UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

CÁLCULO DE PROBABILIDAD DE FORMACIÓN DE PEROVSKITAS BASADAS EN LANTANO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

NATHANIEL MARDONES HUALA

PROFESOR GUÍA:
ALI AKBARI FAKHRABADI

PROFESORA CO-GUÍA:
VIVIANA MERUANE NARANJO

COMISIÓN:
RODRIGO ESPINOZA GONZÁLEZ

Este trabajo ha sido parcialmente financiado por proyecto FONDECYT de iniciación 1200141

SANTIAGO DE CHILE
2022

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: **NATHANIEL MARDONES HUALA**
FECHA: 2022
PROF. GUÍA: ALI AKBARI

CÁLCULO DE PROBABILIDAD DE FORMACIÓN DE PEROVSKITAS BASADAS EN LANTANO

Las celdas de combustible de óxido sólido permiten generar electricidad a partir de hidrógeno y oxígeno, los componentes de éstas deben satisfacer requerimientos específicos de conductividad mixta iónicas y eléctricas para los electrodos, y conductividad iónica simple en el caso de los electrolitos.

Las perovskitas son un tipo de material que cumple dichos requisitos de conductividad, además de alta resistencia térmica, por lo tanto son ampliamente estudiados en sus vasta cantidad de combinaciones posibles, producto de que su estructura permite albergar un dopaje catiónico doble en diferentes porcentajes.

El objetivo del trabajo de titulo consiste en generar una base de datos que contenga los valores de probabilidad de formación y factores de tolerancia para composiciones de perovskita específicas que presenten doble dopaje catiónico y variación en la concentración de dopaje, es decir perovskitas que posean la fórmula química tipo $La_{(1-x)}M_xB_{(1-y)}N_yO_{3-\delta}$.

Los alcances del trabajo incluyen el análisis exclusivo de perovskitas de doble dopaje en los cationes, que posean lantano como catión de sitio A y oxígeno como anión. Además, para cada compuesto se estudiarán las probabilidades de formación para diferentes combinaciones de valores composicionales (con $0.1 \leq x, y \leq 0.9 \ \ \wedge \ \ 0 \leq \delta \leq 0.5$) y los elementos específicos para cada sitio según corresponda.

El trabajo inicia con una etapa de revisión bibliográfica, para seguir luego con el trabajo digital, el cuál está estructurado en las etapas de: preparación, desarrollo de código, ejecución, obtención de resultados y validación de resultados, dando lugar así a la base de datos final.

Como resultados del trabajo se logró generar 15 470 archivos de compuestos de perovskitas, donde cada uno contiene la información de probabilidades de formación y cálculo de factores de tolerancia para diferentes compuestos. Todos los archivos generados durante el trabajo están disponibles en el repositorio de [19].

Agradecimientos

Quiero dar las gracias a todas las personas que conocí en mi paso durante la universidad y en especial a mi grupo más cercano del final, Seba, Javier, Paula, Ralf, Emiliano, Victor, Byron, Carlota y Cristóbal, muchas gracias por todas las risas. También a mis demás compañeros queridos y con los cuales compartimos muchas veces en la facultad, Octavio, Samuel y Stefan. Adrián, pasó, aquí estoy.

Agradecer especialmente a la Cata por su valiosísima compañía durante estos últimos años, gracias por haber estado siempre ahí y por ser quien eres, lo logramos. Agradezco también a toda mi familia por apoyarme y permitirme llegar hasta este punto.

Finalmente muchas gracias a los profesores de mi comisión, por siempre estar disponibles a resolver cualquiera de mis preguntas y brindarme asistencia durante el proceso de trabajo de título.

Tabla de Contenido

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

Las celdas sólidas de combustible (SOFC, Solid Oxide Fuel Cell) son dispositivos electroquímicos capaces de entregar energía eléctrica a partir de una reacción química, la cual involucra hidrógeno y oxígeno como combustibles, mientras que se tienen agua y electricidad como productos. Los materiales que componen a las celdas sólidas consisten en principalmente en cerámicos, ya que de éstos destacan sus capacidades físicas específicas tales como resistencia a altas temperaturas.

En los últimos años se ha puesto foco en la posibilidad de utilizar un tipo de material denominado "*Perovskita*". Este material es un tipo de cerámico, ideal para este tipo de aplicaciones, dada la resistencia a altas temperaturas y conductividad eléctrica que estas poseen, sin embargo es necesario ajustar algunas propiedades físicas de estos materiales, como por ejemplo el gran porcentaje de expansión térmica que algunas composiciones tienen.

Para esto se recurre al dopaje de la estructura base, con diferentes elementos y porcentajes composicionales, lo cual permite modificar las propiedades físicas de las perovskitas según los requerimientos que se tengan.

Al momento de realizar dopaje sobre las estructuras de perovskita, se cuenta con un amplio número de elementos posibles los cuales se puede utilizar, es así como se genera entonces un gran número de combinaciones de elementos, que pueden dar lugar a estructuras de tipo perovskita.

La cantidad de posibles combinaciones de elementos es tal, que no es posible fabricar cada una de estas para verificar la estabilidad de las estructuras. En su lugar, es necesario recurrir a fórmulas que permitan predecir el comportamiento de las estructuras de perovskita a formar.

Las fórmulas que se utilizan son los llamados "factores de tolerancia" los cuales, a partir de parámetros físicos de los elementos que conforma la perovskita, entregan una valor indicador de la estabilidad de la estructura de perovskita a fabricar.

1.1. Objetivos

Generar una base de datos que contenga los valores de probabilidad de formación y factores de tolerancia para compuestos teóricos de perovskita. Esto permitirá comprobar la existencia teórica de composiciones de perovskitas que aun no han sido desarrolladas en laboratorio.

Los objetivos específicos para esto son:

- Adaptar el esquema original para el análisis de las perovskitas objetivo.
- Generar la base de datos final.
- Analizar la base de datos final, utilizando el esquema de funciones original y el modificado.
- Consolidar archivos y subirlos a la web.

1.2. Alcances

Para la realización del trabajo, se fijan los siguiente alcances:

- Sólo se trabajarán con perovskitas del tipo $La_{(1-x)}M_xB_{(1-y)}N_yO_{3-\delta}$ mientras que M, B y N corresponde a elementos variables.
- Se calcularán los valores de nuevo factor de tolerancia y probabilidad de formación, (τ y P(τ)) para cada uno de los compuestos candidatos que se generen.
- Se entregarán los resultado en formato .csv junto a los demás archivos utilizados para la obtención de los resultados.

Capítulo 2

Antecedentes

2.1. Celdas de combustible

Las celdas de combustible de óxido sólido (SOFC por sus siglas en inglés, Solid Oxide Fuel Cell) son dispositivos de generación de energía eléctrica, mediante una reacción electroquímica a altas temperaturas. Como elementos reactivos se tiene hidrógeno, oxígeno y la energía térmica que se debe proporcionar al sistema, mientras que como productos se obtiene energía eléctrica y agua en estado gaseoso.

A continuación se presenta la descripción de cada una de las partes que componen una celda de combustible, las cuales son: Ánodo, Electrolito y Cátodo.

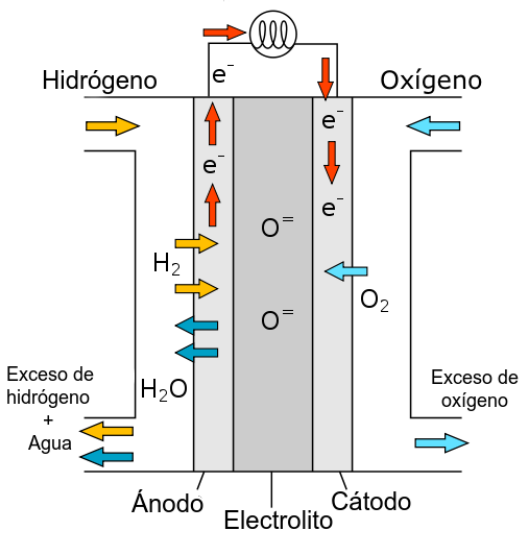


Figura 2.1: Partes principales de una celda de combustible. [2]

- **Cátodo:** Sección de material que es encuentra cargada con electrones, para así inducir la reducción del oxígeno, compuesto con el cuál está en contacto. Como producto de

la reacción de reducción, las moléculas de oxígeno O_2 se separan, generando iones de oxígeno O^-, los cuales se desplazan desde la sección del cátodo hacia el ánodo, pasando por el electrolito. En conclusión, el cátodo debe ser un buen conductor de electrones y iones. Una composición conocida que funciona como cátodo en este tipo de aplicaciones es la perovskita $La_{0.6}Sr_{0.4}CoO_{3-\delta}$.

