tendría la celda primitiva generada para cada MT en $LoMat$. Estos datos son:

- **Lattice:** Nombre de la red perteneciente a la capa.
- **T:** MT que construye la supercelda correspondiente a la celda primitiva en dicha capa.
- **Deformation:** Matriz de deformación que requiere la red para mantener la periodicidad.
- **Distortion δ // θ:** Cambio sufrido por los vectores primitivos de la red tras la deformación, donde δ es el cambio en su magnitud y θ el cambio en su dirección.
- **#Atoms:** Número de átomos contenidos en cada capa.

Para finalizar se muestra el número total de átomos en la celda primitiva asociada a cada MT y una calificación DD (Degree of distortion); la cual está relacionada al porcentaje de distorsión total que debe sufrir el sistema para ser periódico con esta celda primitiva[3].

[2] El algoritmo empleado aquí será descrito con detalle más adelante.

[3] La matriz de deformación, los cambios δ y θ y la DD se describirán más adelante en este capítulo.



***Option 1: T <- Matrix loMat[0]

Lattice	T	Deformation	Distortion:δ//θ	#Atoms
Grafeno	1 3 / -7 -2	1.00000 0.00000 / 0.00000 1.00000	+0.0% // +0.0°	38
Grafeno(13.15°)	-1 2 / -8 -3	1.00024 -0.00047 / 0.00047 0.99976	+0.0% // +0.02°	38
Black-Phosphorene(27.50°)	1 2 / -4 -2	1.07381 0.04535 / 0.12880 1.07834	+4.129% // -1.53° / -2.132% // +2.22°	24

Total atoms:100 DD: 2.5147

***Option 2: T <- Matrix loMat[1]

Lattice	T	Deformation	Distortion:δ//θ	#Atoms
Grafeno	3 9 / -2 1	1.00000 0.00000 / 0.00000 1.00000	+0.0% // +0.0°	42
Grafeno(13.15°)	2 8 / -3 -1	0.97872 -0.01482 / -0.03541 0.97584	+0.0% // +0.02° / -1.51°	44
Black-Phosphorene(27.50°)	2 6 / -2 -2	0.97132 -0.05713 / 0.03181 0.98134	-2.132% // +2.22° / -2.609% // +2.28°	32

Total atoms:118 DD: 2.6946

Figura 3.5: Tablas correspondientes a MTs, $LoMat[0]$ y $loMat[1]$, calculadas para una heteroestructura formada por dos capas de grafeno y una de fosforeno negro. La capa de grafeno superior tiene una rotación con respecto a la inferior de 13.15^o, y la capa de Fosforeno está rotada 27.5^o con respecto a la misma capa de grafeno. En este caso, la MT recomendada es LoMat[0], esto es por su menor distorsión y tamaño, el cual es de solo 100 átomos.

Para finalizar, teniendo una MT adecuada T, se calcula la celda primitiva para S usando el método `S.createSuperLattice(T)`, el cual crea un nuevo objeto tipo *Lattice* para S llamado *Supercelda*. Esta corresponde a la red formada por la celda primitiva generada por T. El resultado puede ser presentado en pantalla con una proyección 2D en el espacio real de la celda primitiva calculada y una representación en el espacio recíproco del sistema S con el método `show`, ver figura 3.6.

Como acción extra, si se requiere, mediante la función `exportLattice` de la clase *System* se puede crear un archivo POSCAR[4] para la supercelda calculada, este archivo puede ocuparse como entrada para cálculos *ab initio* [21], o para ser visualizado con paquetes de cristalografía [22].

Al analizar la complejidad del algoritmo principal, observamos que está determinada principalmente por los algoritmos utilizados para generar la lista de vectores R y la lista de matrices *LoMat*. La complejidad de estos algoritmos depende directamente del núme-

[4]Este formato es utilizado como archivo de entrada, con información de la geometría y composición del sistema, en códigos computacionales de primeros principios que emplean la DFT como VASP (Vienna *ab initio* Simulation Package).

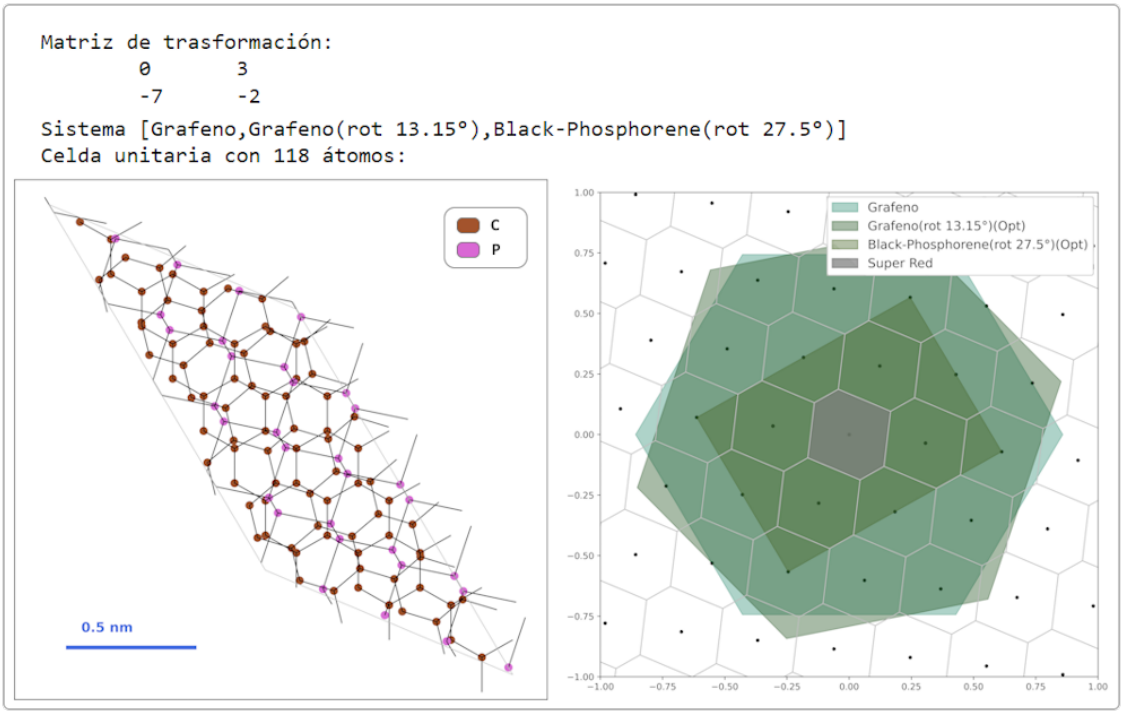


Figura 3.6: Imágenes desplegadas en pantalla al generar la supercelda del sistema compuesto por dos capas de grafeno y una de fósforo negro. El panel izquierdo muestra una representación 2D en el espacio real de la supercelda calculada. El panel derecho muestra la correspondiente representación en el espacio recíproco. Los puntos de la red, en el espacio recíproco son identificados por puntos negros que son encerrados por celdas hexagonales deformadas como aquella en morado centrada en el origen. Por otra parte, en este mismo espacio recíproco de la supercelda, es posible dibujar también las primeras zonas de Brillouin de cada celda primitiva: Dos hexagonales para grafeno, y una rectangular para fosforeno.

ro de puntos de red en la capa de sustrato dentro del espacio de búsqueda designado, con una complejidad de $O(n^2)$ para la generación de R y $O(n^4)$ para la generación de *LoMat*[5]. En consecuencia, la complejidad general del algoritmo está limitada por la creación de la lista *LoMat*, lo que implica un orden de complejidad de $O(n^4)$. Sin embargo, como se analizará más adelante, en la práctica la complejidad tiende a ser considerablemente inferior a este límite teórico.

[5]Estos resultados se verán en el análisis de dichos algoritmos en la siguiente sección de este mismo capítulo.

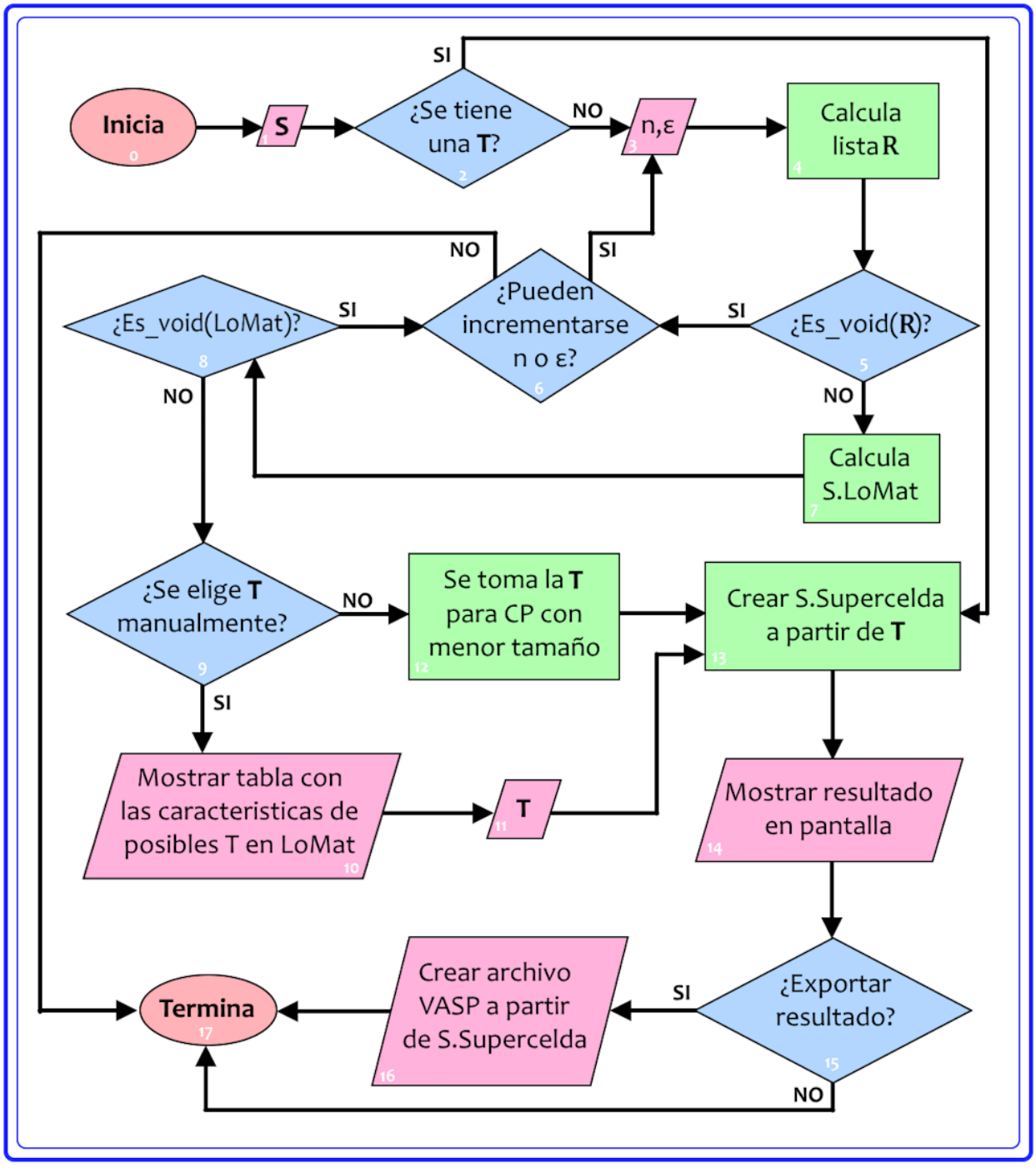


Figura 3.7: Diagrama de flujo del algoritmo general que sigue el programa para encontrar la supercelda.

El algoritmo principal del programa puede resumirse de la siguiente manera:

Algoritmo principal (3.1)

0. Inicia.
1. Crea S.
2. Si se cuenta con una matriz de transformación T se salta al paso 11, de lo contrario continúa.
3. Se define n y ε para la búsqueda.
4. Calcula la lista R con los vectores de traslación que satisfacen el límite de error ε para todas las capas de S.
5. Si R está vacía continúa, de lo contrario salta al paso 7.
6. Si se pueden dar nuevos valores mayores para n o ε se actualizan y se regresa al paso 3. De lo contrario no se puede encontrar una celda primitiva para S en el área de búsqueda y se termina el algoritmo (salto a 17).
7. Calcula una lista *LoMat* de S.
8. Si *LoMat* está vacía se regresa al paso 6, de lo contrario se continúa.
9. Si se desea elegir una T manualmente a partir de las matrices en *LoMat* se continúa, caso contrario se salta al paso 12.
10. Se despliega en pantalla una tabla con los datos de las matrices de transformación calculadas en LoMat.
11. Ingresa la T y salta al paso 13.
12. Escoge la mejor T de las MT de LoMat.
13. Calcula la Lattice que funcionará como CP para S a partir de la matriz T obtenida.
14. Muestra en pantalla el resultado.
15. Si se desea exportar la supercelda calculada se continúa, de lo contrario el algoritmo finaliza.
16. Se crea un archivo POSCAR con los datos de la supercelda calculada.
17. Finaliza el algoritmo.

Métodos y algoritmos esenciales

En esta sección, se abordan los algoritmos centrales usados en la solución del problema, así como los métodos principales que los implementan. Estos algoritmos son aquellos que por su complejidad e importancia sobresalen entre todos los demás.

La explicación detallada de estos conjuntos de instrucciones proporcionará una comprensión profunda de su funcionalidad y la teoría subyacente que los sustenta. Este análisis tiene como objetivo no solo demostrar la robustez de dichos métodos, sino también arrojar luz sobre su aplicación práctica y su relevancia en la solución del problema para la investigación en materiales avanzados.

4.1. Búsqueda de candidatos a vectores primitivos

Siguiendo el algoritmo principal (algoritmo 3.1), si desconocemos una MT T que resuelva el problema, entonces, una vez determinados los valores n y ε, lo primero a hacer es calcular una lista de posibles vectores primitivos, R, para el sistema evaluado, S, estos vectores deben estar en el área de búsqueda delimitada por n y tener un desajuste menor que ε.

Para llevar a cabo esta tarea, se requieren dos cosas. La primera es un método que nos permita encontrar el VT $\mathbf{u}_L$ en una red L, que sea el más parecido al VT $\mathbf{u}$ en otra red. La segunda es una forma de medir la diferencia, desajuste o error entre el vector original y el aproximado. Con estas dos herramientas, podremos obtener la lista deseada.

4.1.1. Determinar $\mathbf{u}_L$ para un u.

El encargado de este trabajo en el programa es el método llamado `calcCD()`, localizado en el archivo *Functions.py*.

Para iniciar, en el problema se tienen 2 redes, S y L. En la red S el vector $\mathbf{u}$ es un vector de traslación válido y se requiere encontrar el vector $\mathbf{u}_L$, el cual cumple con ser el vector de traslación válido en L más parecido al vector $\mathbf{u}$. Para resolver este problema, lo primero es darle uso al concepto de puntos de red[1]. De esta manera se puede simplificar una red bidimensional a un conjunto de puntos en el plano colocados de forma periódica, lo que elimina mucho ruido y simplifica el problema.