- **Electrolito:** Permite el paso de iones de oxígeno desde el cátodo hacia al ánodo, al mismo tiempo que impide el flujo de electrones desde el ánodo, por lo tanto, debe ser un material que cumpla las siguientes características: ser conductor de iones, ser aislante eléctrico, y lo suficientemente denso para impedir el paso de gases. Una composición típica de electrolito puede ser la zirconia estabilizada por ytrio.

- **Ánodo:** Es la capa opuesta al cátodo, ya que se encarga de oxidar al hidrógeno con el cuál está en contacto. El ánodo, por el extremo del electrolito, recibe los iones de oxígeno O^-, los cuales al reaccionar con el hidrógeno liberan un par electrónico, al mismo tiempo de que se forma agua en estado gaseoso $H_2O_{(g)}$. Los electrones libres que quedan de la reacción, viajan al cátodo nuevamente, mediante un puente de conexión ánodo - cátodo, generando una corriente eléctrica y completando así el ciclo de funcionamiento. El material a utilizar para este tipo de aplicaciones debe ser catalizador, conductor tanto de iones como de electrones y poroso para facilitar la reacción hidrógeno - oxígeno. Para estas aplicaciones, se utiliza níquel, mezclado con materiales electrolíticos.

2.1.1. Calor en funcionamiento

Una de las desventajas, que tienen las celdas de combustible, es el hecho de que necesitan de muy altas temperaturas de operación (entre 500 °C y 1000 ° C). Esto permite activar la migración de los iones de oxígenos y facilitar las reacciones electroquímicas de ambos electrodos. Se requiere un buen sistema de disipación de calor en la celda además de materiales específicos que sean capaces de resistir tales temperaturas, para evitar fallas o destrucción de la celda misma.

2.1.2. Fabricación y aplicaciones

La fabricación de las celdas de combustibles incluye el trabajo por secciones, donde con polvos cerámicos y tratamientos de sinterización, se da lugar a cada una de las capas (cátodo, electrolito y ánodo) las cuales finalmente son unidas entre sí, formando así la celda de combustible.

Respecto a la utilización de las celdas de combustible, estas se utilizan conectadas en serie, donde el número de celdas a utilizar depende de manera directa de la aplicación deseada, mientras mayor sea la demanda eléctrica, mayor será el número de celdas conectadas en serie. La conexión de varias celdas de combustibles da lugar a un *stack* de celdas de combus-

tible,como se puede observar en la figura 2.2 a continuación.

Figura 2.2: Stack de celdas de combustible. [3]

Eventualmente, mediante la conexión de suficientes celdas de combustible, se puede dar lugar a grandes potencias eléctricas, donde incluso se tienen ejemplos de la utilización de celdas de combustible en el contexto de alimentación industrial, lo cual permitiría dar lugar a un amplio rango de aplicaciones.

2.2. Materiales cerámicos

Son materiales inorgánicos, que están compuestos principalmente por elementos no metálicos, aunque también se pueden encontrar cerámicos con elementos metálicos en su estructura. Su estructura se denota de la forma AX donde A es un catión y X el anión de la estructura. Típicamente sus métodos de fabricación involucran tratamientos térmicos a altas presiones, sobre algún conjunto de polvos, proceso también conocido como sinterizado.

Entre las principales características de los materiales cerámicos, se tienen las propiedades físicas, las cuales dependen de los elementos que componen al cerámico. Típicamente un material cerámico posee las siguientes propiedades; alta dureza, gran resistencia a la compresión, resistencia al desgaste y a la corrosión, alto punto de fusión y ser aislantes térmicos y eléctricos. Sin embargo, pese a todas estas capacidades, son un material frágil, lo que significa que no resisten las cargas de impacto.

La micro estructura de los cerámicos varía dependiendo del material, dicho esto se pueden encontrar cerámicos con una estructura: cristalina, no cristalina, así también como una combinación de las dos.

En la actualidad se han desarrollado diferentes tipos de cerámicos, donde se seleccionan de manera específica los elementos que los componen, lo cual afecta de gran manera las propiedades físicas del cerámico, logrando así controlar y elaborar cerámicas específicas según lo exija la aplicación. Por último mencionar que las aplicaciones de los materiales cerámicos son variadas, las cuales van desde elaboración de ornamentos decorativos, hasta aplicaciones demandantes como lo son los discos de freno cerámicos.

2.3. Perovskitas

Las perovskitas son un tipo de material cerámico, cuya fórmula química es de la forma ABX_3: A y B corresponden a cationes, de tamaño mayor y menor respectivamente, el elemento X corresponde al anión, que típicamente es oxígeno, sin embargo en algunas ocasiones se pueden encontrar cloro o incluso flúor tomando el lugar del anión X.

Las perovskitas poseen una serie de propiedades físicas y químicas que las destacan por sobre otros materiales, entre estas se encuentra superconductividad, conductividad iónica, y diferentes propiedades dieléctricas. Este gran número de propiedades abre la posibilidad de que las perovskitas sean utilizadas en las ya descritas, celdas de combustible.

Los elementos que pueden tomar la posición A o la posición B están definidos según la siguiente distribución que se observa en la figura 2.3.

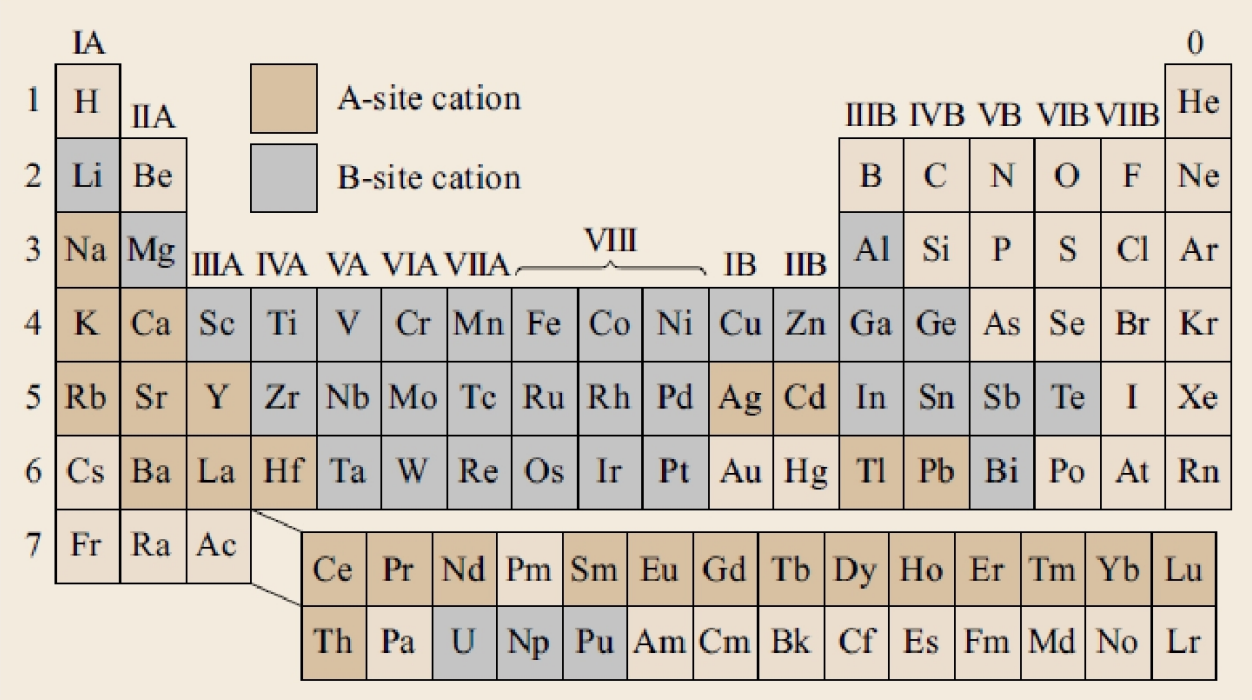


Figura 2.3: Distribución de diferentes cationes posibles. [4]

Según la figura, se tienen 27 elementos tipo A y 36 elementos tipo B, los cuales pueden dar lugar a casi mil combinaciones $A - B$ para conformación de perovskitas. Pese al gran número de combinaciones posibles, no todas las combinaciones de elementos son compatibles para formar una estructura estable de perovskita y es necesario estudiar cuales estructuras son más probables de formar.

Es importante notar también que cada elemento modificará las propiedades de las perovskitas de distinta manera, por lo tanto es necesario analizar el impacto que cada elemento

tiene sobre las propiedades finales de la perovskita.

2.3.1. Variaciones en micro estructura de perovskitas

Las perovskitas poseen una estructura cristalina, lo que implica un empaquetamiento ordenado de los átomos que componen la estructura. El arreglo base de las perovskitas es el arreglo cúbico, de donde se puede extraer la estructura secundaria de octaedro, formados por los átomos B y X y que se ordenan de manera tal que comparten sus esquinas B.

A continuación en se observan las estructuras octaédricas que corresponden a los volúmenes verdes, los átomos rojos son elemento B, los átomos verdes son X y los átomos amarillos corresponden al elemento A.

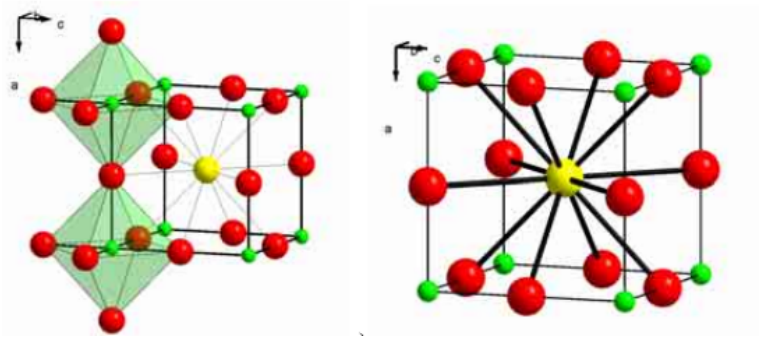


Figura 2.4: Estructura cristalina cúbica de perovskitas. [5]

En la estructura basal de la perovskita se observa una disposición simétrica con los octaedros orientados de manera vertical, pero estas características en la estructura pueden cambiar a medida que la relación de tamaño entre los átomos varía, viéndose así una inclinación en los octaedros.

La inclinación de los octaedros implica una deformación en la red, la cual al superar cierto nivel, se tiene un cambio en la micro estructura, donde pasa de ser una cúbica, a poseer una estructura ortorómbica. Por otro lado, si la deformación es inferior a cierto margen, la estructura varía desde una cúbica a otra hexagonal.

Ambos tipos de modificaciones sobre la estructura, se observan en la figura 2.5. A la izquierda, un arreglo ortorómbico y a la derecha el hexagonal.

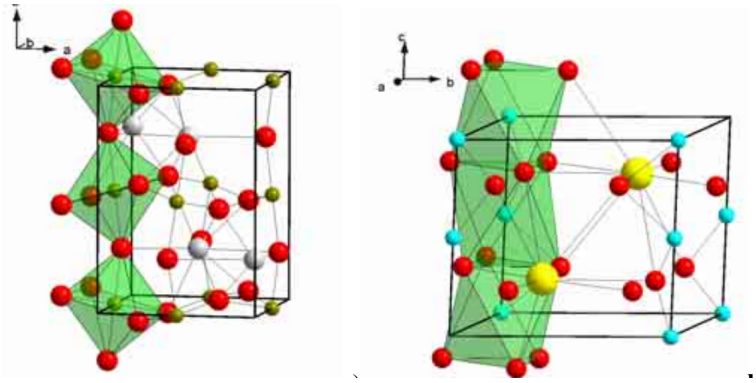


Figura 2.5: Estructuras cristalinas modificadas. [5]

2.3.2. Dopaje de perovskitas

Además de las perovskitas base mencionadas de tipo ABX_3, también se tienen modificaciones sobre la cantidad de elementos que componen la perovskita, mediante el dopaje de la estructura. Este procedimiento se realiza principalmente se sobre los cátodos, tanto de manera aislada (sólo A, o sólo B) como también de manera simultánea sobre ambos, sobre A y B.

Este tipo de perovskitas presenta se observa en la figura 2.6 donde se tiene un dopaje único sobre los cationes de sitio B, provocando así la presencia de celdas octaédricas intercaladas (estructuras verde y azul).

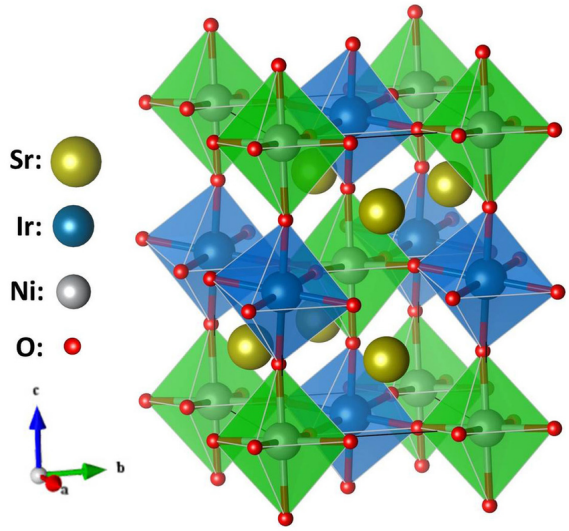


Figura 2.6: Microestructura de perovskita $SrIr_{0.5}Ni_{0.5}O_3$ con dopaje sobre sitio B con elemento Ni sobre elemento Ir. [6]

En el caso de un dopaje simultáneo sobre ambos cationes A y B, además de poseer estructuras de octaedro intercaladas, los cationes A intermedios también estarán intercalados, generando una sustitución de éstos en función del porcentaje del dopaje. En la figura de recién, esto se observaría como una sustitución sobre los átomos de Sr, amarillos.

Se desprende entonces que al momento de definir un elemento dopante, también es necesario fijar el porcentaje de composición con el cual este elemento se incorpora en la estructura.

Los valores de porcentaje de composición están en el intervalo [0, 1] y se representan mediante la incorporación de un subíndice para el elemento dopante asociado, en la fórmula química. A continuación se tiene la fórmula química final una perovskita de doble dopaje, los sub índices de porcentaje composicional se denominan con las variables x y δ.

$$A_{(1-x)}A'_xB_{(1-y)}B'_yX_{(3-\delta)} \tag{2.1}$$

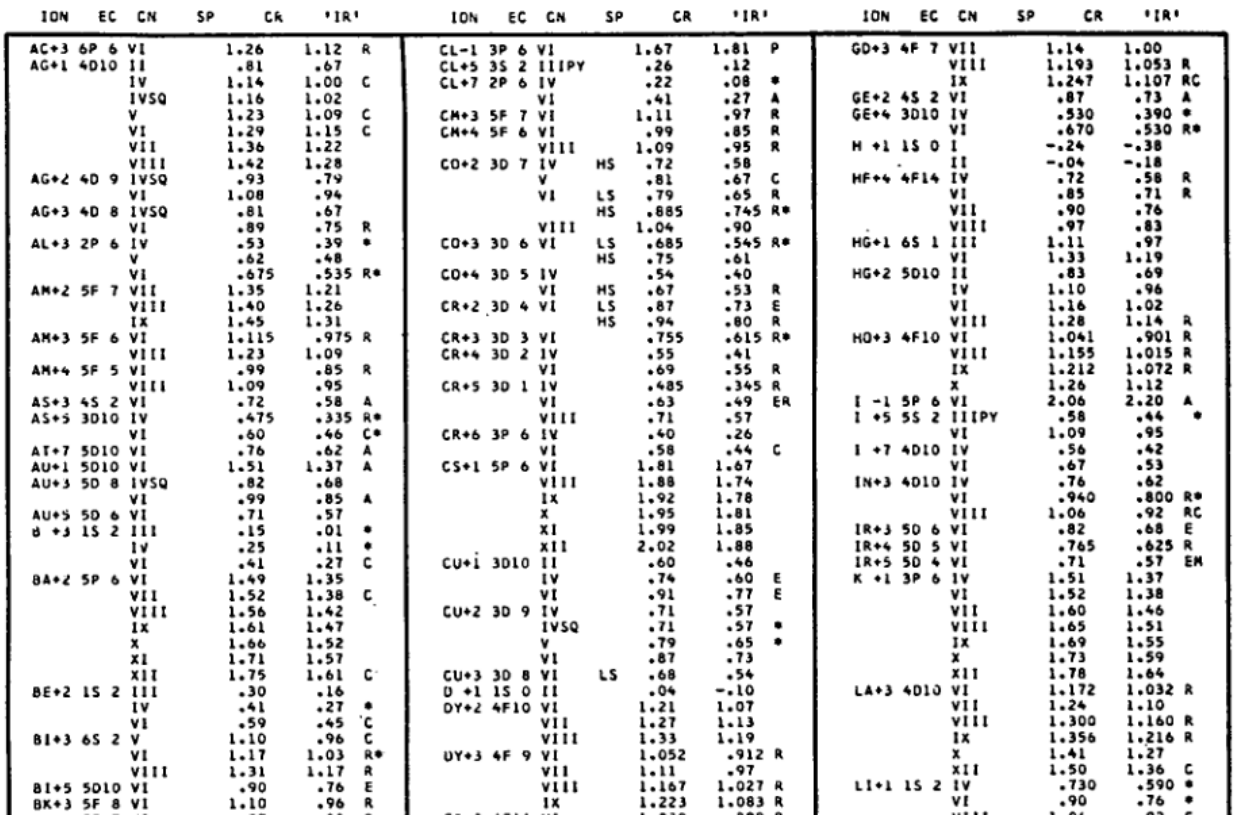
.