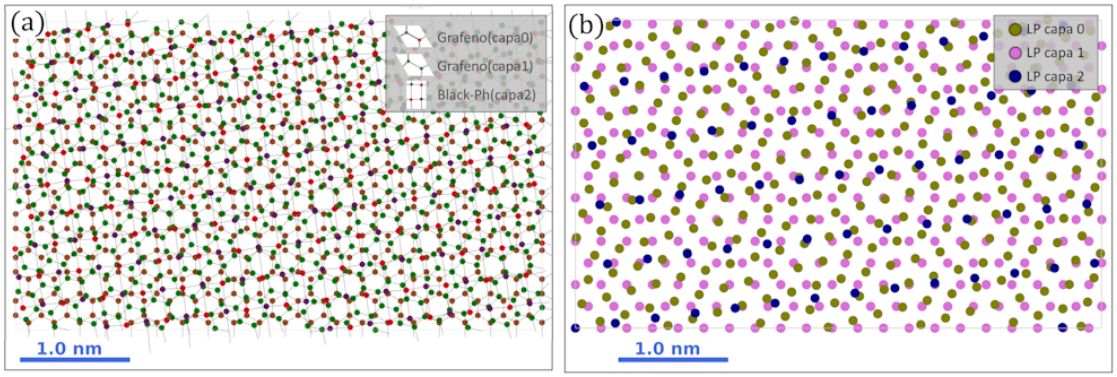


Figura 4.1: (a) Proyección en el plano de un sistema formado por dos capas de grafeno y una de fosforeno negro. La rotación entre la capa 0 y 1 es de 15.5^o, mientras que la rotación entre la 0 y 2 es de 7.3^o. (b) Representación en el plano de los PRs de las tres capas del sistema compuesto.

Sean los VPs de una red 2D $\mathbf{a}$ y $\mathbf{b}$, con un punto fijo en el plano $\mathbf{P}_0$, como un PR origen; entonces tomando en cuenta que, por periodicidad, todo PR corresponde a la traslación del punto $\mathbf{P}_0$ con un VT T y que:

$$T = i\mathbf{a} + j\mathbf{b}$$

entonces se puede relacionar cada PR (y al VT que la define), con un par de valores enteros.

$$\mathbf{P}_{i,j} = \mathbf{P}_0 + \mathbf{T}_{i,j} = \mathbf{P}_0 + i\mathbf{a} + j\mathbf{b} \quad \text{con } i, j \in \mathbb{Z} \tag{4.1}$$

Entonces, el sistema puede verse como un grupo de familias de puntos colocados de forma periódica en el plano (una por cada capa del sistema), donde todas las familias

[1]Puntos discretos en el espacio que definen la periodicidad de una red cristalina. Estos puntos corresponden a las posiciones donde las unidades de la estructura del cristal (como átomos, iones o moléculas) se repiten de manera idéntica, según un conjunto de vectores de traslación definidos por los parámetros de la celda unitaria.

contienen al punto $\mathbf{P}_0$. Así el problema de encontrar un VT común entre 2 redes de un sistema, se traduce a encontrar un punto diferente a $\mathbf{P}_0$ compartido en sus familias correspondientes.

Dado que los VPs de una red U en 2D deben ser linealmente independientes, estos pueden ser usados como una base β (en este caso para $\mathbb{R}^2$). Entonces puede observarse que si se usa β como una base para el plano, en cada posición $(x, y)_{[\beta]}$, con $x, y \in \mathbb{Z}$, se obtiene el PR, $\mathbf{P}_{x,y}$, de la red U. Explotando el fenómeno anterior se planea lo siguiente. Se define la notación $\mathbf{T}_{R,i,j}$ como el vector de traslación de la red R cuyas componentes son los enteros i y j, además sean $\mathbf{P}_{i,j}$ los PRs correspondientes a S y $Q_{k,l}$ los PRs correspondientes a L.

Si se tiene un vector $\mathbf{u}$=$\mathbf{T}_{S,a,b}$ es posible ubicar su correspondiente PR $\mathbf{P}_{a,b}$ en el plano. Suponiendo que $\mathbf{Q}_{c,d}$ es el PR de L más cercano a $\mathbf{P}_{a,b}$, entonces al hacer un cambio de base usando β (la base formada por los VPs de L), se consigue una nueva posición para $\mathbf{P}_{a,b}$, aquí $\mathbf{P}_{a,b}$ seguirá siendo cercano a $\mathbf{Q}_{c,d}$ (dado que este cambio de base no es siempre una transformación *isométrica* esto no se puede generalizar, sin embargo, en vecindades muy pequeñas se puede suponer que se cumple).

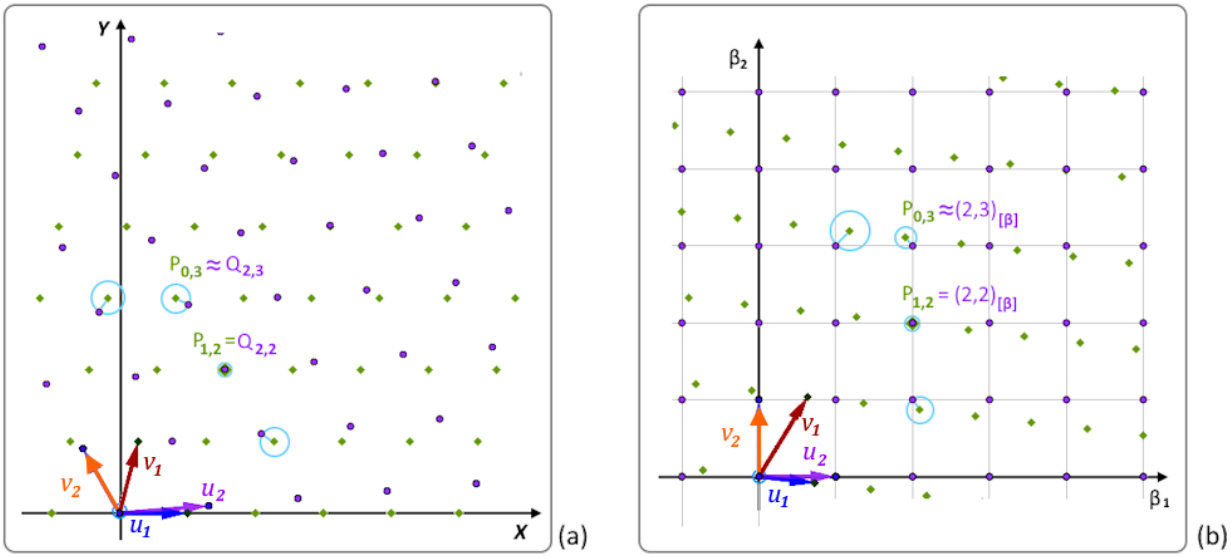


Figura 4.2: (a) Proyecciones en el plano de los PRs para 2 redes, una con VPs $\mathbf{u}_1, \mathbf{v}_1$ (rombos verdes) y otra con VPs $\mathbf{u}_2, \mathbf{v}_2$ (círculos morados). Se muestra cómo, si en (a) la base canónica para el plano, un PR $\mathbf{P}_{a,b}$ es muy cercano a un PR $\mathbf{Q}_{c,d}$, entonces en (b) el plano con base $\beta = \{\mathbf{u}_2, \mathbf{v}_2\}$, se tiene que $\mathbf{P}_{a,b} \approx (c, d)$.

Entonces, como se dijo antes, la posición de $\mathbf{P}_{a,b}$ será cercana a la posición de $\mathbf{Q}_{c,d}$ en la base β; pero $\mathbf{Q}_{c,d}$, bajo la nueva base, está posicionado en unas coordenadas enteras. Para ser más preciso,en las coordenadas (c, d). Por lo que la nueva posición de $\mathbf{P}_{a,b} \approx (c, d)_{[\beta]}$,

como se ejemplifica en la figura 4.2. Así, la forma en que encontramos estos enteros c, d es obtener la posición de $P_{a,b}$ en base β, $(P'_x, P'_y)_{[\beta]}$, y redondear al par entero más próximo, teniendo $round\left((P'_x)_{[\beta]}\right) = c$ y $round\left((P'_y)_{[\beta]}\right) = d$.

De forma general, sean $[\mathbf{u} = (u_1, u_2), \mathbf{v} = (v_1, v_2)]$ y $[\mathbf{w} = (w_1, w_2), \mathbf{z} = (z_1, z_2)]$ los VPs de las redes S y L, respectivamente y sea $T_{L,c,d}$ el VT de L más parecido a $T_{S,a,b}$; tenemos que:

$$\begin{pmatrix} P_x \\ P_y \end{pmatrix} = \begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \quad y \quad \begin{pmatrix} w_1 & z_1 \\ w_2 & z_2 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \end{pmatrix}_{[\beta]} = \begin{pmatrix} P_x \\ P_y \end{pmatrix}$$

Entonces:

$$\begin{pmatrix} P'_x \\ P'_y \end{pmatrix}_{[\beta]} = \begin{pmatrix} w_1 & z_1 \\ w_2 & z_2 \end{pmatrix}^{-1} \begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

Por lo tanto:

$$\begin{pmatrix} P'_x \\ P'_y \end{pmatrix}_{[\beta]} = \begin{pmatrix} \frac{u_1 z_2 - u_2 z_1}{w_1 z_2 - z_1 w_2} & \frac{v_1 z_2 - v_2 z_1}{w_1 z_2 - z_1 w_2} \\ \\ \frac{u_2 w_1 - u_1 w_2}{w_1 z_2 - z_1 w_2} & \frac{v_2 w_1 - v_1 w_2}{w_1 z_2 - z_1 w_2} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \frac{a(u_1 z_2 - u_2 z_1) + b(v_1 z_2 - v_2 z_1)}{w_1 z_2 - z_1 w_2} \\ \\ \frac{a(u_2 w_1 - u_1 w_2) + b(v_2 w_1 - v_1 w_2)}{w_1 z_2 - z_1 w_2} \end{pmatrix}$$

Así se tiene que:

$$\begin{aligned} c = &\quad round\left(\left(a(u_1 z_2 - u_2 z_1) + b(v_1 z_2 - v_2 z_1)\right) / (w_1 z_2 - z_1 w_2)\right) \\ d = &\quad round\left(\left(a(u_2 w_1 - u_1 w_2) + b(v_2 w_1 - v_1 w_2)\right) / (w_1 z_2 - z_1 w_2)\right) \end{aligned} \tag{4.2}$$

Con las Ecs. (4.2) podemos definir la función "*calculaCD*", la cual al recibir 2 redes S, L y los coeficientes enteros a, b de $T_{S,a,b}$, regresará los coeficientes enteros c, d de $T_{L,c,d}$.

4.1.2. Cálculo del *error* para los vectores de traslación

Esta función desempeña un papel fundamental en la funcionalidad total del programa, ya que es usada como el primer filtro para llegar a un resultado final. El propósito principal radica en establecer un criterio de evaluación para cada vector candidato a VP, con el fin de discernir cuáles son las mejores opciones entre ellos.

Para poder elegir entre los candidatos a VP, es necesario definir una función, $err(\mathbf{u}, \mathbf{u}_L)$ que evalúe el error $\varepsilon_{\mathbf{u}_L}$ de cada VT $\mathbf{u}$ en la capa sustrato, con respecto a un vector $\mathbf{u}_L$, el cual corresponde al VT de la red L que es más parecido a $\mathbf{u}$. Esta función debe cumplir lo siguiente:

- El valor de $\varepsilon_{\mathbf{u}L}$ debe ser proporcional a la deformación requerida en la red en la capa L para mantener la periodicidad en toda la estructura.
- Para cada $\mathbf{u}$ debe existir un único valor $\varepsilon_{\mathbf{u}L}$ que lo califique.
- El valor $\varepsilon_{\mathbf{u}_L}$ debe ser siempre no negativo.
- Si $\varepsilon_{\mathbf{u}_L} = 0$, entonces el resultado es óptimo. Mientras mayor sea su valor, menos elegible es el vector $\mathbf{u}$ como VP.

Que el valor de $\varepsilon_{\mathbf{u}_L}$ sea no negativo y que una calificación cercana a cero sea el resultado óptimo, permite una fácil interpretación de la calificación de $\mathbf{u}$. Además de que junto a la relación entre $\varepsilon_{\mathbf{u}_L}$ y las deformaciones finales requeridas en el sistema permite la introducción de un límite ϵ. Este valor límite permite descartar aquellos vectores que no cumplen con los estándares requeridos, garantizando la selección de vectores óptimos para el proceso de determinación de la CP.

Lo primero que se requiere para calcular $\varepsilon_{\mathbf{u}_L}$ es conocer para el vector $\mathbf{u}$ su correspondiente vector $\mathbf{u}_L$, lo cual ya fue definido. Una vez conocido este vector se puede definir una forma de medir la diferencia entre ellos.

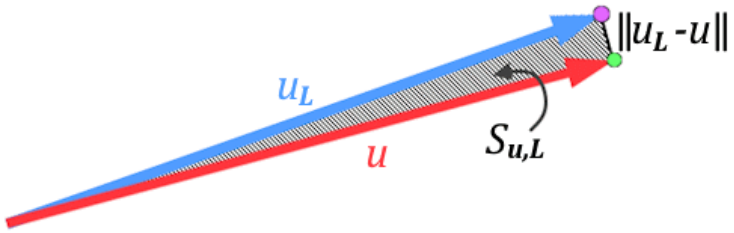


Figura 4.3: $S_{\mathbf{u}L}$ es la diferencia entre el vector $\mathbf{u}$ y su vector correspondiente en la red L, $\mathbf{u}_L$

.

Al conocer $\mathbf{u}_L$, se puede pensar que este se debe *deformar* hasta coincidir con $\mathbf{u}$, esta deformación nos permitiría medir su *error*. Así, definimos $S_{\mathbf{u}_L}$ como la suma de las distancias desde cada punto en $\mathbf{u}_L$ y su correspondiente punto en $\mathbf{u}$, ver la figura 4.3.

Con lo anterior podemos definir $err(\mathbf{u}, \mathbf{u}_L) = \varepsilon_{\mathbf{u}L}$ como el promedio de $S_{\mathbf{u}_L}$:

$$\varepsilon_{\mathbf{u}_L} = \frac{S_{uL}}{\|\mathbf{u}_L\|}. \tag{4.3}$$

Sean $\mathbf{u} = (u_x, u_y)$ y $\mathbf{u}_L = (v_x, v_y)$, entonces, para cada punto $\mathbf{v}_t = (t * v_x, t * v_y) \in \mathbf{u}_L$ corresponde el punto $\mathbf{u}_t = (t * u_x, t * u_y) \in \mathbf{u}$ con $t \in [0, 1]$. Así:

$$S_{\mathbf{u}_L} = \int_0^1 dist(\mathbf{v}_t, \mathbf{u}_t)\, dt$$

La función "*dist*" es una función que dados 2 vectores regresa la distancia entre ellos, esta es igual a la longitud del vector resultante de los vectores que recibió, osea, dados $\mathbf{a} = (a_1, a_2)$ y $\mathbf{b} = (b_1, b_2)$, entonces:

$$dist(\mathbf{a}, \mathbf{b}) = ||\mathbf{(b - a)}|| = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$

.

Por lo que tenemos que:

$$\begin{aligned} S_{\mathbf{u}L} &= \int_0^1 \sqrt{(t * u_x - t * v_x)^2 - (t * u_y - t * v_y)^2}\, dt \\ &= \int_0^1 \sqrt{t^2 \left((u_x - v_x)^2 - (u_y - v_y)^2\right)}\, dt \\ &= \int_0^1 t\sqrt{(u_x - v_x)^2 - (u_y - v_y)^2}\, dt \\ &= \int_0^1 t\, \|\mathbf{u} - \mathbf{u}_L\|\, dt \\ &= \|\mathbf{u} - \mathbf{u}_L\| \int_0^1 t\, dt \end{aligned}$$

Así:

$$S_{\mathbf{u}L} = \frac{\|\mathbf{u} - \mathbf{u}_L\|}{2} \tag{4.4}$$

Sustituyendo $S_{\mathbf{u}L}$ de la ecuación (4.4) en la ecuación (4.3), se obtiene:

$$err(\mathbf{u}, \mathbf{u}_L) = \varepsilon_{\mathbf{u}L} = \frac{\|\mathbf{u} - \mathbf{u}_L\|}{2\|\mathbf{u}_L\|} \tag{4.5}$$

De esta forma se obtiene $\varepsilon_{\mathbf{u}L}$, que será el error asociado al vector $\mathbf{u}$ con respecto a la red L. Este valor de error indicará la elegibilidad para este vector como VP del sistema estudiado.

La definición de $\varepsilon_{\mathbf{u}L}$ en la ecuación (4.5) cumple los requisitos arriba estipulados para el error que se busca:

- Por la forma en que se definió, $\varepsilon_{\mathbf{u}L}$ está relacionada a la deformación requerida en la red L para mantener la periodicidad del sistema.
- $\varepsilon_{\mathbf{u}L}$ tiene una solución única para cualquier vector valido para ser VP. Dado que un vector nulo no puede ser elegible como VP, entonces siempre se cumple que $\|\mathbf{u}\| \neq 0$, lo que evita indeterminaciones en $\varepsilon_{\mathbf{u}_L}$.
- El valor de $\varepsilon_{\mathbf{u}L}$ es siempre no negativo, dado que $\|\mathbf{u} - \mathbf{u}_L\| \geq 0$ y $\|\mathbf{u}\| > 0$.
- En el resultado óptimo $\mathbf{u} = \mathbf{u}_L$ y $\varepsilon_{\mathbf{u}L} = 0$. Entre mayor sea la diferencia entre $\mathbf{u}$ y $\mathbf{u}_L$ mayor será el valor de $\varepsilon_{\mathbf{u}L}$.

4.1.3. Algoritmo

Ya definidas las herramientas utilizadas por el el programa para resolver este primer paso, se tiene el siguiente algoritmo:

Creación de lista **R** (4.6)

0. Inicia para un sistema S.
1. Recibe n y ε.
2. Inicializa la lista R con todos los VTs de la *"capa sustrato"* ($S.redes[0]$) localizados dentro del área de búsqueda. Inicializa un contador i desde 1: $i = 1$.
3. Depura la lista R para la red L. Sea L la red en la capa i de S, entonces para cada vector $\mathbf{u} \in R$:
   - Identifica $\mathbf{u}_L$, el VT de L más parecido a $\mathbf{u}$.
   - Calcula el error entre $\mathbf{u}$ y $\mathbf{u}_L$, $err(\mathbf{u}, \mathbf{u}_L)$.
   - Si no se cumple que $err(\mathbf{u}, \mathbf{u}_L) < \varepsilon$, entonces elimina $\mathbf{u}$ de R.
4. Si $i \geq (len(S.redes) - 1)$, entonces salta al paso 6.
5. Incrementa el contador i en 1 y regresa al paso 3.
6. Regresa R como la lista de posibles VPs para S con los valores para n y ε dados.
7. Termina.

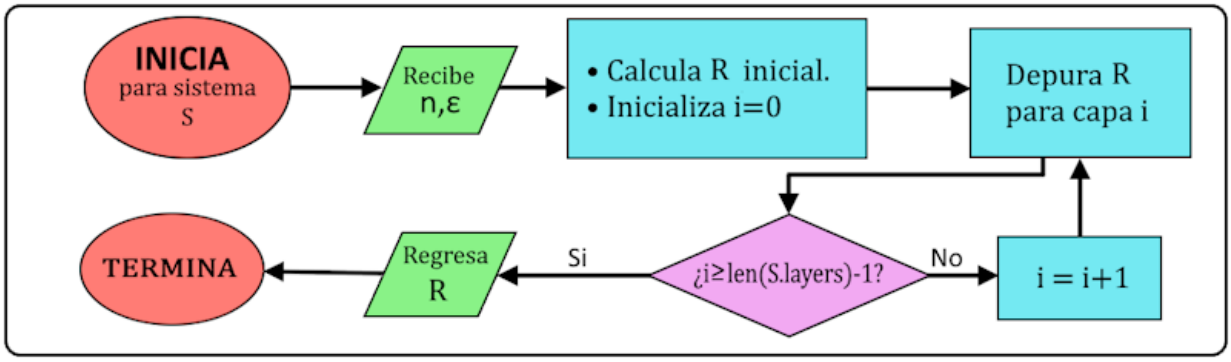


Figura 4.4: Diagrama de flujo para el algoritmo que calcula candidatos a VPs

En el programa, `searchLP(rangeOfSearch, epsilon)` es la instrucción encargada de hacer este cálculo, a *rangeOfSearch* le corresponde el valor de n y a *epsilon* el de ε.

Al finalizar el cálculo de la lista **R**, si **R** esta vacía significa que en toda el área de búsqueda no existe ningún **u** en los VTs de la capa sustrato del sistema. En este caso

los vectores correspondientes $\mathbf{u}_L$ en todas las demás capas, presentan un error menor al límite dado. Así que para obtener una lista *no vacía* se requiere aumentar el área de búsqueda incrementando el valor de n, o permitir un error mayor incrementando el valor de ε.

Al analizar la complejidad de este algoritmo, dado que el cálculo de los vectores u_L para los vectores u y el cálculo del error entre estos se obtienen respectivamente de las ecuaciones (4.2) y (4.5), su cálculo se obtiene en tiempo constante ($O(1)$). Así, es importante mencionar que el peso de la complejidad del algoritmo está determinado por el número de posibles vectores u en R y el número de capas del sistema.

El tamaño de la lista R está acotado por el total de puntos de red de la capa cero de S (capa sustrato), que se encuentran dentro del área de búsqueda. Este total es de $2n \times 2n = 4n^2$, por otro lado si tomamos el número de capas del sistema como c, este será una constante por lo regular pequeña (la mayoría de los sistemas evaluados serán de hasta 3 capas), así que podemos obviarla y evaluar este algoritmo como perteneciente al $O(n^2)$ donde n es referente a la variable que indica el área de búsqueda para este.

Por otro lado, al hacer un análisis más detallado del tamaño final de la lista R, podemos ver que esta solo puede llegar a ser igual a su cota en el caso de que no se descarte ninguno de los PRs iniciales, osea que para todos los PRs de la capa sustrato exista un PR en cada capa para el que su error calculado sea menor al ε dado. Esto solamente ocurre si el valor de ε es muy grande o si todas las redes que conformen el sistema tienen la misma orientación y la CP de la capa sustrato es de misma forma y tamaño que alguna supercelda de cada otra capa del sistema.

Lo anterior ocurre dado que en el paso del algoritmo en que se efectúa la comparación con la red de cada capa, el tamaño de la R inicial usualmente disminuye, ya que existen PRs de la capa sustrato para los que el error con su PR correspondiente a esa capa es mayor que ε, eliminando dichos puntos de la lista R para el análisis de la siguiente capa. El tamaño de esta disminución es mayor entre más distintas sean las redes en el sistema y menor sea el valor ε dado, de esta forma en sistemas con rotaciones y distintas redes de Bravais en cada capa o en análisis con ε muy pequeñas, el tamaño final de R será mucho menor a su cota n^2.

Con el análisis anterior también podemos concluir que el tamaño de la lista R final tiende a ser inversamente proporcional al número de capas del sistema.

4.2. Cálculo de matrices de transformación

El método en el programa encargado de llevar a cabo esta tarea es `calculateTM()` de la clase "System", el cual guarda la lista de matrices calculadas como la lista *loMat* de la clase System.

Las MTs indican una transformación lineal que, al aplicarse a los VPs de una red, generan los vectores que conforman una supercelda para dicha red. Debido a esto, el determinante de una MT nos indica cuántas veces es más grande la supercelda relacionada con ella en comparación con la celda primitiva de la red. Por lo tanto, al buscar las superceldas más pequeñas para utilizarlas como celdas primitivas (CP) del sistema, podemos calcular el determinante de las MTs para ordenar estos resultados y seleccionar las superceldas más pequeñas.

Dado que ya se tiene *R*, una lista de VT válidos en todas las capas del sistema, se puede generar MTs al tomar pares de vectores en *R*, sin embargo no todas los pares de vectores dan MTs que generen superceldas que puedan ser usadas como CP para el sistema. Por ejemplo si los vectores son colineales. Para descartar estos casos, también es útil calcular el determinante de la MT. Si el determinante es cero, entonces la MT es desechada y se continúa con otra.

Al momento de calcular el determinante también se puede forzar a que los vectores mantengan el mismo "*sentido*", así, si el determinante es negativo los vectores se intercambian, de manera que el nuevo determinante sea positivo, dejando resultados coherentes y de forma similar.

Por otro lado, aprovechando que los resultados obtenidos son los más "pequeños", se puede acotar el número de respuestas para limitar el costo computacional del cálculo. El límite es dado por la variable de la clase sistema llamada *MaxNumM*, la cual por defecto vale 10 pero puede ser cambiada por el usuario. Esta variable va a limitar el tamaño de la lista *loMat*, acotando el número de MTs que se mostrarán al usuario por respuestas.

Una vez creada la lista *loMat*, utilizando las Ecs. 4.2 podemos obtener las MTs para cada capa correspondiente a las MTs guardadas en ella.

El algoritmo usado para calcular esta lista de matrices es el siguiente:

Creación de lista de MTs *loMat* (4.7)

0. Inicia para un sistema S.
1. Recibe lista de vectores R y la variable *numOfM*
2. Inicializa $loMat$ con una la lista vacía. Inicializa contador $i = 0$.
3. Inicializa contador $j = i + 1$.
4. Si $j \geq |R|$, entonces salta al paso 10.
5. Crea la matriz $M_{i,j}$ a partir de los vectores $R[i]$ y $R[j]$.
6. $$biggestM = \begin{cases} \infty & \text{si } loMat \text{ es vacía} \\ max\{det(M) : M \in loMat\} & \text{si } loMat \text{ no es vacía} \end{cases}$$
   Si $det(M_{i,j}) \geq biggestM$ salta al paso 9.
7. Si $M_{i,j}$ es una MT válida y es nueva para $loMat$, la agrega de forma ordenada a la lista.
8. Si $len(loMat) >$*numOfM*, $loMat$ se corta hasta tener *numOfM* elementos.
9. Incrementa el contador j en 1, regresa al paso 4.
10. Incrementa el contador i en 1.
11. Si $i < |R|$ regresa al paso 3.
12. Regresa $loMat$ como la lista de las "*numOfM*" MT que generan las celdas periódicas más pequeñas para S.
13. Termina.

En el paso 7 del algoritmo se señala que se agrega la matriz $M_{i,j}$ calculada a la lista *loMat* si esta es una MT válida y si, además, es nueva para la lista. Lo primero es referente a lo señalado antes, si el determinante de la matriz es cero implicaría que los vectores utilizados son colineales, entonces la matriz se descarta ya que no hace referencia a una celda válida para la red. Lo segundo es un poco más complicado, pero igual de importante, ya que aquí se descartan matrices que sean una rotación exacta de alguna matriz que ya esté contemplada en *loMat*.

Otras matrices que se descartan en ese mismo paso son aquellas que generen celdas muy poco convencionales, como aquellas que tienen un ángulo muy cerrado entre los vectores que la conforman (por defecto se descartan matrices que generen celdas con

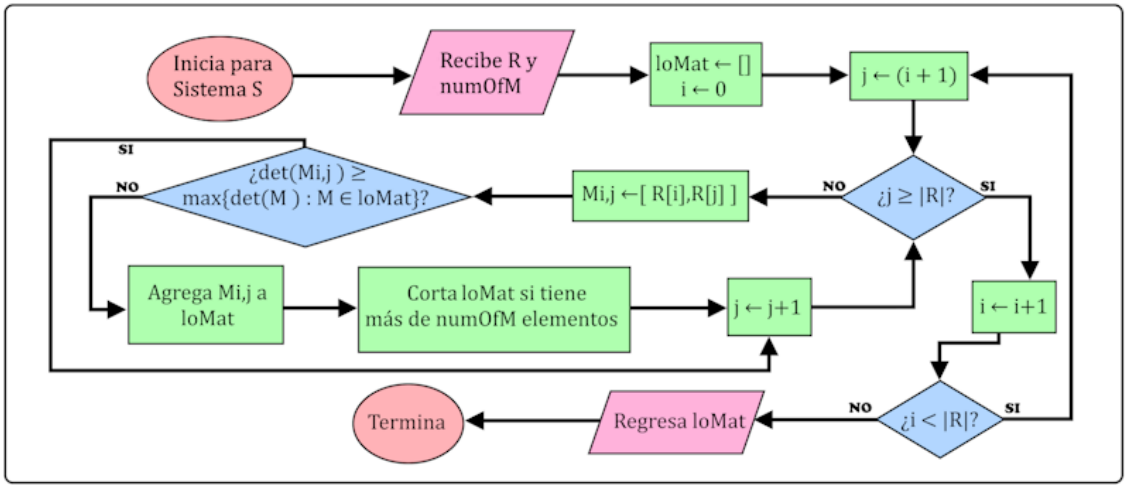


Figura 4.5: Diagrama de flujo para el algoritmo que calcula la lista loMat

menos de 20^o de ángulo interno), o que la proporción entre el tamaño de los vectores sea mayor a 10.

En el análisis de complejidad de este algoritmo se puede ver que el mayor peso de esta se lleva en el proceso de agregar las matrices de forma ordenada por el valor de su determinante, después de ser evaluadas, en la lista *loMat*. Entonces la complejidad de este algoritmo se acota solamente por el número de matrices evaluadas, que para una lista R de tamaño m se tienen un número de matrices distintas igual a $\frac{m(m-1)}{2}$. Esto es debido a que el proceso para mantener el orden de la lista resultante al agregar una nueva matriz a esta se acota por la constante *numOfM*.

Recordando que el tamaño de la lista R está acotado por el cuadrado de la constante n que determina el área de búsqueda para el algoritmo general, se puede acotar entonces este algoritmo en $O(n^4)$, aunque, como ya se señaló, en la ejecución normalmente el tamaño de la lista R es mucho menor a n^2, por lo que esta cota queda muy por encima de lo que se ve al ejecutar el código.

Otra medida de *optimización* se da en el caso de que todas las redes que los conforman al sistema evaluado sean redes hexagonales, en estos casos, dadas las simetrías de los sistemas, se espera que la supercelda usada como CP, también tenga una red de Bravais hexagonal, por lo que se puede evitar calcular todas las combinaciones de pares de elementos de la lista R al momento de crear las matrices a evaluar. Así, si el sistema está formado solo por redes hexagonales se sigue el siguiente método para calcular una MT que dé como resultado una celda hexagonal para cada VT en R:

Sea $V = x\mathbf{a} + y\mathbf{b}$ un VT guardado en R, entonces a este vector se le asocia la MT, T,

tal que:

$$T = \begin{pmatrix} x & -y \\ y & x-y \end{pmatrix}$$

.

De esta forma, cuando se evalúen sistemas formados por solo redes hexagonales (sin importar que sus constantes de red sean distintas y/o estén rotadas), el número de matrices que se evaluarán es igual al tamaño de R.

4.3. Cálculo de la matriz de deformación

Salvo casos especiales, algunas capas del sistema requerirán sufrir una deformación para mantener la periodicidad completa del sistema. Esta deformación no puede ser muy grade y está limitada directamente por el valor que se le dio a la variable ε en el momento de calcular los candidatos a VPs para el sistema.

Esta deformación se ve reflejada en matrices por las que requieren ser multiplicados los VPs de cada capa respectivamente, para que al aplicar su correspondiente MT, el resultado coincidan perfectamente con los vectores seleccionados a usar como VPs para el sistema. A esta matriz se le llamará matriz de deformación (MD).