Donde:

- A: Catión base de sitio A
- A': Catión dopante de sitio A
- B: Catión base de sitio B
- B': Catión dopante de sitio B
- X: Anión

2.4. Radios iónicos de Shannon

Los radios iónicos de Shannon, consisten en los valores numéricos de radio iónico obtenidos a partir del estudio "*Revised effective ionic radii and systemaatic studies of interatomic distances in halides and chalcogenides*" [1], donde se obtuvieron los radios iónicos de diferentes elementos en diferentes configuraciones electrónicas, elaborando así una tabla de datos, como se observa en la figura a continuación, con la información obtenida a partir del estudio.



ION	EC	CN	SP	CR	'IR'	
AC+3	6P 6	VI		1.26	1.12	R
AG+1	4D10	II		.81	.67	
		IV		1.14	1.00	C
		IVSQ		1.16	1.02	
		V		1.23	1.09	C
		VI		1.29	1.15	C
		VII		1.36	1.22	
		VIII		1.42	1.28	
AG+2	4D 9	IVSQ		.93	.79	
		VI		1.08	.94	
AG+3	4D 8	IVSQ		.81	.67	
		VI		.89	.75	R
AL+3	2P 6	IV		.53	.39	*
		V		.62	.48	
		VI		.675	.535	R*
AM+2	5F 7	VII		1.35	1.21	
		VIII		1.40	1.26	
		IX		1.45	1.31	
AM+3	5F 6	VI		1.115	.975	R
		VIII		1.23	1.09	
AM+4	5F 5	VI		.99	.85	R
		VIII		1.09	.95	
AS+3	4S 2	VI		.72	.58	A
AS+5	3D10	IV		.475	.335	R*
		VI		.60	.46	C*
AT+7	5D10	VI		.76	.62	A
AU+1	5D10	VI		1.51	1.37	A
AU+3	5D 8	IVSQ		.82	.68	
		VI		.99	.85	A
AU+5	5D 6	VI		.71	.57	
B +3	1S 2	III		.15	.01	*
		IV		.25	.11	*
		VI		.41	.27	C
BA+2	5P 6	VI		1.49	1.35	
		VII		1.52	1.38	C
		VIII		1.56	1.42	
		IX		1.61	1.47	
		X		1.66	1.52	
		XI		1.71	1.57	
		XII		1.75	1.61	C
BE+2	1S 2	III		.30	.16	
		IV		.41	.27	*
		VI		.59	.45	C
BI+3	6S 2	V		1.10	.96	C
		VI		1.17	1.03	R*
		VIII		1.31	1.17	R
BI+5	5D10	VI		.90	.76	E
BK+3	5F 8	VI		1.10	.96	R

ION	EC	CN	SP	CR	'IR'	
CL-1	3P 6	VI		1.67	1.81	P
CL+5	3S 2	IIIPY		.26	.12	
CL+7	2P 6	IV		.22	.08	*
		VI		.41	.27	A
CM+3	5F 7	VI		1.11	.97	R
CM+4	5F 6	VI		.99	.85	R
		VIII		1.09	.95	R
CO+2	3D 7	IV	HS	.72	.58	
		V		.81	.67	C
		VI	LS	.79	.65	R
			HS	.885	.745	R*
		VIII		1.04	.90	
CO+3	3D 6	VI	LS	.685	.545	R*
			HS	.75	.61	
CO+4	3D 5	IV		.54	.40	
		VI	HS	.67	.53	R
CR+2	3D 4	VI	LS	.87	.73	E
			HS	.94	.80	R
CR+3	3D 3	VI		.755	.615	R*
CR+4	3D 2	IV		.55	.41	
		VI		.69	.55	R
CR+5	3D 1	IV		.485	.345	R
		VI		.63	.49	ER
		VIII		.71	.57	
CR+6	3P 6	IV		.40	.26	
		VI		.58	.44	C
CS+1	5P 6	VI		1.81	1.67	
		VIII		1.88	1.74	
		IX		1.92	1.78	
		X		1.95	1.81	
		XI		1.99	1.85	
		XII		2.02	1.88	
CU+1	3D10	II		.60	.46	
		IV		.74	.60	E
		VI		.91	.77	E
CU+2	3D 9	IV		.71	.57	
		IVSQ		.71	.57	*
		V		.79	.65	*
		VI		.87	.73	
CU+3	3D 8	VI	LS	.68	.54	
D +1	1S 0	II		.04	-.10	
DY+2	4F10	VI		1.21	1.07	
		VII		1.27	1.13	
		VIII		1.33	1.19	
DY+3	4F 9	VI		1.052	.912	R
		VII		1.11	.97	
		VIII		1.167	1.027	R
		IX		1.223	1.083	R

ION	EC	CN	SP	CR	'IR'	
GD+3	4F 7	VII		1.14	1.00	
		VIII		1.193	1.053	R
		IX		1.247	1.107	RC
GE+2	4S 2	VI		.87	.73	A
GE+4	3D10	IV		.530	.390	*
		VI		.670	.530	R*
H +1	1S 0	I		-.24	-.38	
		II		-.04	-.18	
HF+4	4F14	IV		.72	.58	R
		VI		.85	.71	R
		VII		.90	.76	
		VIII		.97	.83	
HG+1	6S 1	III		1.11	.97	
		VI		1.33	1.19	
HG+2	5D10	II		.83	.69	
		IV		1.10	.96	
		VI		1.16	1.02	
		VIII		1.28	1.14	R
HO+3	4F10	VI		1.041	.901	R
		VIII		1.155	1.015	R
		IX		1.212	1.072	R
		X		1.26	1.12	
I -1	5P 6	VI		2.06	2.20	A
I +5	5S 2	IIIPY		.58	.44	*
		VI		1.09	.95	
I +7	4D10	IV		.56	.42	
		VI		.67	.53	
IN+3	4D10	IV		.76	.62	
		VI		.940	.800	R*
		VIII		1.06	.92	RC
IR+3	5D 6	VI		.82	.68	E
IR+4	5D 5	VI		.765	.625	R
IR+5	5D 4	VI		.71	.57	EM
K +1	3P 6	IV		1.51	1.37	
		VI		1.52	1.38	
		VII		1.60	1.46	
		VIII		1.65	1.51	
		IX		1.69	1.55	
		X		1.73	1.59	
		XII		1.78	1.64	
LA+3	4D10	VI		1.172	1.032	R
		VII		1.24	1.10	
		VIII		1.300	1.160	R
		IX		1.356	1.216	R
		X		1.41	1.27	
		XII		1.50	1.36	C
LI+1	1S 2	IV		.730	.590	*
		VI		.90	.76	*

Figura 2.7: Extracto de tabla con datos de información descrita desde [1]

Estos valores son muy utilizados al momento de realizar investigaciones de materiales, y constituyen el estándar en los estudios que involucren la utilización de información de radio iónico de los elementos. Con el tiempo se han incorporado nuevos valores de radio iónico, en base a diversos estudios que se han realizado en relacionados al descubrimiento de nuevos estados electrónicos para los elementos.

2.5. Factores de tolerancia de perovskitas

Son valores numéricos que se obtienen a partir de propiedades físicas de los elementos que conforman la perovskita. Se tienen dos factores y se denominan; *Factor de tolerancia de Goldschmidt (t)* y el recientemente desarrollado *Nuevo factor de tolerancia (τ)*. A continuación se tiene el detalle de cada uno de ellos.