Sea:

- V_s la matriz de VPs de la red en la capa sustrato del sistema (capa 0).
- V_i la matriz de VPs de la red en la capa i.
- T la MT que generará la CP para el sistema.
- T_i la MT propia para la capa i del sistema.

Entonces la MD para la capa i será D_i, la cual es una matriz que cumple la siguiente igualdad:

$$V_s T = (V_i D_i) T_i$$

Sabemos que V_i y T_i son matrices con determinante distinto de cero, por lo tanto tienen inverso, así obtenemos la siguiente ecuación:

$$D_i = V_i^{-1} \left(V_s T\right) T_i^{-1} \tag{4.8}$$

Los nuevos VPs para la red del sistema en la capa i se obtendrán de las columnas de la matriz $V_i' = V_i D_i$, lo cual generará un ajuste en la CP de la red. El cambio experimentado por un VP $\mathbf{u}$ tras la deformación se puede describir de manera más clara

utilizando las variables ΔM_u y ΔA_u, donde ΔM_u representa el porcentaje de variación en la magnitud del vector respecto a su valor original, y ΔA_u indica el cambio en el ángulo de su dirección.

Así, sean $\mathbf{u} = m\left(\cos(t)\hat{i} + \sin(t)\hat{j}\right)$ y $\mathbf{u'} = m'\left(\cos(t')\hat{i} + \sin(t')\hat{j}\right)$ un vector original y el resultado de este tras sufrir una deformación, entonces, definimos ΔM_u y ΔA_u de la siguiente manera:

$$\Delta M_u = \frac{(m' - m)}{m} \times 100 \quad \Delta A_u = (t' - t) \tag{4.9}$$

Para contabilizar el cambio producido en la red se utiliza el siguiente método:

Sean a, b los VPs originales de la red y a', b' los VPs deformados, entonces tomando los vectores $P_k = i_k a + j_k b$, $Q_k = i_k a' + j_k b'$ con $i_k, j_k \in [0, 1]$, $k \in \{1, 2, ..., n\}$, tendremos n parejas de vectores que van del origen a puntos dentro del área delimitada por las CPs original y deformada de la red respectivamente, entonces se define el grado de distorsión de la red DD como:

$$DD = \sum_{k=1}^{n} \left(\frac{err(P_k, Q_k)}{n} \right) \tag{4.10}$$

En donde se hace uso de la función *err*, definida en la ecuación (4.5), para obtener una aproximación al error promedio (la distorsión promedio) generado tras la deformación.

Cada MT que haga referencia a una posible solución al sistema va a implicar deformaciones para cada capa del sistema (la cual puede ser nula), estas deformaciones en las redes de cada capa las podemos calificar con su respectiva DD, por construcción, la deformación requerida por la capa sustrato es nula, lo que implica siempre tener un $DD = 0$ para esta capa.

A la suma de los DD en todas las capas del sistema inducidos por una MT la denominamos *DD total* (DDt). De esta manera podemos calificar cada MT por el grado de deformación que esta implica, entre menor sea el valor del DDt mejor opción será la MT como resultado, siendo un resultado $DDt = 0$ el óptimo indicando una deformación nula en todo el sistema.

Tras diversas pruebas en ejecución, se observó que pueden catalogarse como *resultados muy buenos* aquellos cuyo DDt sea menor a $0.02 * (c - 1)$ con c el número de capas del sistema, ya que esto significa que, en promedio, el valor del "error" correspondiente los VPs de todas las capas sobre la capa sustrato del sistema será menor al 4 %.

Tener esta "calificación" para cada resultado permite obtener un resultado que optimice tanto el tamaño de la CP (esto gracias a que las matrices en *loMat* ya son las MTs

más pequeñas posibles con los parámetros dados), así como la deformación de las capas del sistema. Esta es la matriz que el programa recomendará como respuesta al usuario si este decide no escoger una MT manualmente.

Para ejemplificar todo lo anterior, se tomará un sistema S, compuesto de tres capas: *grafeno, grafeno rotado* 13.5^o *y fosforeno negro*. Para este ejemplo se decide usar la MT:

$$T = \begin{pmatrix} 3 & 9 \\ -2 & 2 \end{pmatrix}$$

Esta MT induce a que los VPs para el sistema sean los vectores $\mathbf{u} = 3a_0 - 2b_0$ y $\mathbf{v} = 9a_0 + 2b_0$, donde $a_0 = (2.44, 0.0)$ y $b_0 = (-1.22, 2.1131)$ son los VPs de la primer capa de grafeno (la capa sustrato del sistema). De esta forma se tendría que:

$$\mathbf{u} = (9.76, -4.2262),\ \mathbf{v} = (19.52, 4.2262)$$

Al calcular los VTs para la red de grafeno rotado más parecidas a estos vectores se obtienen $\mathbf{u}_1 = 2a_1 - 3b_1$ y $\mathbf{v}_1 = 8a_1$. Aquí los VPs del grafeno rotado son $a_1 = (2.3726, 0.5696)$ y $b_1 = (-1.6796, 1.7699)$, así:

$$\mathbf{u}_1 = (9.7839, -4.1705),\ \mathbf{v}_1 = (18.9807, 4.5569)$$

De forma similar para la red de fosforeno, $\mathbf{u}_2 = 3a_2 - b_2$ y $\mathbf{v}_2 = 6a_2 + b_2$. En este caso los VPs de la red son $a_2 = (3.2601, 0.0)$ y $b_2 = (0.0, 4.347)$ haciendo que:

$$\mathbf{u}_2 = (9.7804, -4.3470),\ \mathbf{v}_2 = (19.5608, 4.3470)$$

Con este ejemplo se puede ver que $\mathbf{u} \neq \mathbf{u}_1 \neq \mathbf{u}_2$ y $\mathbf{v} \neq \mathbf{v}_1 \neq \mathbf{v}_2$, por lo que si no se aplica una deformación a las 2 capas superiores , estas perderán su periodicidad.

Para este caso:

$$V_s = \begin{pmatrix} 2.4400 & -1.2200 \\ 0.0000 & 2.1131 \end{pmatrix},\ T = \begin{pmatrix} 3 & 9 \\ -2 & 2 \end{pmatrix}$$

$$V_1 = \begin{pmatrix} 2.3726 & -1.6796 \\ 0.5696 & 1.7699 \end{pmatrix},\ T_1 = \begin{pmatrix} 2 & 8 \\ -3 & 0 \end{pmatrix}$$

$$V_2 = \begin{pmatrix} 3.2601 & 0.0000 \\ 0.0000 & 4.347 \end{pmatrix},\ T_2 = \begin{pmatrix} 3 & 6 \\ -1 & 1 \end{pmatrix}$$

Utilizando la ecuación 4.8 obtenemos las MDs correspondientes para cada capa.

$$D_0 = \begin{pmatrix} 2.4400 & -1.2200 \\ 0.0000 & 2.1131 \end{pmatrix}^{-1} \left(\begin{pmatrix} 2.4400 & -1.2200 \\ 0.0000 & 2.1131 \end{pmatrix} \begin{pmatrix} 3 & 9 \\ -2 & 2 \end{pmatrix} \right) \begin{pmatrix} 3 & 9 \\ -2 & 2 \end{pmatrix}^{-1}$$

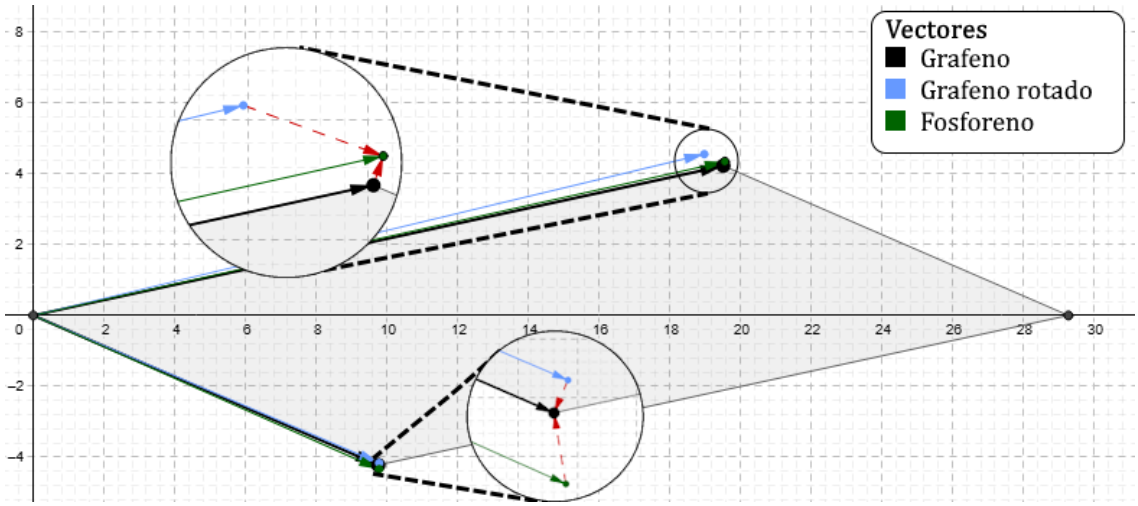


Figura 4.6: Celda primitiva tentativa para un sistema formado por grafeno, grafeno rotado 13.5^o, y fosforeno negro. La celda esta definida por los VTs de la primer capa de grafeno (vectores negros), se muestran también los VT más parecidos a estos en las capas de grafeno rotado (azul) y fosforeno (verde). Se puede apreciar que estos últimos no coinciden perfectamente con la celda primitiva por lo que requieren una deformación para empatar las celdas.

$$D_1 = \begin{pmatrix} 2.3726 & -1.6796 \\ 0.5696 & 1.7699 \end{pmatrix}^{-1} \left(\begin{pmatrix} 2.4400 & -1.2200 \\ 0.0000 & 2.1131 \end{pmatrix} \begin{pmatrix} 3 & 9 \\ -2 & 2 \end{pmatrix} \right) \begin{pmatrix} 2 & 8 \\ -3 & 0 \end{pmatrix}^{-1}$$

$$D_2 = \begin{pmatrix} 3.2601 & 0.0000 \\ 0.0000 & 4.347 \end{pmatrix}^{-1} \left(\begin{pmatrix} 2.4400 & -1.2200 \\ 0.0000 & 2.1131 \end{pmatrix} \begin{pmatrix} 3 & 9 \\ -2 & 2 \end{pmatrix} \right) \begin{pmatrix} 3 & 6 \\ -1 & 1 \end{pmatrix}^{-1}$$

Obteniendo:

$$D_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad D_1 = \begin{pmatrix} 1.00968 & 0.01524 \\ -0.02647 & 0.99001 \end{pmatrix} \quad D_2 = \begin{pmatrix} 0.99792 & 0.00000 \\ 0.00000 & 0.97220 \end{pmatrix}$$

Determinando que a excepción de la capa cero del sistema, las demás capas sí deben tener una deformación. Al aplicar esta deformación a los VPs de cada capa obtenemos unos nuevos VPs, los cuales sí mantienen la periodicidad en todo el sistema con la PC elegida.

$$V_1' = \begin{pmatrix} 2.4400 & -1.6267 \\ 0.5283 & 1.7609 \end{pmatrix} \quad V_2' = \begin{pmatrix} 3.253318992 & 0 \\ 0 & 4.2261534 \end{pmatrix}$$

Estas deformaciones sufridas por las capas 1 y 2 del sistema, al aplicar la ecuación 4.10, usando 100 pares de vectores en ella, dan como resultado $DD_1 = 0.021002$ y $DD_2 = 0.009611$, con un $DDt = 0.030613$, este valor nos indica que el resultado es lo suficientemente bueno ya que:

$$DDt < 0.02 * (3 - 1)$$

Lo que cumple la condición antes mencionada.

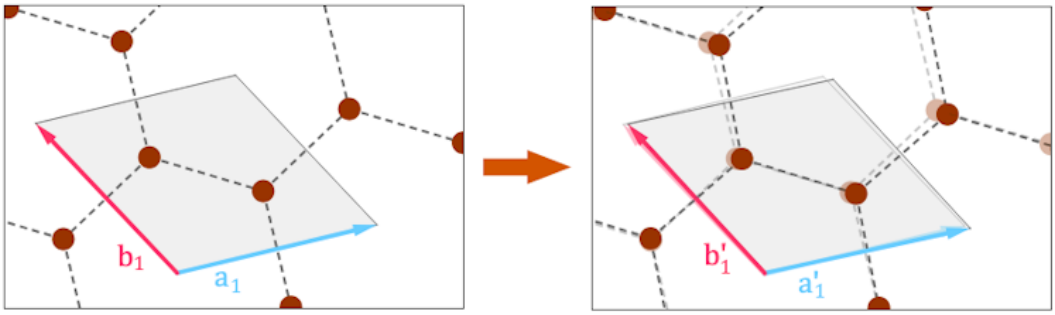


Figura 4.7: Deformación efectuada en la capa de grafeno rotada para el sistema definido. Esta deformación induce un cambio en sus VPs a y b de manera que: $\Delta M_a = 2.3\,\%$, $\Delta M_b = 1.7\,\%$, $\Delta A_a = -1.28^o$ y $\Delta A_b = -0.76^o$. Esta deformación tiene un $DD = 0.021002$.

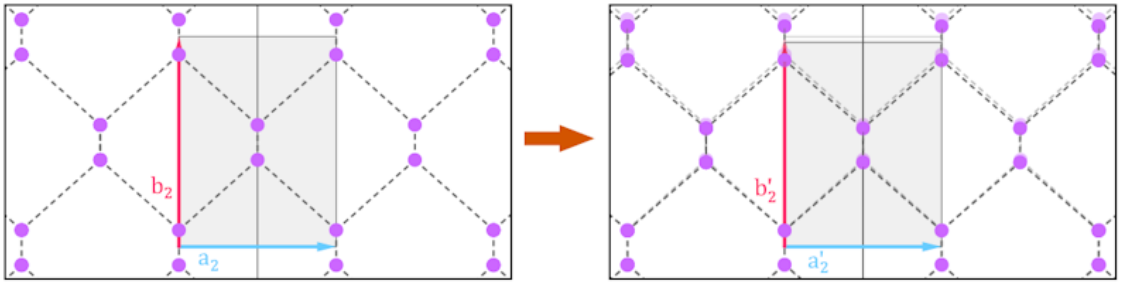


Figura 4.8: Deformación efectuada en la capa de fosforeno negro para el sistema definido. En este caso, la deformación induce un cambio en los VPs a y b de manera que: $\Delta M_a = 0.2\,\%$, $\Delta M_b = 2.8\,\%$, no cambia su dirección por lo que $\Delta A_a = \Delta A_b = 0^o$. Esta deformación tiene un $DD = 0.009611$.

La información sobre la deformación requerida por las redes del sistema para cada MT en *loMat*, que es proporcionada por el método *ShowTMs* mencionado en la subsección anterior, se trata de la MD correspondiente a cada red, las cuales están en la columna "**Deformation**" y las ΔM y ΔA que dichas deformaciones generan en sus VPs, estas en la columna "**Distortion** δ**//**ϑ**:**", donde δ corresponde al ΔM y ϑ al ΔA. Por último, el valor **DD** expresado para la tabla de cada MT corresponde al DDt del sistema para dicha MT. Estos datos pueden verse en la figura 3.5, ubicada en la sección *"Algoritmo principal"* de este mismo capítulo, la cual muestra ejemplos de las tablas desplegadas por dicho método.

4.4. Cálculo y muestra de la primera zona de Brillouin

Como ya se ha señalado, uno de los resultados extras dados por el programa, es una imagen de la primer zona de Brillouin (PZB) de las celdas primitivas para cada capa del sistema, así como una imagen de la PZB del sistema completo. Estos datos pueden ser de ayuda para el usuario ya que proporciona información crucial sobre las trayectorias que conectan puntos de alta simetría para la determinación de relaciones de dispersión electrónicas y fonónicas.