2.5.1. Factor de Tolerancia de Goldschmidt (FTG)

El factor de Goldschmidt es la base de cualquier estudio relacionado a las estructura de perovskitas. Este valor consiste una una relación geométrica, donde se consideran los radios iónicos de los elementos que conforman la perovskita, para entregar un valor adimensional "t" que describe entonces la micro estructura aproximada de la perovskita. A continuación se tiene la ecuación con la cual se obtiene este factor.

$$t = \frac{r_A + r_X}{(r_B + r_X)\sqrt{2}} \tag{2.2}$$

Donde:

- r_A: radio iónico del catión de sitio A
- r_B: radio iónico del catión de sitio B
- r_X: radio iónico del anión X

Los valores que se obtienen con este coeficiente "t" son cercanos al número uno, donde diferentes rangos de valores están asociados a diferentes grados de distorsión en la red y en consecuencia, distintos tipos de estructuras. La escala del tipo de estructura según el valor numérico se tiene a continuación:

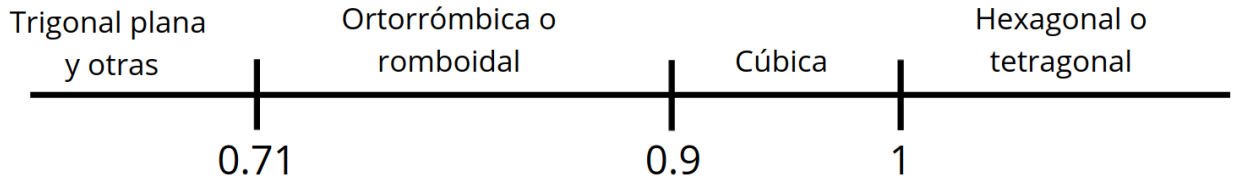


Figura 2.8: Recta de valores para el factor de tolerancia de Goldschmidt. [7]

Este factor es utilizado en algunas aplicaciones como indicador respecto a que tan probable es la formación de perovskita a partir de un conjunto de elementos arbitrario, donde

llega a obtener 72 % de efectividad al momento de predecir la formación de una perovskita [8].

2.5.2. Nuevo factor de tolerancia (NFT)

El nuevo factor de tolerancia es una ecuación reciente, que se obtuvo a partir de la investigación del trabajo "*New tolerance factor to predict the stabilityof perovskite oxides and halides. Science Advance*" [8]. Este factor se utiliza para el cálculo de probabilidad de formación de una perovskita.

Al igual que el factor de Goldschmidt, el nuevo factor de tolerancia toma propiedades físicas de los elementos que componen la perovskita y mediante una relación matemática, entrega un valor decimal que se ubica generalmente en el rango [1, 40]. La fórmula para la obtención del factor de tolerancia (tau) se muestra a continuación:

$$\tau = \frac{r_X}{r_B} - n_A(n_A - \frac{r_A/r_B}{ln(r_A/r_B)}) \tag{2.3}$$

Donde:

- r_A: radio iónico del catión de sitio A
- r_B: radio iónico del catión de sitio B
- r_X: radio iónico del anión X
- n_A: estado de oxidación del catión de sitio A en el compuesto

De la ecuación, se observa que el término r_b/r_x corresponde al factor octaedral μ el cual indica la estabilidad del octaedro de la perovskita.

En el estudio de [8], además de utilizar la ecuación 2.3, se realiza otro procedimiento de ajuste estadístico, donde se utiliza un valor τ de una perovskita conocida, para calibrar los resultados en una escala de probabilidad y así obtener los valores de probabilidad de formación finales.

Dado esto, se llega a la conclusión de que a diferencia del factor de Goldschmidt, no se tiene una escala para caracterizar el resultado de τ de manera directa, en cambio, es necesario evaluar el valor τ entregado en una escala mencionada.

2.5.2.1. Probabilidad de formación

Para obtener el valor de probabilidad de formación de un compuesto teórico de perovskita, es necesario asignar un valor probabilístico a los resultados de τ que se obtiene con la

ecuación 2.3, este procedimiento se conoce como ajuste probabilístico y en específico, para este caso se utiliza el método de escalamiento de Platt.

En primer lugar cabe mencionar que el valor τ posee un comportamiento monótono decreciente con respecto a la estabilidad de la perovskita analizada, esto implica que mientras mayor sea el valor que se obtiene para τ menor será la estabilidad del compuesto. En específico se tiene el valor umbral 4.18, que permite entonces obtener la clasificación binaria como: perovskita (si $\tau < 4.18$) o no perovskita (si $\tau > 4.18$).

Gracias a estos dos puntos, de comportamiento monótono de τ y la clasificación binaria del factor, se utiliza entonces el modelo de regresión logarítmico para el rango continuo de valores resultado τ que se obtiene para los diferentes compuestos, donde finalmente, se les asigna un valor de probabilidad continua, que es estimado como el valor de probabilidad de formación de las perovskitas.

A continuación se tiene un gráfico que esquematiza la variación de probabilidad de formación para los valores de τ entre 1 y 8. Este gráfico se obtuvo con los valores de las muestras de compuestos utilizadas en [8] (TableS1.csv).

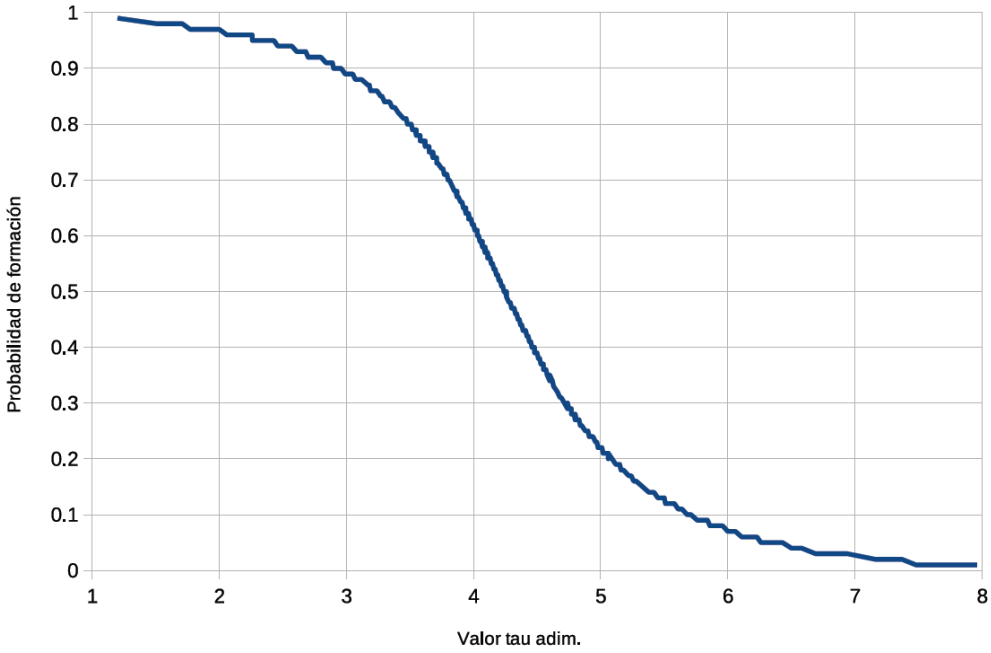


Figura 2.9: Gráfico de probabilidad acumulada para τ elaborado con los datos de [8].

2.6. Ciencia de datos en materiales

El concepto de ciencia de datos hace referencia a la utilización de múltiples ciencias; estadística, probabilidades, métodos científicos e inteligencia artificial, los cuales se aplican en conjunto, para efectuar un análisis detallado de grandes cantidades de datos, para extraer información implícita, aumentando el valor a los datos.

El objetivo principal de la ciencia de datos en materiales es generar conocimiento y nuevos descubrimientos respecto a materiales, sin la necesidad de tantos experimentos físicos en laboratorio, lo cual permite agilizar los procesos de investigación.

Para trabajar con ciencia de datos, es esencial contar con una base de datos sobre la cual se realizan diferentes tipos de análisis, en el caso del trabajo con materiales, las bases de datos a utilizar pueden incluir información respecto a elementos y compuestos, algunos de estos datos se pueden encontrar radios atómicos, masa molar, tipo de micro estructura, etc.

En este trabajo se obtendrá entonces una base de datos que entregue la información de estabilidad de perovskitas. Esto permitirá sentar las bases para diferentes estudios a futuro relacionados con perovskitas.

Capítulo 3

Metodología

3.1. Perovskitas a estudiar

Como ya se ha adelantado, se trabajará en torno a los compuestos cerámicos denominados "*Perovskitas*". El estudio se limitará a un grupo específico de estos materiales, el cual tiene las siguientes características: Doble dopaje, Variación composicional y Lantano como catión base de sitio A, cada una de éstas se explica a continuación.

- **Doble dopaje :** La estructura base de las perovskitas (ABX_3) puede verse modificada mediante el dopaje sobre los elementos A y B. En este caso, se analizarán las estructuras que exhiban un dopaje doble sobre ambos cationes A y B, de manera simultánea, resultando en perovskitas con los elementos AA'BB'X.
- **Variación composicional :** En el caso de tener un dopaje en la estructura de perovskita, el porcentaje composicional con la cual se inserta este nuevo elemento sobre la estructura base puede variar, lo cual incide sobre la estabilidad final del compuesto de perovskita. Este es justamente el efecto que se analizará en las perovskitas, donde se harán variar los subíndices x, y, δ de la fórmula $A_{1-x}A'_xB_{1-y}B'_yX_{3-delta}$ de las perovskitas a analizar.
- **Lantano como catión base de sitio A :** Las perovskitas a estudiar tendrán el lantano como elemento base en su composición, en específico como catión base de sitio A. Serán los otros tres cationes: A', B y B' los que se harán variar con diferentes elementos.