La PZB de cada red cristalina 2D corresponderá a un paralelogramo dependiendo de la simetría de la red de Bravais a la que pertenece. Para calcular la PZB se calcula la celda de Wigner-Seitz en el espacio recíproco usando los vectores recíprocos de la red.

Así la PZB corresponderá al polígono convexo delimitado por las mediatrices entre el origen y los puntos de la red recíproca. Para definir este polígono debemos conocer los puntos donde caen sus vértices, que serán donde se crucen las mediatrices que tocan al polígono.

Para calcular estos puntos, el programa hace uso de una modificación del *algoritmo de optimización Simplex*[23][2] con el método `calcVerticesFBZ` en el archivo *basics.py*. El método Simplex puede adaptarse para recorrer el perímetro de una figura convexa en 2D debido a que su algoritmo se basa en el movimiento sistemático entre vértices de un poliedro convexo a través de sus aristas, optimizando una función objetivo en cada paso. En el caso de una figura convexa bidimensional, como en el caso de la PZB que buscamos, este principio es directamente aplicable, ya que las aristas y vértices del polígono convexo representan el límite de la región factible. Según Bazaraa, Jarvis y Sherali [24], el método Simplex utiliza un enfoque estructurado para transitar por las aristas de un poliedro, lo que permite adaptar su estrategia para recorrer de manera ordenada los vértices de una figura convexa, garantizando un tránsito continuo por su perímetro.

En esta modificación del algoritmo Simplex que implementamos, efectuamos las operaciones sobre una matriz dada por las ecuaciones de las mediatrices calculadas. En el programa hacemos uso de las mediatrices entre el origen y los 8 puntos de la red recíproca más cercanos para encontrar la primer frontera; y a partir de ahí recorrer cada arista del polígono encontrando los vértices, terminando así cuando el polígono se cierra.

Para ejemplificar el funcionamiento del algoritmo, se irá resolviendo un ejemplo, en donde se mostrará lo que geométricamente está ocurriendo en cada paso.

[2]También conocido como método de Nelder-Mead o método *Amoeba* entre otros.

Sea L una red cristalina 2D oblicua con vectores primitivos $a = (2.5, 0.0)$ y $b = (-1.75, 3.0)$ a la que calcularemos su PZB. El primer paso es identificar los 8 puntos de la red, en el espacio recíproco, más cercanos al origen, los cuales en este caso son:

$$\begin{aligned} V_1 &= (0.0000, -2.0944) & V_2 &= (0.0000, 2.0944) \\ V_3 &= (-2.5133, 0.6283) & V_4 &= (2.5133, -0.6283) \\ V_5 &= (-2.5133, -1.4661) & V_6 &= (2.5133, 1.4661) \\ V_7 &= (-2.5133, 2.7227) & V_8 &= (2.5133, -2.7227) \end{aligned}$$

Con estos puntos v_i calculamos las rectas mediatrices m_i entre ellos y el origen (figura 4.9).

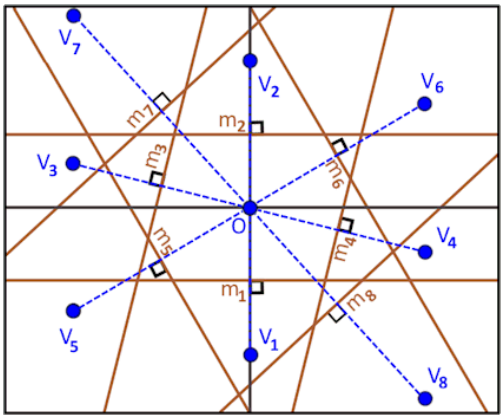


Figura 4.9: PRs más cercanos al origen y sus mediatrices con este.

A partir de estas rectas creamos una tabla de 11 columnas por 8 filas. Con las primeras 8 columnas crearemos una matriz identidad de 8×8. Mientras que las columnas 9, 10 y 11 serán los valores a, b y c de las ecuaciones de cada una de las rectas m_i, donde cada fila i corresponderá a una recta m_i; tal y como se ve en la tabla 4.1.

$$\begin{aligned} m_1 &: 0.00x - 4.19y = 4.39 & m_2 &: 0.00x + 4.19y = 4.39 \\ m_3 &: -5.03x + 1.26y = 6.71 & m_4 &: 5.03x - 1.26y = 6.71 \\ m_5 &: -5.03x - 2.93y = 8.47 & m_6 &: 5.03x + 2.93y = 8.47 \\ m_7 &: -5.03x + 5.45y = 13.73 & m_8 &: 5.03x - 5.45y = 13.73 \end{aligned}$$

En esta tabla etiquetaremos las primeras 8 columnas para hacer referencia a las rectas m_i mientas que las columnas 9 y 10 corresponderán a las rectas $x = 0$ y $y = 0$ (aquí, al estar en el espacio recíproco deberíamos trabajar con la base recíproca x', y', sin embargo por comodidad se llamarán como x y y). A partir de esta tabla se iniciará el algoritmo.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-4.19	4.39
m_2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.19	4.39
m_3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	-5.03	1.26	6.71
m_4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	5.03	-1.26	6.71
m_5	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	-5.03	-2.93	8.47
m_6	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	5.03	2.93	8.47
m_7	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-5.03	5.45	13.73
m_8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	5.03	-5.45	13.73

Tabla 4.1: Estado inicial de la tabla

El primer paso es ubicarnos en el perímetro del polígono para movernos por él y encontrar un primer vértice. Iniciamos encontrando la intersección del perímetro con el eje x en el lado positivo, para esto, en la tabla buscamos *"sacar"* una de las filas m_i y *"meter"* en su lugar una fila correspondiente a la columna x.

Lo que hacemos es fijarnos en las filas con valores mayores a cero en la columna x; en estas filas dividimos el valor en la columna c entre el valor en la columna x y nos tomamos como *pivote* a la celda en la columna x de la fila que arrojó el menor resultado.

En este caso, las filas en que nos fijamos son m_4, m_6 y m_8 que son las únicas con valores mayores a cero en la columna x, al hacer las operaciones obtenemos $\frac{6.71}{5.03}$ para m_4, $\frac{8.47}{5.03}$ para m_6 y $\frac{13.73}{5.03}$ para m_8, por lo que la fila con el menor valor es m_4, así tomamos como pivote la casilla $[m_4, x]$.

Teniendo localizado el pivote, debemos *"meter"* la fila x, para esto debemos hacer operaciones de tal manera que al final la columna x tenga ceros en todas sus casillas excepto en la casilla pivote, donde debemos tener un valor de 1, llegando así a una nueva tabla (Tabla 4.2).

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-4.19	4.39
m_2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.19	4.39
m_3	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	13.42
x	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	1.0	-0.25	1.34
m_5	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	-4.19	15.18
m_6	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	0.0	0.0	4.19	1.75
m_7	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	4.19	20.44
m_8	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	1.0	0.0	-4.19	7.02

Tabla 4.2: Tabla con una fila correspondiente a x.

Las operaciones que se siguen para realizar el proceso de *sustituir* una fila **a** por una nueva fila **b** en una matriz de valores reales, utilizando como *pivote* el valor en la posición $[a, b] = p$, son los siguientes:

1. **Normalización de la fila pivote:**

- Se dividen todos los elementos de la fila **a** por el valor del pivote p, realizando la operación $\mathbf{a} = \mathbf{a}/p$.
- Se renombra la fila **a** como **b**. Con esto, la fila **b** ocupa el lugar de la fila original **a**, y el valor en la posición de pivote (ahora $[b, b]$) se convierte en 1.

2. **Reducción de las demás filas:**

   - Para cada fila **c** distinta de la fila **b** ($\mathbf{c} \neq \mathbf{b}$), se realiza la operación $\mathbf{c} = \mathbf{c} - q_c\mathbf{b}$, donde q_c es el valor de la casilla $[c, b]$ en la fila **c**. Esta operación hace que el valor en la posición $[c, b]$ de cada fila **c** (distinta de **b**) se convierta en 0.

Siguiendo estos pasos, se obtiene una nueva fila **b** en la posición de la fila **a**, logrando que toda la columna correspondiente a **b** tenga ceros, excepto en la posición $[b, b]$, que contiene el valor 1 resultante de la normalización de la fila **b**.

Lo que se hizo al *meter* la fila x, es equivalente a lanzar un rayo desde el origen en dirección del eje X, obtener las intersecciones de este con las mediatrices dibujadas y quedarnos con la más próxima al origen[3], en el caso del ejemplo fue la intersección del eje X con la recta m_4 (figura4.10), posicionándonos en un punto del perímetro de la PZB de la red.

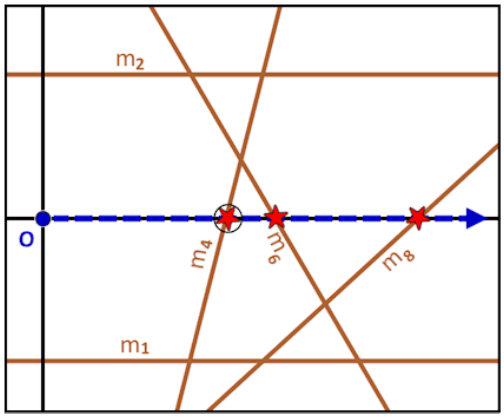


Figura 4.10: Representación visual del primer paso del algoritmo, en este se encuentran las intersecciones del rayo a partir del origen en dirección del eje X y las mediatrices calculadas.

Ahora, para encontrar el primer vértice del polígono debemos meter a y en las filas. Para esto hacemos un paso similar al anterior, buscando un pivote, pero ahora en la columna y en lugar de la columna x, siendo en esta ocasión la casilla $[m_6, y]$ la que tomamos

[3]Esta interpretación es tomada directamente de la interpretación geométrica para el inicio del método Simplex, donde nos posicionamos de inicio en un punto que es una solución básica factible, en este caso, el punto máximo con y=0 dentro del polígono convexo delimitado por las mediatrices trazadas.

de pivote. Ejecutamos las operaciones para meter la fila y, obteniendo una nueva tabla (Tabla 4.3).

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_1	1.0	0.0	0.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	6.14
m_2	0.0	1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	2.63
m_3	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	13.42
x	0.0	0.0	0.0	0.14	0.0	0.06	0.0	0.0	1.0	0.0	1.44
m_5	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	16.93
y	0.0	0.0	0.0	-0.24	0.0	0.24	0.0	0.0	0.0	1.0	0.42
m_7	0.0	0.0	0.0	2.0	0.0	-1.0	1.0	0.0	0.0	0.0	18.69
m_8	0.0	0.0	0.0	-2.0	0.0	1.0	0.0	1.0	0.0	0.0	8.77

Tabla 4.3: Tabla después de meter a x y y en las filas, en este podemos observar que el primer vértice del polígono corresponderá al punto $(1.44, 0.42)$

Con esta tabla obtenemos el valor del punto P_1, el primer vértice del polígono que representa la PZB de la red, este punto coincidirá con el cruce de las dos rectas que "sacamos" de las filas, en este caso son m_4 y m_6. Para obtener la posición de este punto debemos ver los valores correspondientes a las filas x y y en la columna c de la última tabla obtenida. En este ejemplo el primer vértice del polígono es $P_1 = (1.44, 0.42)$ (figura 4.11).

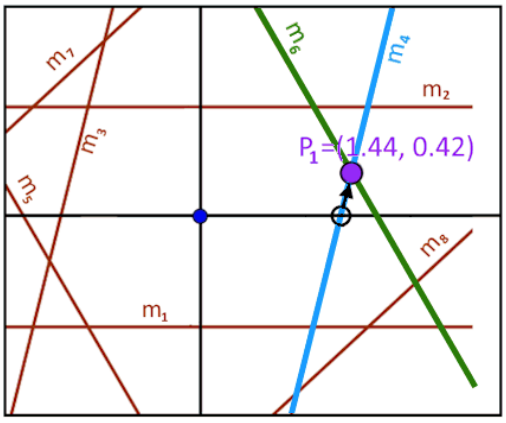


Figura 4.11: Primer vertice encontrado en la intersección de m_4 y m_6

A partir de aquí comienza la segunda fase del algoritmo, en esta se recorren las aristas del polígono encontrando en orden antihorario los demás vértices.

Se inicia buscando "meter" m_k, la penúltima recta que ha salido de las filas. Para esto buscamos la celda pivote (m_l, m_k), donde m_l será la fila que "saldrá" y será remplazada por la fila que intentamos "meter", teniendo cuidado de nunca "sacar" las filas x o y.

La fila m_l será aquella fila entre las filas con valores mayores a cero en la columna m_k con el **menor resultado no negativo** al dividir el valor en la columna c entre su valor en la columna m_k. Si el menor valor no negativo corresponde a las filas x o y optamos por la siguiente mejor opción.

Una vez obtenida la celda pivote, efectuamos las operaciones para meter la fila m_k y sacar la fila m_l, obteniendo una nueva tabla, la cual mostrará la posición del punto $P_i = ([x, c], [y, c])$ el corresponderá a una nueva arista del polinomio.

Repetiremos esto hasta que $P_i = P_1$, donde sabremos que hemos recorrido todo el perímetro y encontrado todos los vértices del polígono.

En el ejemplo, la primer fila que buscamos meter es la correspondiente a la recta m_4, aquí las filas candidatas a salir son m_2, m_3, x y m_7, teniendo como resultados de dividir c/m_4: $\frac{2.63}{1} = 2.63$, $\frac{13.42}{1} = 13.42$, $\frac{1.44}{0.14} = 10.29$ y $\frac{18.69}{2} = 9.35$ respectivamente. Indicando entonces que el pivote será la celda $[m_2, m_4]$. Efectuando las operaciones correspondientes obtenemos una nueva tabla (Tabla 4.4).

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_1	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.77
m_4	0.0	1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	2.63
m_3	0.0	-1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	10.79
x	0.0	-0.14	0.0	0.0	0.0	0.2	0.0	0.0	1.0	0.0	1.07
m_5	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	16.93
y	0.0	0.24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.05
m_7	0.0	-2.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	13.42
m_8	0.0	2.0	0.0	0.0	0.0	-1.0	0.0	1.0	0.0	0.0	14.04

Tabla 4.4: Tabla después de regresar a m_4 y sacar m_2 de las filas.

De esta tabla obtenemos el punto $P_2 = ([x, c], [y, c]) = (1.07, 1.05)$. En este punto se intersectan las rectas m_6 y m_2 (figura 4.12).

Dado que el punto P_2 es distinto a P_1 repetimos el algoritmo para buscar el siguiente punto.

Ahora intentamos meter la fila m_6. En este caso la mejor opción de pivote es $[x, m_6]$ pero dado que esto sacaría a x de las filas, se toma la siguiente mejor opción, $[m_3, m_6]$,y se efectúan las operaciones correspondientes para obtener una nueva tabla (Tabla 4.5).

De esta tabla se obtiene el punto $P_3 = (-1.07, 1.05)$, dado que este es distinto a P_1 continuamos (figura 4.13).