Por lo tanto, la fórmula química de los compuestos de perovskita a analizar tiene la siguiente forma:

$$La_{(1-x)}M_xB_{(1-y)}N_yO_{3-\delta} \tag{3.1}$$

Donde:

- La: Lantano, catión base de sitio A
- (1-x): Porcentaje composicional de elemento La
- M: Catión dopante de sitio A
- x: Porcentaje composicional de elemento M
- B: Catión base de sitio B
- (1-y): Porcentaje composicional de elemento B
- N: Catión dopante de sitio B
- y: Porcentaje composicional de elemento N
- O: Oxígeno, anión
- δ : Porcentaje composicional de oxígeno

Los elementos M se pueden extraer desde los cationes de sitio A, que se presentaron en la tabla periódica de la figura 2.3 de *Antecedentes*, mientras que los elementos B y N pueden ser cualquiera de los cationes de sitio B, de la misma figura.

Los intervalos en los cuales se definirán los valores composicionales $x - y - \delta$ se definen según la lista a continuación:

- $0.1 \leq x \leq 0.9$
- $0.1 \leq y \leq 0.9$
- $0 \leq \delta \leq 0.5$

Al fijar el límite inferior de los intervalos composicionales con 0.1 para x e y, se eliminan los casos de compuestos sin dopaje.

Ejemplos con este tipo de estructura

A continuación se tienen algunos ejemplos de este tipo de composiciones de perovskitas, junto a algunas propiedades de cada uno de ellos:

- $La_{0.6}Sr_{0.4}Co_{0.1}Fe_{0.9}O_{3-\delta}$: Es un material semiconductor a temperatura ambiente. Mediante la sustitución del 40 % de los cationes de sitio A con estroncio (Sr), se aumenta la conductividad iónica y electrónica. Por otro lado, mediante el dopaje con hierro (Fe), se reduce la dilatación térmica del compuesto y aumenta su conductividad, resultando en un elemento ideal para la aplicación como cátodo en una celda de combustible sólidas. [9]
- $La_{0.6}Sr_{0.4}Ni_{0.1}Fe_{0.9}O_{3-\delta}$: Posee características similares al compuesto de cobalto recién descrito, sin embargo, la diferencia que tiene es que el níquel surge como un alternativa de menor costo que el cobalto. Posee baja resistencia a la polarización, y en general es un buen candidato para su utilización en celdas de combustible sólidas. [9]
- $La_{0.6}Sr_{0.4}Co_{0.2}Fe_{0.8}O_{3-\delta}$: Se ha estudiado la síntesis de este tipo de compuesto, utilizando técnicas de ultrasonido y secado a altas temperaturas, logrando así la fabricación del compuesto de manera exitosa, probando así su composición estable. [10]

3.2. Nuevo Factor de Tolerancia

Como ya se presentó anteriormente, el nuevo factor de tolerancia es una ecuación que se utiliza para obtener una caracterización de la estabilidad de las perovskitas. Esta fórmula se aplicará sobre cada uno de los compuestos de perovskitas a estudiar, obteniendo así el valor de probabilidad de formación deseado.

Esta ecuación es fundamental para el desarrollo del trabajo, y es el resultado del trabajo "*New tolerance factor to predict the stability of perovskite oxides and halides. Science Advances*" [8] utilizando un método de aprendizaje automático denominado SISSO.

3.3. Ponderación de propiedades atómicas

Es necesario adaptar el funcionamiento de la fórmula del Nuevo Factor de Tolerancia para perovskitas que presenten doble dopaje, es decir encontrar algún método que permita ponderar los radios de dos cationes en un único valor, lo cual permite el cual pueda ser utilizado en la fórmula del Nuevo Factor de Tolerancia τ.

Para esto se utilizó el método planteado en el estudio "*Can we predict the formability of perovskite oxynitrides from tolerance and octahedral factors?*" [11], donde se realizan diferentes análisis sobre estructuras de perovskitas con dopaje. Finalmente se utiliza la fórmula de media geométrica, para ponderar propiedades físicas de múltiples cationes, obteniendo un único valor que considera los múltiples dopajes que se puedan tener en el sitio.

Esta interrogante también se abordó en el estudio de [8], donde en el contexto de la validación de la ecuación, se estudiaron entonces dos métodos para ponderar los radios de múltiples cationes, específicamente los métodos de media geométrica y media aritmética.

Se concluyó que independiente del tipo de media que se utilice, los resultados obtenidos se mantenían, optando así por la utilización de la media aritmética simple, dada su simplicidad. Finalmente se utilizarán las siguiente fórmulas de ponderación para las propiedades de los cationes base junto a los dopantes:

$$r_A = (r_{La} \cdot (1 - x)) + (r_M \cdot x) \tag{3.2}$$

$$r_B = (r_B \cdot (1 - y)) + (r_N \cdot y) \tag{3.3}$$

$$n_A = (n_{La} \cdot (1 - x)) + (n_M \cdot x) \tag{3.4}$$

Donde:

- x: porcentaje de composición del catión dopante de sitio A
- y: porcentaje de composición del catión dopante de sitio B
- r_E: radio iónico de elemento E
- n_E: estado de oxidación de elemento E

Las ecuaciones están basadas en la expresión de la ecuación 3.1 de las perovskitas a estudiar ($La_{(1-x)}M_xB_{(1-y)}N_yO_3$). Además, al reemplazar cualquiera de las variables x, y, δ por 0.5 (equivalente a 50 %), se vuelve a la expresión original de media aritmética.

3.4. Algoritmo base

Para la confección de la base de datos se utilizarán el esqumea de funciones de "*predict-perovskites*" [12] los cuales fueron elaborados y utilizados en la investigación donde se obtuvo el nuevo factor de tolerancia NFT (τ).

Los archivos del esquema de funciones de "*predict-perovskites*" realizan varias acciones que permiten finalmente obtener la probabilidad de formación de un conjunto de compuestos de perovskitas, el cual también es generado por el algoritmo. Las principales tareas que el algoritmo desempeña son:

- Extraer y ordenar información de radios iónicos Shannon radii.
- Generar de compuestos candidatos de perovskitas, como objetos python de las clases `PredictABX3` y `PredictAABBXX6`.
- Analizar y cálculo de factores de tolerancia de perovskitas.
- Generar planillas de resultados con detalles de valores obtenidos y propiedades utilizadas.

A continuación se presenta un diagrama de flujo del funcionamiento del repositorio. Más adelante, en la tabla 3.1 se especifican todos los archivos del esquema original de [12].

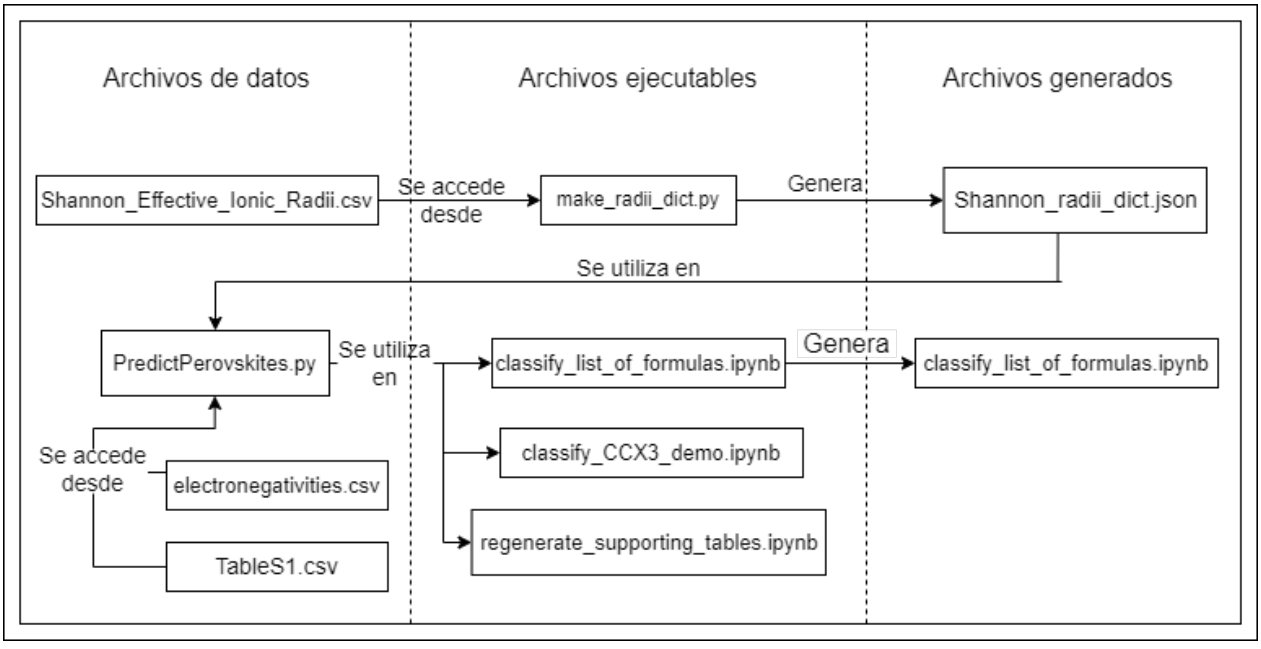


Figura 3.1: Diagrama de flujo del funcionamiento del algoritmo base a utilizar. [12]

Tabla 3.1: Archivos de repositorio *predict-perovskites* [12].