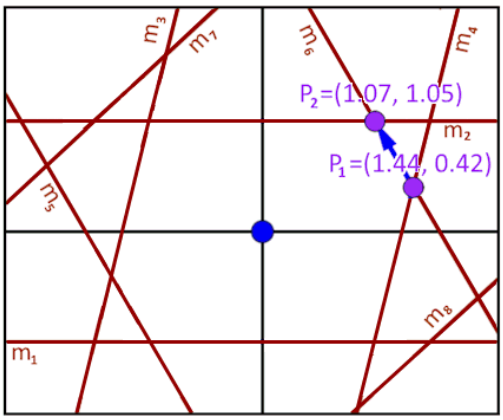


Figura 4.12: Segundo vértice encontrado en la intersección de m_6 y m_2

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_1	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.77
m_4	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	13.42
m_6	0.0	-1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	10.79
x	0.0	0.06	-0.2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-1.07
m_5	0.0	1.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	6.14
y	0.0	0.24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.05
m_7	0.0	-1.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	2.63
m_8	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	24.83

Tabla 4.5: Tabla después de meter a m_6 y sacar m_3 de las filas.

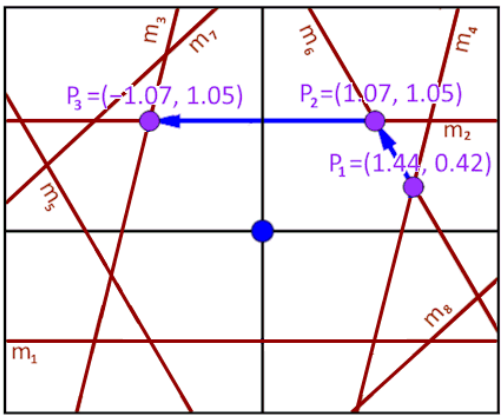


Figura 4.13: Tercer vértice encontrado en la intersección de m_2 y m_3

Buscamos ingresar la fila m_2, aquí se tiene como mejor opción de pivote $[y, m_2]$, descartamos esta opción y usamos el siguiente mejor, $[m_5, m_2]$ y generamos la nueva tabla (Tabla 4.6).

De esta se obtiene la intersección entre las rectas m_3 y m_5, $P_4 = (-1.44, -0.42)$ (figura 4.14).

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_1	1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	2.63
m_4	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	13.42
m_6	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	16.93
x	0.0	0.0	-0.14	0.0	-0.06	0.0	0.0	0.0	1.0	0.0	-1.44
m_2	0.0	1.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	6.14
y	0.0	0.0	0.24	0.0	-0.24	0.0	0.0	0.0	0.0	1.0	-0.42
m_7	0.0	0.0	-2.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	8.77
m_8	0.0	0.0	2.0	0.0	-1.0	0.0	0.0	1.0	0.0	0.0	18.69

Tabla 4.6: Tabla después de meter a m_2 y sacar m_5 de las filas.

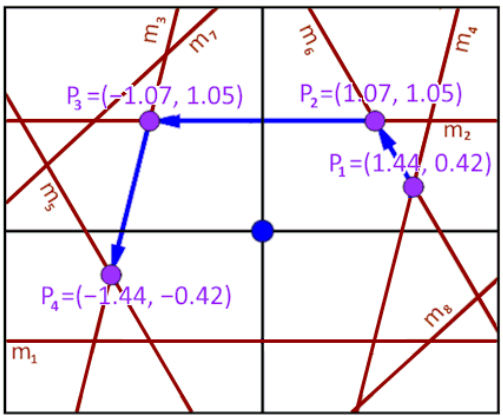


Figura 4.14: Cuarto vértice encontrado en la intersección de m_3 y m_5

Continuamos el algoritmo un par de veces más, metiendo m_3 y sacando m_1, después metiendo m_5 y sacando m_4, obteniendo los puntos $P_5 = (-1.07, -1.05)$ y $P_6 = (1.07, -1.05)$.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_3	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	13.42
m_5	-1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	10.79
m_6	1.0	0.0	0.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	6.14
x	-0.06	0.0	0.0	0.2	0.0	0.0	0.0	0.0	1.0	0.0	1.07
m_2	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.77
y	-0.24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	-1.05
m_7	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	24.83
m_8	-1.0	0.0	0.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	2.63

Tabla 4.7: Tabla después de efectuar varias repeticiones del algoritmo y haber obtenido los puntos $P_1, P_2, P_3, P_4, P_5, P_6$.

Tenemos ahora la Tabla 4.7, donde el pivote es (m_1, m_6), al sacar m_6 para meter m_1 y hacer las operaciones adecuadas obtenemos una nueva Tabla 4.8, la cual nos señala como nueva arista el punto $P_7 = (1.44, 0.42)$. Pero este es igual a P_1, por lo que ya no admitimos a P_7 como nuevo vértice y terminamos el algoritmo, teniendo que la PZB de la red corresponderá al *polígono* $[P_1, P_2, P_3, P_4, P_5, P_6]$ (figura 4.15).

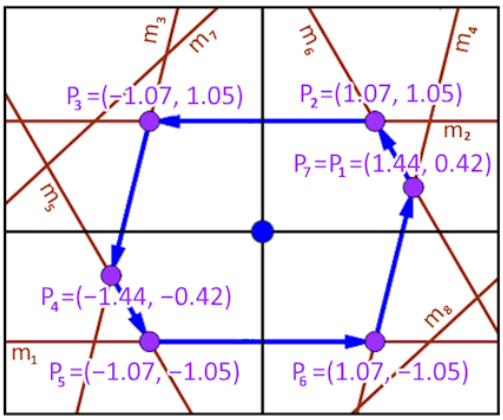


Figura 4.15: Todos los vértices del polígono que forma la PZB y su recorrido en la búsqueda, comenzando en P_1 y terminando con P_7 al ser el mismo punto.

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	x	y	c
m_1	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	13.42
m_4	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	16.93
m_6	1.0	0.0	0.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	6.14
x	0.0	0.0	0.0	0.14	0.0	0.06	0.0	0.0	1.0	0.0	1.44
m_2	0.0	1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	2.63
y	0.0	0.0	0.0	-0.24	0.0	0.24	0.0	0.0	0.0	1.0	0.42
m_7	0.0	0.0	0.0	2.0	0.0	-1.0	1.0	0.0	0.0	0.0	18.69
m_8	0.0	0.0	0.0	-2.0	0.0	1.0	0.0	1.0	0.0	0.0	8.77

Tabla 4.8: Último estado de la tabla. Aquí los valores para las celdas (c, x) y (c, y) vuelven a ser los mismos que en el inicio de la segunda fase del algoritmo, indicando que se terminó de recorrer todo el perímetro del polígono.

Conociendo las posiciones de los vértices del polígono que delimita la PZB de la red, es posible representarlo gráficamente en el espacio recíproco (figura 4.16). Analizando la complejidad de este algoritmo, se observa que el número de repeticiones de las operaciones necesarias para *meter* y *sacar* filas de la matriz, está limitado por el número de vértices del polígono resultante. Este número, debido a las propiedades de teselación del plano[4], no puede exceder seis como consecuencia de restricciones geométricas [25].

Por lo tanto, la complejidad de nuestro algoritmo para determinar la PZB de una red es independiente de la red específica, ya que está acotada por una constante. Esto implica que pertenece a la clase de complejidad $O(1)$.

En la ejecución del software, al aplicar este algoritmo para un sistema, este se ejecuta un número de veces igual al numero de capas que conforman dicho sistema más uno (el correspondiente a la CP calculada para el sistema), por lo que su complejidad final

[4]Un polígono convexo puede teselar el plano si y solo si los ángulos interiores permiten que los polígonos encajen sin dejar huecos, de manera que la suma de los ángulos alrededor de un vértice sea igual a 360°.

pertenece a la clase $O(m)$ con m igual al número de capas del sistema.

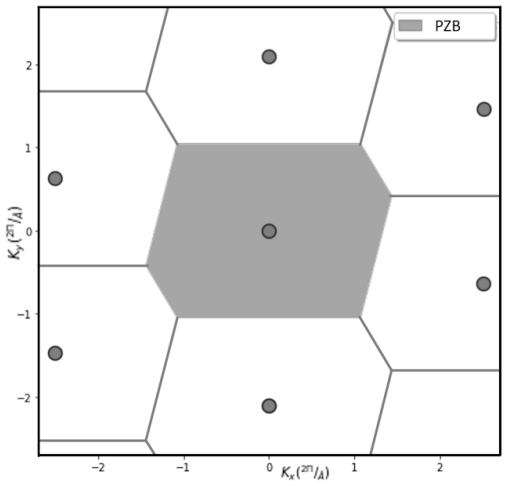


Figura 4.16: Imagen dada por el programa de la PZB de la red evaluada obtenida a partir del algoritmo descrito.

Funciones

A.1. Basics

A.1.1. Operaciones con vectores

En todo el código, se realizan operaciones con vectores. Dado que estamos trabajando con periodicidad en dos dimensiones, la mayoría de los vectores utilizados son bidimensionales. En este programa, representamos estos vectores mediante pares de números que indican la posición en el plano del punto correspondiente al vector que estamos representando (su tipo sería descrito como `v2d::(float,float)`).

En el caso de las operaciones de producto punto y producto cruz, operamos con vectores tridimensionales. En estos casos, un vector se representa con un conjunto de tres números que indican la posición en el espacio del punto que representa este vector (en este caso su tipo sería descrito como `v3d::(float,float,float)`).

A continuación, se presentan las funciones que realizan operaciones sobre vectores:

- **sumaV($\vec{a}, \vec{b}$)**:: `v2d,v2d` $\rightarrow$ `v2d`.
  Suma los vectores $\vec{a}$ y $\vec{b}$.
- **multV($n, \vec{a}$)**:: `v2d,v2d` $\rightarrow$ `v2d`.
  Multiplica el vector $\vec{a}$ por la constante n.
- **m2V($\vec{a}, \vec{b}, s$)**:: `v2d,v2d,(float,float)` $\rightarrow$ `v2d`.
  Recibe los vectores $\vec{a}$, $\vec{b}$ y un par de números $s = (m, n)$. Calcula la combinación lineal $m\vec{a} + n\vec{b}$.
- **rota($\vec{v}, \theta$)**:: `v2d,float` $\rightarrow$ `v2d`.
  Regresa el vector resultante de rotar el vector $\vec{v}$ por un ángulo de θ grados.
- **dist($\vec{a}, \vec{b}$)**:: `v2d,v2d` $\rightarrow$ `float`.
  Calcula la distancia entre los puntos representados por los vectores $\vec{a}$ y $\vec{b}$.

- **long($\vec{v}$)**:: `v2d` $\rightarrow$ `float`.
  Calcula la magnitud del vector $\vec{v}$.

- **cRot($\vec{v}$)**:: `v2d` $\rightarrow$ `float`.
  Devuelve la dirección del vector $\vec{v}$ en grados.

- **cAng($\vec{a}, \vec{b}$)**:: `v2d,v2d` $\rightarrow$ `float`.
  Calcula el ángulo entre los vectores $\vec{a}$ y $\vec{b}$, el resultado toma en cuenta la dirección del ángulo, por lo que puede ser positivo (si este es en sentido antihorario) o negativo (si es en sentido horario).

- **pC($\vec{a}, \vec{b}$)**:: `v3d,v3d` $\rightarrow$ `v3d`.
  Calcula el producto cruz de los vectores $\vec{a}$ y $\vec{b}$.

- **pP($\vec{a}, \vec{b}$)**:: `v3d,v3d` $\rightarrow$ `v3d`.
  Calcula el producto punto de los vectores $\vec{a}$ y $\vec{b}$.

- **to2D($\vec{v}$)**:: `v3d` $\rightarrow$ `v2d`.
  Obtiene la proyección bidimensional del vector tridimensional $\vec{v}$.

- **to3D($\vec{v}$)**:: `v2d` $\rightarrow$ `v3d`.
  Obtiene un vector 3D equivalente al vector 2D dado $\vec{v}$.

A.1.2. Operaciones con matrices

En el programa Trabaja solamente con operaciones en matrices de 2×2, estas son representadas por una lista de 2 listas de 2 números que representarán sus filas (`m2x2::[[float,float],|`

Las operaciones que realizan operaciones sobre las matrices son las siguientes:

- **sumaM(A, B)**:: `m2x2, m2x2` $\rightarrow$ `m2x2`.
  Suma las mtrices A y B.

- **multM(c, M)**:: `float, m2x2` $\rightarrow$ `m2x2`.
  Multiplica la matriz M por la constante c

- **VtM($\vec{u}, \vec{v}$)**: Genera una matriz 2x2 a partir de dos vectores 2D dados usándolos como columnas de esta.

- **MtV(M)**:: `m2x2` $\rightarrow$ `v2d,v2d`.
  Convierte las 2 columnas de la matriz dada en vectores bidimensionales.

- **det(M)**:: `m2x2` $\rightarrow$ `float`.
  Calcula el determinante de la matriz M

- **inv2x2(**M**)**:: `m2x2` $\rightarrow$ `m2x2`.
  Calcula la matriz inversa de la matriz M.

- **m2M(**A, B**)**:: `m2x2,m2x2` $\rightarrow$ `m2x2`.
  Multiplica la matriz A por la matriz B.

- **transfVs(**$\vec{u}, \vec{v}, T$**)**:: `v2d,v2d,m2x2` $\rightarrow$ `v2d,v2d`.
  Regresa como 2 vectores bidimensionales la matriz resultante de multiplicar la matriz $[\vec{u}, \vec{v}]$ por la matriz T.

A.1.3. Funciones auxiliares

A continuación se enlistan las funciones en Basics.py usadas como funciones auxiliares en funciones próximas.

- **getLim(**$\vec{u}, \vec{v}, m, n$**)**:: `v2d,v2d,int,int` $\rightarrow$ `[float,float],[float,float]`.
  Calcula los valores máximos y mínimos en X y Y del rectángulo donde está inscrito el paralelogramo formado por los vectores $m\vec{u}, n\vec{v}$. Es utilizada para determinar los limites del espacio para dibujar en pantalla.

- **esRotacion(**$\vec{a}, \vec{b}, \vec{c}, \vec{d}, \varepsilon$**)**:: `v2d,v2d,v2d,v2d,float` $\rightarrow$ `bool`.
  Determina si el par de vectores $\vec{a}, \vec{b}$ son resultado de rotar el par de vectores $\vec{c}, \vec{d}$ permitiendo un error máximo dado por ε, si este no se especifica se toma por defecto $\varepsilon = 0.001$.

- **acomoda(**$x, valor, lis, tamMax$**)**:: `object, float, [[object, float]], float` $\rightarrow$ `[[object, float]]`.
  Agrega el objeto x en la lista ordenada lis de acuerdo a su valor asociado $valor$, limitando el tamaño de la lista a $tamMax$.

- **pmat(**M**)**:: `m2x2` $\rightarrow$.
  Imprime una matriz M en pantalla como un String.

- **checkP(**$x, y, err = 0.001$**)**:: `float,float,float` $\rightarrow$ `bool`.
  Evalúa si dos valores representan al mismo en un periodo de tamaño 1.

- **cRecip(**$\vec{a}, \vec{b}, \vec{c}$**)**:: `v3d,v3d,v3d` $\rightarrow$ `[v3d,v3d,v3d]`.
  Dada una terna de Vectores tridimensionales, calcula sus respectivos Vectores recíprocos.

- **buscaEsquina(**$M, j, e1, e2$**)**:: `[[float]],int,int,int` $\rightarrow$ `int`.
  Funcion auxiliar usada para calcular los vértices de la FBZ.

- **opera(***M, e, s***)**:: `[[float]],int,int` $\rightarrow$.
  Usando operaciones en la Matriz M la transforma en otra donde la columna e sea de ceros excepto en la celda $M_{s,e}$ donde tiene el valor 1.

- **dameVecinos(***rv***)**:: `[v3d,v3d,v3d]` $\rightarrow$ `[(float,float)]`.
  Regresa los 8 puntos de red más cercanos al origen dados por una red con los vectores primitivos señalados en *rv*.