Nombre función	Descripción
TableS1.csv	576 compuestos ABX 3 utilizados for entrenar y probar el valor t
TableS2.csv	Matrices de confusión para t (superior) y t (inferior)
TableS3.csv	Información adicional relaciondada con la figura 2D de [8].
TableS4.csv	Double perovskite oxides and halides
electronegativities.csv	Planilla de datos con valores de electronegatividades de elementos
Shannon_Effective_Ionic_Radii.csv	Planilla de datos con información de radios iónicos de elementos con varios estados de oxidación
make_radii_dict.py	Script de python para transformar la planilla de $\text{Shannon}_{E}ffective_{I}onic_{R}adiiadiccionario$ de python
PredictPerovskites.py	Archivo python con las clases python para perovskitas dobles y simples
classify_CCX3_demo.ipynb	Archivo jupyter con ejemplo paso a paso desde generación de compuestos hasta obtención de probabilidades de formación
classify_list_of_formulas.ipynb	Archivo jupyter con ejemplo de generación y análisis de una lista de compuestos de perovskitas
regenerate_supporting_tables.ipynb	Archivo jupyter para generar archivos de validación del nuevo factor de tolerancia
classified_formulas.csv	Planilla de datos, con los resultados de archivo classify_list_of_formulas
icsd_A2BBX6.csv	Planilla con resultados de análisis de estabilidad para perovskitas tipo A2BBX6
LICENSE	Licencia MIT del código y los demás programas.
README.md	Resumen general de los archivos del repositorio
Shannon_radii_dict.json	Información de Shannon radii en formato json

Pese a que el esquema base se adapta de gran manera al trabajo a realizar, se deben incorporar los requerimientos específicos del proyecto, tales como: generar compuestos que presenten dopaje y la posibilidad de variación composicional sobre las perovskitas dopadas, por lo cual es necesario trabajar en diferentes modificaciones para alcanzar los objetivos propuestos.

3.5. Software a utilizar

Para lograr trabajar con los diferentes archivos y desarrollar el trabajo, se definió la siguiente tabla con los softwares requeridos:

Tabla 3.2: Software a utilizar para el trabajo

Softwares a utilizar	
python v3.8	Lenguaje de programación ampliamente utilizado. Los códigos y módulos a utilizar están basados en este lenguaje. Formato .py.
jupyter	Herramienta que combina la edición de visualizaciones, texto, expresiones matemáticas, y ejecución de código python en un mismo archivo. Se utiliza para los archivos de formato .ipynb
pyCharm	Editor de código (IDE) para el trabajo con códigos de diferente formato. Se utiliza entonces para la edición de los archivos de formato .py. Se utilizó la licencia de estudiante que entrega la universidad para la instalación del programa.
LibreOffice	Editor de plantillas de datos (similar a excel) de libre acceso, se utiliza para la edición de los archivos de formato .csv.

3.6. Esquema de trabajo

Para la realización del trabajo se siguió la secuencia de pasos que se presenta en la figura 3.2, posteriormente se tiene la descripción de cada una de las etapas. Se dispone de 5 etapas en total, las cuales se siguen de manera secuencial hasta la obtención de los resultados finales.

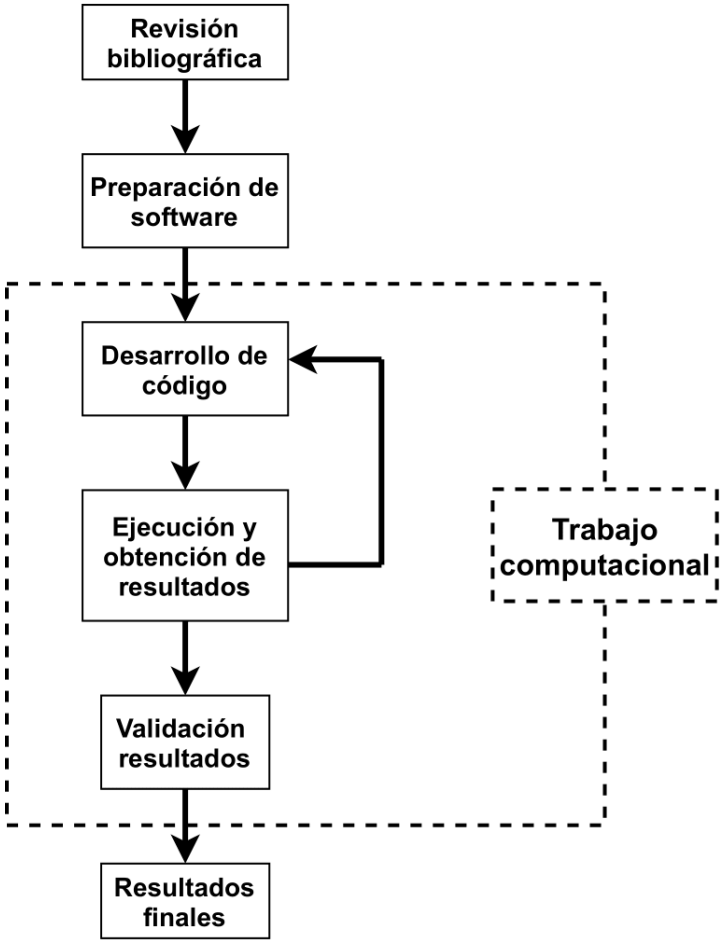


Figura 3.2: Metodología de pasos a seguir para la realización del trabajo.

3.6.1. Revisión bibliográfica

La revisión bibliográfica consiste en el proceso de recopilación de información relevante para el trabajo a realizar, lo cual permite sentar las bases del trabajo.

Se deberá finalizar la etapa de revisión con la consolidación de un conjunto de archivos y referencias que contengan todo lo necesario para la realización de la investigación. Esta etapa será la única enfocada en la búsqueda de nueva información, para así posteriormente dedicarse exclusivamente al trabajo práctico.

3.6.2. Preparación de software

Una vez definidos los procedimientos, y con la información necesaria para el trabajo, se puede dar paso al trabajo práctico. Se deben completar las descargas e instalaciones de todos los archivos y software necesarios. En resumen, es dejar toda la materia computacional preparada.

3.6.3. Trabajo computacional

Esta etapa consiste en la implementación en el código con el cual se obtendrán los resultados finales.

Como indica el diagrama de la figura 3.2, se trabajará de manera iterativa, obteniendo así diferentes niveles de resultados, cada uno con la incorporación de uno de los requerimientos necesarios. Finalmente tendrá el código final que permitirá generar la base de datos esperada.

Esta etapa se considera como un subproceso del desarrollo de código, donde para cada nueva incorporación sobre el código, se obtienen resultados de avance. Este proceso se realiza de manera iterativa, razón por la cual en el diagrama de la figura 3.2 se tiene una flecha segmentada desde *Obtención de resultados* hacia *Desarrollo de código*.

Se esperan entonces diferentes tipos de resultados, según el nivel de avance que se tenga en el código. Además, se irán completando diferentes metas de avance, hasta llegar así a los resultados finales de la siguiente etapa. Los procesos a completar serán:

- Compuestos de estudio
- Formato de archivos
- Variacion composicional
- Cálculo de factores de tolerancia
- Archivo de porcentajes de formación

3.6.4. Validación

Se realizará un procedimiento de validación de resultados, que incluye la comparación de la versión final de trabajo, con la versión original del algoritmo *predict-perovskites*. También se hará una evaluación de un set de muestras de perovskitas, en el código desarrollado. Estas muestras serán tomadas a partir de diferentes publicaciones que contengan información de perovskitas ya estudiadas en laboratorio.

3.6.5. Resultados finales

Esta etapa consiste en la finalización del trabajo, donde se da lugar a la versión de código final, subiendo, consolidando todos los archivos utilizados y terminando la redacción de la tesis, la cual se llevó de manera paralela a lo largo del trabajo.