- **califica(***pos, lista***)**:: `(float,float),[(float,float)]` $\rightarrow$ `bool`.
  Funcion auxiliar de *dameVecinos*, determina si un punto en la posición relativa *pos* no es "cubierto" por algún punto con una posición en *lista* desde el origen.

- **calcVerticesFBZ(***rv***)**:: `[v3d,v3d,v3d]` $\rightarrow$ `[(float,float)],[[float]]`.
  Calcula los vértices de la celda de Wigner-Seitz para la red con los vectores primitivos señalados en *rv*.

- **aN(a)**::`str->float`
  Analiza la entrada *'a'*, si esta coincide con el *símbolo atómico* de un elemento químico entonces regresa el *Número atómico* de este elemento, de lo contrario regresa 0.5

- **F_G(a,G)**::`(str,v2d)->float`
  Calcula el *factor de forma atómica* para el átomo con símbolo químico 'a' en la posición dada por el vector G[1].

[1]Los datos para el cálculo provienen de las Tablas Internacionales de Cristalografía http://it.iucr.org/Cb/ch6o1v0001/

A.2. Atom

A.2.1. Inicialización

El objeto *Atom* recibe como parámetros de inicio:

- **pos**:: `(float,float)`.
  Indica la posición relativa del Átomo a los vectores primitivos $\vec{a}$ y $\vec{b}$ de la red donde se encuentra.

- **posZ**:: `float`.
  Indica la posición relativa del Átomo al vector primitivo $\vec{c}$ de la red donde se encuentra. Si no se indica se toma el valor 0.0 por default.

- **color**:: `str`.
  Indica el color con el que se pintará el Átomo al dibujarse en pantalla. Si no se indica se toma por default $'black'$.

- **sig**:: `str`.
  Identificador para señalar el tipo de Átomo al que pertenece este, se recomienda usar el Símbolo del Átomo. Si este no se indica se toma como valor por default $'C'$.

Estos parámetros originan los Atributos homónimos del objeto Átom.

A.2.2. Métodos de Atom

Los métodos para Atom son los siguientes:

- **__str__()**:: $\rightarrow$`str`.
  Expresa Atom cómo un str de la forma: Atom.sig+str(Atom.pos).

- **setData(***color*, *sig***)**:: `color,str`$\rightarrow$.
  Cambia los atributos 'color' y 'sig' del Atom.

- **getPos()**:: $\rightarrow$`(float,float)`.
  Regresa el atributo 'pos'.

- **clasifica(***loa***)**:: `[[Atom]]`$\rightarrow$`int`.
  Dada *loa*, una lista de listas de Atoms clasificados por su atributo 'sig'. Si en *loa* hay una lista de objetos Atom con el mismo 'sig' que el Atom lo agrega a esta lista y regresa 1, de lo contrario genera una nueva lista en *loa* que lo contenga y regresa 0.

A.3. Lattice

A.3.1. Inicialización

Los objetos de clase Lattice modelan redes cristalinas, los parámetros de entrada para inicializar esta clase son:

- **vA**:: `v2d`.
  Vector primitivo a de la red cristalina.
- **vB**:: `v2d`.
  Vector primitivo b de la red cristalina.
- **atm**:: `[Atom]`.
  BA de la red cristalina.
- **enls**:: `[((float,float),(float,float))]`.
  Lista de las posiciones inicio y final de los segmentos de línea que se pintarán para representar los enlaces atómicos. (Solo requerido para representarse en pantalla dentro del programa, si este no de da o como parámetro se da una lista vacía, en las imágenes no se pintarán los enlaces pero no afecta en el archivo POSCAR generado al exportarse la Red o Sistema del que forme parte).
- **name**:: `str`.
  Nombre que se le da a la red cristalina.
- **detachment** :: `float`.
  Indica el grosor de la red cristalina en Angstroms.
- **prof**:: `int`.
  Indica la capa a la que pertenece esta red si es que pertenece a un sistema apilado, si no es parte de un Sistema su valor es 1.

Los Atributos propios de esta Clase generados a partir de los parámetros de inicio son:

- a:: `v2d` ; b:: `v2d`.
  Atributos que señalan los vectores primitivos $\vec{a}$ y $\vec{b}$ de la red, son dados directamente por vA y vB.
- OriginalA:: `v2d` ; OriginalB:: `v2d`.
  Atributos que señalan los vectores primitivos $\vec{a}$ y $\vec{b}$ de la red, aquí se toman los vectores de la red rotada de tal manera que la dirección del vector $\vec{a}$ sea de 0^o.

- prof:: `int`; detachment:: `float`; name:: `str`; enls:: `[(float,float),(float,float)]`.
  Atributos donde se guardan directamente los parámetros homónimos.

- theta:: `float` ; inAngle:: `float`.
  Atributos que señalan ángulos de la Red, el primero es dado por la rotación requerida para llegar a la red por modelar a partir de la misma red si la dirección del vector $\vec{a}$ fuera de 0^o. El segundo ángulo es el ángulo inscrito entre los vectores $\vec{a}$ y $\vec{b}$.

- atms:: `[[Atom]]`.
  Lista de Listas de Atoms clasificados por su Atributo 'sig', esta es obtenida a partir del parámetro homónimo.

- reciprocalVectors:: `[v3d,v3d,v3d]`.
  Terna de los vectores primitivos de la red reciproca correspondiente.

- layerList :: `[Lattice]`.
  Si esta Lattice modela una Red cristalina formada por el apilamiento de 2 o más redes, aquí se enlistan las Lattices correspondientes a estas.

A.3.2. Métodos de Lattice

- **__str__()**:: →`str`.
  Expresa el Lattice cómo un str con el formato de un archivo POSCAR.

- **get_pv**:: → `v2d,v2d`.
  Regresa los vectores primitivos $\vec{a}$ y $\vec{b}$.

- **showme(**$x, y, x0, y0, t, iName, sampling, scalePosL$**)**::
  `int,int,int,int,float,str,bool,bool` →.
  Despliega en pantalla una representación de la super-red dada por los valores de x y y. Los parámetros $x0$ y $y0$ son usados si se requiere imprimir la super-red a partir de una posición distinta al origen. El parámetro t indica el grosor con el que se dibujarán los átomos y enlaces. El parámetro *iName* indica el nombre del archivo png con el que se guardará la imagen generada (en la carpeta Images), si este parámetro es proporcionado, la imagen no se guardará y solo se desplegará en pantalla. Los parámetros booleanos sampling y scalePosL determinan correspondientemente si se dibujarán los vectores primitivos $\vec{a}, \vec{b}$ en la imagen y si la posición de la escala se posicionará o no a la izquierda.

- **showPC(iName)**::`str`→`none`
  Este llama al método *showme* con los parametros nesesaros para imprimir una imagen con solo la celda primitiva de la red indicando sus vectores primitivos. Igualmente que el método anterior, el parámetro *iName* es opcional, si este es dado será

el nombre con el que se guardará la imagen, en caso de no darse la imagen solo se desplegará en pantalla sin guardarse.

- **addAtms(*loa*)**:: `[Atom]` $\rightarrow$.
  Agrega de forma ordenada los Atoms en la lista *loa* al parámetro *atoms*.

- **showData()**::$\rightarrow$`str`
  Genera un texto con formato POSCAR con los datos de la red.

- **aligned()**
  Rota la red de tal manera que el vector $\vec{a}$ esté alineado al eje X.

- **rotate(θ)**:: `float` $\rightarrow$.
  Rota la red en θ^o.

- **alignedLattice()**:: $\rightarrow$ `Lattice`.
  Regresa una copia de la red alineada al eje X.

- **mRot(θ)**:: θ $\rightarrow$ `Lattice`.
  Regresa una copia de la red rotada en θ^o.

- **export(*name*)**:: `str` $\rightarrow$.
  Exporta la red en un archivo POSCAR <name>.vasp

- **setNewVectors(*newA, newB*)**:: `v2d,v2d` $\rightarrow$.
  Establece nuevos valores para los atributos de Lattice *a* y *b*.

- **getVectors()**:: $\rightarrow$ `v2d,v2d`.
  Regresa los atributos de Lattice *a* y *b*.

- **getOV()**:: $\rightarrow$ `v2d,v2d`.
  Regresa los atributos de Lattice *OriginalA* y *OriginalB*.

- **nOAtms()**:: $\rightarrow$ `int`.
  Regresa el número de Átomos en la BA de la red.

- **loAtms()**:: $\rightarrow$ `[[str,float,float,float,v2d]]`.
  Esta es una función auxiliar utilizada para calcular el patrón de difracción de la red. Genera una lista con datos de cada objeto *Atom* en la BA de la red (el atributo *atms* de la clase *Lattice*). Cada elemento de la lista de salida consta de los siguientes datos tomados de cada átomo: atributo *sig*, posición relativa X, posión relativa Y, posición relativa Z, vector 2D con la posición absoluta de la proyección del átomo en el plano.

- **F_hkl(**h, k, FG**)**::`float,float,bool→float`
  Esta es una función auxiliar utilizada para calcular el patrón de difracción de la red. Calcula el *Factor de estructura* correspondiente al punto de la red reciproca h, k, l correspondiente (dado que nuestra estructura es 2D solo se utilizan h y k).

- **reciprocalBackgroundMesh(**$vl, t, border, calcS, rnd, FG$**)**::
  `[(float,float)],float,float,bool,int,bool →`
  `[float],[float],[float],linkList`.
  Esta es una función auxiliar utilizada para calcular el patrón de difracción y la representación del espacio reciproco de la red. Calcula los datos necesarios para dibujar es espacio recíproco de la red. La lista *vl* corresponde a los vértices de la FBZ de la red, el espacio que se muestra es limitado por *border*, *rnd* indica el número máximo de dígitos después del punto para redondear los resultados de la lista *S* que es parte de la salida de esta función. Los booleanos *calcS* y *FG* determinan respectivamente si se calculará el factor de estructura (que será guardado en la lista S) y si para este cálculo se tomará en cuenta el vector de dispersión.
  La salida de este método está formada por tres listas de valores reales y una lista de vértices, las primeras 2 listas (X,Y) guardan las posiciones de los puntos de la red reciproca dentro del espacio determinado por *border*, la tercera lista (S) tiene los valores del Factor de estructura calculados para cada punto (es vacía si $calcS = False$) y la lista de vértices tiene los vértices requeridos para crear una maya compuesta por calcular la FBZ en cada punto.

- **printReciprocalSpace(**$t, border, prnt, zoom, colors$**)**::
  `float,float,bool,bool,[str]→`.
  Despliega en pantalla una representación del espacio reciproco de la red. Si el Lattice corresponde a una estructura de redes apiladas dibuja la FBZ de cada red que lo forma y la de esta misma. Los valores *t* y *border* determinan el grosor general con el que se dibujará todo y el límite del espacio de dibujo respectivamente. Si el booleano $prnt = True$ entonces guarda la imagen en la carpeta "image", si el $zoom = True$ entonces la imagen desplegada corresponderá solamente al primer cuadrante del plano. La lista *colors* determina los colores con los que se dibujarán las FBZ de cada red en el sistema, si esta se proporciona debe indicar un color para cada capa del sistema, si no se proporciona entonces el color para cada capa será al azar.

- **printLightPoints(**$t, border, prnt, rnd$**)**::`float,float,bool,int→none`
  Despliega en pantalla una representación del patrón de difracción de la red. Los valores *t* y *border* determinan el grosor general con el que se dibujará todo y el límite del espacio de dibujo respectivamente, *rnd* indica el número de dígitos después del punto con el que se redondeará el Factor de estructura de cada punto. Si $prnt = True$ se guardará la imagen en la carpeta "image".

A.4. Functions

A.4.1. Funciones sobre átomos en una Base Atómica

- **isitin(**$r, cent, sr, slvl$**)**:: `Lattice,v2d,Lattice,int` $\rightarrow$.
  Verifica qué átomos pertenecientes a la celda de la red r ubicada en la posición *cent*, caen dentro del área correspondiente a la CP de la red sr, loa átomos que cumplen esto son agregados a la BA de sr.

- **megeCut(**r, sr, lvl**)**:: `Lattice,Lattice,int` $\rightarrow$.
  Identifica qué celdas de la red r intersectan la CP de sr y aplica sobre estos la función *isitin*.

- **cleanA(**r, err**)** :: `Lattice,float` $\rightarrow$.
  Verifica los Átomos pertenecientes a la BA de la red r y elimina las repeticiones. La variable err indica un límite de error aceptable, si este no se proporciona se toma 0.001 por default.

- **esClon(**$Atms, atm, \varepsilon$**)**::`[Atom],Atom,float`$\rightarrow$`bool`
  Determina si el Átomo *atom* es equivalente a algún elemento de la lista $Atms$. Esta es una función auxiliar para eliminar posibles átomos repetidos en la BA de alguna red.

- **borders(**$Atms$**)**::`[Atom]`$\rightarrow$`[Atom]`
  Dada una lista de átomos correspondiente a la BA de un red, regresa una sublista con todos aquellos que son cercanos al borde de la CP de dicha red.

- **cleanA(**$Atms, \varepsilon$**)**::`[Atom],float`$\rightarrow$`[Atom]`
  Evalúa una lista de átomos identificando aquellos que son equivalentes por periodicidad con un error delimitado por ε. Todos aquellos que identifica cómo çlon"de alguno ya existente lo quita de la lista original y guarda en una lista nueva que es regresada por esta función.

- **cleanPCell(**L, acc**)**::`Lattice,int`$\rightarrow$`[Atom]`
  Ejecuta la función `cleanA` sobre la báse atómica de la red L con un $\varepsilon = 1/10^{\{acc\}}$.

A.4.2. Cálculo de la Celda Primitiva de una Red

En algunas de estas funciones se hace uso de una lista con formato especifico para evaluar los PR, `pList::[([([(int, int)], v2d)], float)]`, esta es una lista de elementos en pares donde la primera parte se señala el índice de la PR señalada y su vector adjunto, en la segunda parte se señala el *error* calculado para este PR.

- **pts(**l, max_it**)**:: `Lattice,int → pList`.
  Crea una lista con los PR de l contenidos en la supercelda delimitada por max_it. El resultado se regresa en una lista *pts* con el formato *pList*.

- **calcCD(**s, l, ab**)**:: `Lattice,Lattice,(int,int) → (int,int)`.
  Calcula los índices c, d asociados al PR de la red l más cercano al PR de s asociado al índice a, b. Hace uso de la ecuación 4.2.

- **calcPR(**$pts, substrate, l, \varepsilon$**)**:: `pList,Lattice,Lattice,float → pList`.
  Depura la lista *pts* eliminando aquellos cuyo PR en l asociado tiene un error superior al ε dado, si este valor de ε no se proporciona se toma 0.05 por default.

- **commonVs(**Ls, max_val, ε**)**::`[Lattice],int,float→[(int,int),float]`
  Usando cómo red 'sustrato' a la red en la posición 0 de la lista *Ls*, ejecuta secuencialmente la función `calcPR` con cada una de las demás redes en Ls, usando cómo lista pts inicial el resultado de ejecutar la función '`pts(sustrato, max_val)`'. Quedando al final una lista con solamente los PRs que tienen un PR correspondiente en cada capa del sistema con un error inferior a ε.

- **corresponding_points(**$l1, l2, M1$**)**::`Lattice,Lattice,m2x2→m2x2`
  Regresa la matriz 2×2 que hace referencia al PR para $l2$ correspondiente al PR de $l1$ al que referencia $M1$.

- **calc_dd(**Mi, Mo**)**::`m2x2,m2x2→float`
  Calcula el *índice de deformación* correspondiente a *Mi* si esta es deformada hasta ser igual a *Mo*.

A.4.3. Manejo de datos de una Matriz de Trasformación

Los datos de las MTs por mostrar son tomados desde una lista de datos con formato `Mdata::[m2x2,m2x2,float,float,float]`, donde los datos corresponden en orden: la primer matriz correspònde a la MT de la que se va a mostrar información, la segunda es la matriz de deformación asociada a esta MT, los siguientes datos corresponden a la deformación que la matriz de deformación efectúa sobre los VPs de la red, los primeros dos al cambio de su magnitud y los últimos dos al cambio en su dirección.

- **textLonN(**t, n, al**)**::`str,int,str→str`
  Regresa un String con una longitud n cortando o agregando espacios al String t dado. La variable *al* indicará el alineamiento del String resultante en caso de agregar espacios, si $al = l$ los espacios se agregarán a la derecha del texto original, si $al = r$ los espacios se agregarán a la izquierda, si $al = c$ los espacios se agregarán equitativamente a la izquierda y derecha, si no se especifica un valor para *al* se tendrá por default que $al = c$.

- **header()**::→`str`
  Genera el String que es usado cómo encabezado de las tablas con las que se desplegaran los datos de las MTs.

- **infLayer(***L*, *data***)**::`Lattice,Mdata`→`str,int`
  Genera un String desplegando la información contenida en *data* y usando la información de la red *L*. Regresa el string generado y el número de átomos de la supercelda de *L* correspondiente a la MT asociado al *data* dado.

A.4.4. Exportación de una Red desde archivo POSCAR

- **readFile(***name***)**::`str`→`[str]`
  Lee el archivo con nombre *name* (la raíz de la dirección es en la que está código) y tratándolo como un archivo de texto plano transforma su contenido en una lista usando cómo elementos de esta cada línea de texto.

- **leeNumeros(***s***)**::`str`→`[float]`
  Evalúa el texto *s* y regresa una lista con todos los números contenidos en este.

- **importLattice(***name*, *prnt***)**::`str,bool`→`Lattice`
  Crea un objeto Lattice a partir del archivo *name* de formato POSCAR. Si $prnt = True$ se inprime en pantalla un mensaje indicando la creación del objeto Lattice y el nombre del archivo a partir del cual fue creado.

A.4.5. Redes prediseñadas

Estas funciones generan objetos Lattice prediseñados para redes hexagonales, rectangulares y para algunas redes cristalinas comunes.

- **hexa6(***p*, *atms*, *name***)**::`float,[Atom],str`→`Lattice`
  Genera una Red hexagonal con 6 simetrías radiales teniendo como constante de red *p*, *atms* como su base atómica y por nombre *name*. La lista *atms* es opcional, si esta no es dada entonces la red tendrá 2 átomos dentro de su base.

- **hexa3(***p*, *atms*, *name***)**::`float,[Atom],str`→`Lattice`
  Genera una Red hexagonal con 3 simetrías radiales teniendo como constante de red *p*, *atms* como su base atómica y por nombre *name*. La lista *atms* es opcional, si esta no es dada entonces la red tendrá 2 átomos dentro de su base.

- **rectLattice(***p*1, *p*2, *atms*, *name***)**::`float,float,[Atom],str`→`Lattice`
  Genera una Red rectangular con las constantes de red *p*1 y *p*2, los átomos señalados en la lista *atms* como su BA y *name* por nombre. La lista *atms* es opcional, si esta no es proporcionada se generará la red con un solo átomo en el centro.

- **graphen()**::
  Genera una red de Grafeno, con constante de red 2.44Å y con sus átomos centrados generando una red hexagonal con 6 simetrías radiales.

- **graphenC3()**::
  Genera una red de Grafeno, con constante de red 2.44Å y con uno de sus átomos centrado en su base y otro en un vértice, generando una red hexagonal con 3 simetrías radiales.

- **blackPhosphorene()**::
  Genera una red de Fosforeno Negro con las constantes de red 3.2601Å y 4.347Å.

- **h_BN()**::
  Genera una Red que describe la face hexagonal del Nitruro de Boro laminar con constante de red de 2.512Å.

A.5. System

A.5.1. Inicialización

Los objetos de clase *System* modelan las estructuras (sistemas) formadas por apilamiento vertical de redes cristalinas bidimencionales, los **parámetros** de entrada para inicializar esta clase son:

- **lol**::`[Lattice]`
  Lista de redes cristalinas apiladas que conforman el sistema. El orden de la lista determina el orden de apilamiento de las redes, así la red en la posición i de la lista estará inmediatamente sobre la red en la posición $i-1$. La red en la posición 0 de la lista corresponderá a la capa inferior, esta será llamada capa sustrato.

- **name**::`str`
  Nombre con el que se identificará al sistema.

Los **Atributos** propios para esta clase son:

- **redes**::`[Lattice]`
  Lista de redes cristalinas apiladas que conforman el sistema.

- **name**::`str`
  Nombre del sistema.

- **poits**::`[[int,int],float]`
  Lista con todos los PR de la red en la capa sustrato, ligados a los VTs que son comunes para todas las redes del sistema. Al inicializar el objeto es una lista bacía.

- **loMat**::`[m2x2]`
  Lista con las MTs que señalan una posible CP para el sistema. Al inicializar el objeto es una lista bacía.

- **MaxNumM**::`[int]`
  Indica el número máximo de matrices guardados en la lista *loMat*. Al inicializar el objeto este atributo es igual a 10.

- **SuperRed**::`Lattice`
  Guarda la Red que representa al sistema completo en base a la CP calculada. Al inicializar el objeto es "*None*".

- **MT**::`m2x2`
  Guarda la MT que señala la CP calculada. Al inicializar el objeto es "*None*".

A.5.2. Métodos de System

Métodos principales

- **searchLP(***rangeOfSearch*, *epsilon***)**:: `int,float→int`
  Calcula y guarda en el atributo *points* del sistema una lista con todos los PRs comunes para todas las redes del sistema en el área de búsqueda indicada por *rangeOfSearch* con un error menor a *epsilon*.

- **calculateTM(***min_angle***)**:: `bool→int`
  Este método calcula las MTs que indican una CP para el sistema basados en los VTs indicados en la lista *poits* del sistema, las MTs calculadas se guaradán en la lista *loMat*. Si la lista *poits* es vacía, muestra un mensaje de error y regresa -1. Si identifica que todas las redes del sistema son hexagonales, calcula solamente MTs que generan CPs hexagonales y regresa 0. Si **no** todas las redes del sistema son hexagonales regresa 1. La variable *min_angle* indica el ángulo interno mínimo que deben tener las CPs correspondientes a las MTs, por default es 20^o.

- **createSuperLattice(***M***)**:: `m2x2→int`
  Crea y guarda en el atributo del sistema *SuperRed*, la red que represente al sistema completo usando la CP señalada por *M*. Si se creó esta red sin problema se regresa 1, de lo contrario se regresa 0. La red creada con este método puede tener imperfecciones dado que **no** deforma las redes que conforman al sistema.

- **optimize_system(***T*, *prnt***)**:: `m2x2,bool→System,[m2x2]`
  Crea una copia del sistema y deforma las redes de este de forma que, con estas redes deformadas, no se genere imperfecciones al replicar de forma periódica la CP resultante del método *createSuperLattice* sobre este sistema copia deformado con la la MT *T*. Se copian los atributos *SuperRed* y *MT* del sistema copia al sistema original y se regresan el sistema deformado generado y una lista con las matrices de deformación usadas en cada capa de este.

- **ShowTMs(***shw*, *save***)**:: `bool,str→ m2x2`
  Efectúa un análisis sobre todas las MTs en *loMat* obteniendo y regresando aquella MT con un menor grado de distorsión (DD)[2]. Si $shw = True$ imprime en pantalla tablas con las características de las CPs ligadas a las MTs en *loMat*. Brindar la variable *save* es opcional, si esta es dada se creará un archivo de txt con el nombre dado en *save* donde estarán escritas las tablas evaluadas.

- **manualAdjustment(***TMs***)**:: `[m2x2]→ [m2x2]`
  Deforma las redes del sistema de manera que, al transformar las redes con las MTs

[2]Esta medida se calcula con la ecuación 4.10 en la sección Cálculo de Matriz de deformación del capítulo Implementación.

descritas en *TMs*, las superceldas resultantes tengan las mismas forma y dimensión entre si. Las MDs necesarias para esto son regresadas en una lista.

- **show()**:: `none`→ `none`
  Despliega en pantalla una imagen de la CP calculada para el sistema, una representación del la PZB de cada red del sistema y de la CP calculada, además de mostrar la MT usada para calcular la CP.

- **diffractionPattern(***t*, *border*, *prnt***)**:: `float,float,bool`→ `none`
  Despliega en pantalla el Patrón de difracción del sistema a partir de la función *printLightPoits* sobre la red *SuperRed* del sistema con los parámetros *t*, *border* y *prnt* dados.

Métodos rápidos

Estos métodos ejecuta secuencialmente algunos métodos principales para abreviar instrucciones.

- **ejecuta(***n*, *e***)**:: `int,float`→ `m2x2`
  Efectúa secuencialmente las funciones *serchLP, calculateTM* y *ShowTMs*, sin desplegar mensajes en pantalla. Regresa la MT asociada a la CP pequeña con menor distorsión para el área de búsqueda señalada por *n* con error menor a *e*.

- **generateSuperCell(***RoS*, *eps*, *sd***)**:: `int,float,bool`→ `System`
  Efectúa de foerma secuencial las funciones *serchLP*, *calculateTM*, *ShowTMs* y *optimize_system*, regresando un Sistema deformado con su CP calculada a partir de la MT obtenida por la función *ShowTMs*, la cual corresponde a la MT asociada a la CP pequeña con menor distorsión para el área de búsqueda señalada por *RoS* con error menor a *eps*.

Métodos auxiliares

- **clon()**:: `none` → `System`
  Regresa un sistema copia.

- **set_long_loMat(***newMax***)**:: `int`→ `none`
  Indica un nuevo número máximo de MTs guardadas en *loMat*.

- **analyze_T(***T***)**:: `m2x2` → `(m2x2,v2D,int,[Mdata])`
  Analiza la MT *T* y regresa los datos que requiere el método *leeMT* para crear la tabla correspondiente.

- **leeMT(***T*, *prnt*, *shw***)**:: `m2x2,str,bool`→ `str,float`
  Crea una tabla con los datos correspondientes a la CP asociada a la MT *T*, regresa un str con este texto y la DD correspondiente al sistema con esta MT. La variable

prnt es opcional, si este se proporciona, el texto de la tabla es guardado en un archivo txt con ese nombre. Si $shw = True$ la tabla calculada se despliega en pantalla.

- **its_hexagonal_system()**:: `none`→ `bool`
  Si todas las redes del sistema son redes hexagonales regresa *True*.

- **adjust(*M*)**:: `m2x2`→ `int`
  Verifica si la MT dada M puede ser agregada a la lista $loMat$, si el determinante de la matriz es cero entonces regresa 0, de lo contrario regresa 1. Si M puede agregarse a $loMat$ actualiza la lista.

Bibliografía

[1] A. K. Geim and K. S. Novoselov, "The rise of graphene," *Nature materials*, vol. 6, no. 3, pp. 183–191, 2007.

[2] Y. Liu, N. O. Weiss, X. Duan, H.-C. Cheng, Y. Huang, and X. Duan, "Van der waals heterostructures and devices," *Nature Reviews Materials*, vol. 1(9), p. 16042, 2016.

[3] F. H. L. Koppens, T. Mueller, P. Avouris, A. C. Ferrari, M. S. Vitiello, and M. Polini, "Photodetectors based on graphene, other two-dimensional materials and hybrid systems," *Nature Nanotechnology*, vol. 9, p. 780–793, 2014.

[4] A. K. Geim and I. V. Grigorieva, "Van der waals heterostructures," *Nature*, vol. 499, no. 7459, pp. 419–425, 2013.

[5] H. Ibach and H. Lüth, *Solid-state physics: an introduction to principles of materials science*. Springer Science & Business Media, 2009.

[6] S. K. Dutta, S. K. Mehetor, and N. Pradhan, "Metal semiconductor heterostructures for photocatalytic conversion of light energy.," *The Journal of Physical Chemistry Letters*, 2015.

[7] G. Bastard, *Wave mechanics applied to semiconductor heterostructures*. Les Editions de Physique, 2016.

[8] Z. Sun, Z. Yang, J. Zhou, M. H. Yeung, W. Ni, H. Wu, and J. Wang, "A general approach to the synthesis of gold-metal sulfide core-shell and heterostructures," *Angewandte Chemie (International ed. in English)*, vol. 48, no. 16, pp. 2881–2885, 2009.

[9] Y. Ye, Z. J. Wong, X. Lu, X. Ni, H. Zhu, X. Chen, Y. Wang, and X. Zhang, "Monolayer excitonic laser," *Nature Photonics*, vol. 9(11), pp. 733–737, 2015.

[10] F. Withers, O. Del Pozo-Zamudio, A. Mishchenko, A. P. Rooney, A. Gholinia, K. Watanabe, T. Taniguchi, S. J. Haigh, A. K. Geim, A. I. Tartakovskii, and K. S. Novoselov, "Light-emitting diodes by band-structure engineering in van der waals heterostructures. nature materials," *Nature Materials*, vol. 14(3), pp. 301–306, 2015.

[11] G. Grosso and G. P. Parravicini, *Solid state physics*. Academic press, 2013.

[12] Y. Liu, J. N. B. Rodrigues, Y. Z. Luo, L. Li, A. Carvalho, M. Yang, E. Laksono, J. Lu, Y. Bao, H. Xu, S. J. R. Tan, Z. Qiu, C. H. Sow, Y. P. Feng, A. H. C. Neto, S. Adam, J. Lu, and K. P. Loh, "Tailoring sample-wide pseudo-magnetic fields on a graphene–black phosphorus heterostructure," *Nature Nanotech*, vol. 13, pp. 828–834, 2018.

[13] R. G. Parr and Y. Weitao, *Density-Functional Theory of Atoms and Molecules*. Oxford University Press, 1994.

[14] P. Lazić, "Cellmatch: Combining two unit cells into a common supercell with minimal strain.," *Computer Physics Communications*, vol. 197, pp. 324–334, 2015.

[15] T. Necio and M. Birowska, "Supercell-core software: a useful tool to generate an optimal supercell for vertically stacked nanomaterials," *AIP Advances*, vol. 10, no. 10, 2020.

[16] S. Naik, M. H. Naik, I. Maity, and M. Jain, "Twister: Construction and structural relaxation of commensurate moiré superlattices," *Computer Physics Communications*, vol. 271, no. 108184, 2022.

[17] C. DiBona, S. Ockman, and M. Stone, *Open Sources: Voices from the Open Source Revolution*. O'Reilly Media, Inc., 1999.

[18] S. Chazallet, *Python 3 los fundamentos del lenguaje*. Ediciones Eni, 2020.

[19] G. E. Martin, *Transformation geometry: An introduction to symmetry*. Springer Science & Business Media, 2012.

[20] B. Davvaz, *Groups and Symmetry: Theory and Applications*. Singapore: Springer Singapore, 2021.

[21] G. Kresse and J. Furthmüller, "Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set," *Phys. Rev. B*, vol. 54, pp. 11169–11186, 1996.

[22] K. Momma and F. Izumi, "*VESTA3* for three-dimensional visualization of crystal, volumetric and morphology data," *Journal of Applied Crystallography*, vol. 44, no. 6, pp. 1272–1276, 2011.

[23] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*. McGraw-Hill Education, 10 ed., 2014.

[24] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear Programming and Network Flows*. Wiley, 4th ed., 2010.

[25] B. Grünbaum and G. C. Shephard, *Tilings and Patterns*. New York: Dover Publications, 2nd ed., 2016.