Capítulo 4

Resultados y discusión

Los resultados obtenidos inicia con la presentación del estudio y análisis en detalle del esquema original de funciones del esquema de funciones base de [12], luego se continúa con la incorporación de los distintos requerimientos del trabajo al esquema de funciones. Por último se cuenta con el proceso de validación de los datos, seguido de un resumen de los resultados finales del trabajo.

Los puntos descritos se dividen en las secciones de *Análisis en detalle del esquema original*, *Desarrollo de código*, *Validación de resultados* y por último *Resultados finales: Base de datos generada.*

4.1. Análisis en detalle del esquema original

A diferencia del análisis ya presentado en la figura 3.1, en esta sección se presenta un análisis detallado de las funciones que se incluyen en la clase de perovskitas del archivo `PredcitPerovskites.py` incluido en el esquema original. Este archivo contiene dos clases python, las cuales reciben de nombres `PredictABX3` y `PredictAABBXX6`, ambas clases contienen todas las funciones necesarias para trabajar con composiciones de perovskitas teóricas y llegar a obtener los resultados de estabilidad deseados.

Dado los requerimientos del trabajo, el estudio se centró sobre la clase asociada a perovskitas que presenten dopaje, es decir la clase `PredictAABBXX6` asociada a perovskitas de doble dopaje. Del análisis se desprende que las funciones que contiene la clase se pueden dividir en tres grupos principales:

- **Funciones básicas:** Entregan las propiedades físicas y químicas de los compuestos a analizar, tales como radio iónico, o valores de electronegatividad de elementos.
- **Funciones de estado:** Este grupo de funciones obtienen el estado de balance de los compuestos a analizar. Cumplen tareas de balance de cargas, obtención de la fórmula

química empírica de los compuestos, estados de oxidación de los elementos, entre otras tareas.

- **Funciones de cálculo de parámetros:** Son las funciones finales del esquema y realizan el cálculo de los factores de tolerancia de Goldschmidt y Nuevo Factor de Tolerancia para los compuestos a estudiar.

Respecto a la clase `PredictAABBXX6`, se pueden describir sus valores inputs y outputs de la siguiente manera, según lo resume el archivo *README* del esquema original.

Código 4.1: Valores inputs y outputs que se pueden obtener con la clase PredictAABBXX6.

```
PredictAABBXX6(object)
(input A1, A2, B1, B2, X1, X2;
output nA, nB, nX, rA, rB, rX, t, tau, t_prediction, tau_prediction, tau_probability)


```

Al observar los resultados que cada una de las funciones arroja, se llega a la conclusión que se debe trabajar con el segundo grupo de funciones mencionado, las denominadas "*Funciones de estado*" dado que son las que definen los parámetros clave de cada compuesto, que consisten en: estado de balance de cargas y fijar los valores composicionales de cada elemento dopante sobre el compuesto.

A continuación, en la figura 4.1 se presenta un esquema resumen con las conexiones de las funciones que permiten definir el estado de balance de cargas de los compuestos a analizar. Como se aprecia en el esquema, el estado final de cargas del compuesto se define en la función `chosen_ox_states()`, la cual entrega como ouptut un diccionario python de la forma `{ 'Elemento' : n_ox }`, detallando así el estado de oxidación que permiten el balance de cargas, de cada uno de los elementos que forman al compuesto de perovskita.

Para llegar al diccionario mencionado, la función utiliza varias otras funciones, de las cuales destacan dos, `conc_dict()` la cual define las concentraciones de cada elemento en el compuesto y la función `choice_dict()` la cual posee las combinaciones de estados de oxidación de los elementos, que permiten el balance de carga para cada compuesto.

A su vez, ambas funciones requieren de otras para lograr entregar sus valores respectivos, llegando así hasta las funciones `good_form()` y `bal_combos()` que fijan valores base. Al analizar las relaciones e impacto de las funciones de ambas ramas, se llega a la conclusión de modificar las funciones `conc_dict()` y `bal_combos()` (encerradas en cuadros azules), ya que sientan los valores finales de concentración de dopaje y combinaciones de estados de oxidación para el balance de cargas. Ambos conjuntos de valores son posteriormente utilizados por la

función de resultados `chosen_ox_states()` (recuadro rojo).

Es importante lograr modificar los resultados que la función `chosen_ox_states()` arroja, ya que los demás resultados, tanto de propiedades, como también de cálculo de parámetros de cada compuesto, se basan en el estado de balance que entregue dicha función. Este análisis permitirá entonces enfocar el trabajo de desarrollo de código en la sección a continuación.

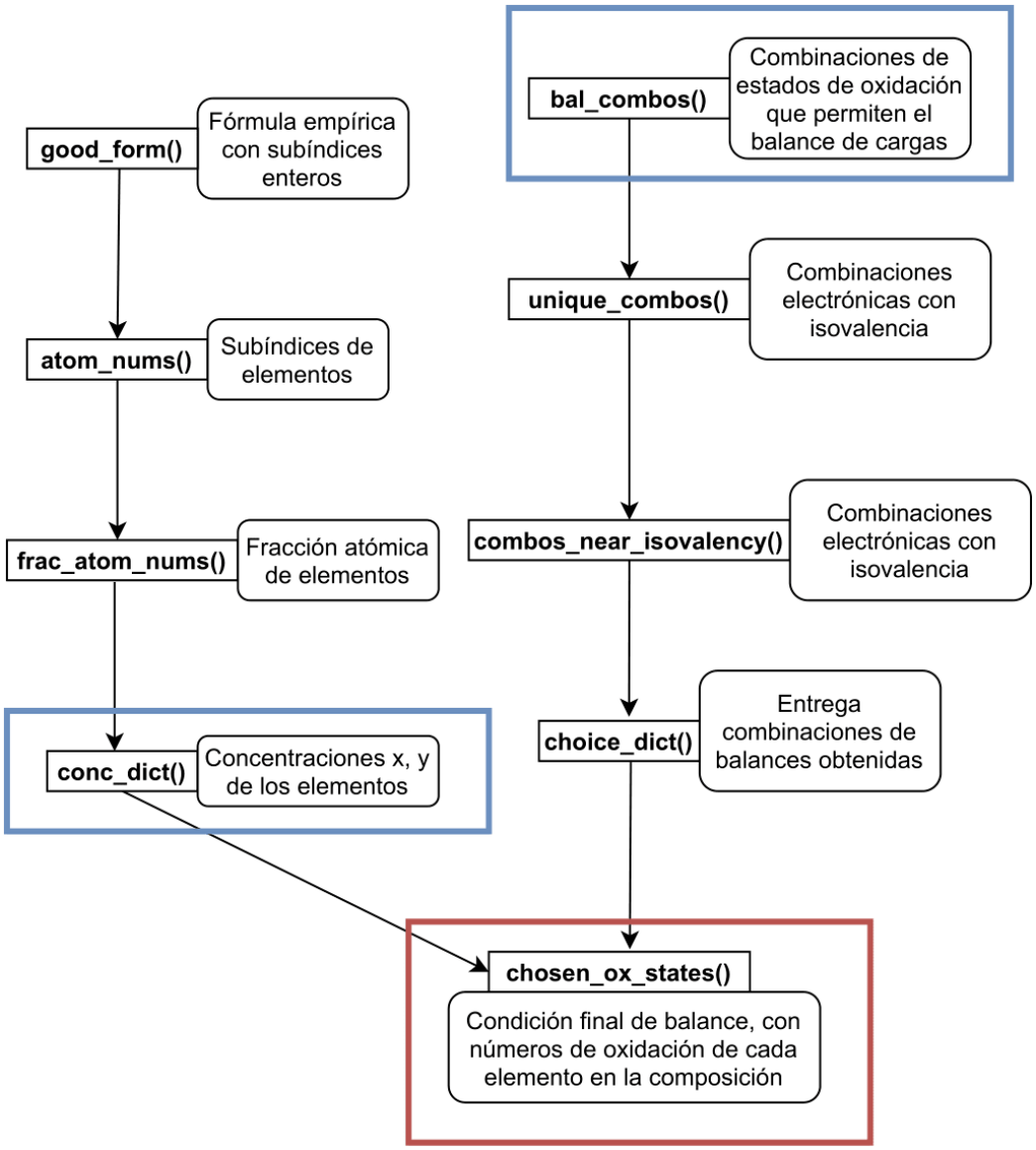


Figura 4.1: Diagrama de flujo de funcionamiento de funciones de la clase `PredictAABBXX6`.

4.2. Desarrollo de código

Tomando en cuenta el análisis que se tiene del esquema original, se da lugar a la incorporación de las modificaciones necesarias mediante el trabajo práctico con código. Los puntos a cumplir se ordenan bajo los títulos de: *Compuestos de estudio*, *Variación composicional*, *Cálculo de factores de tolerancia*, *Formato final* y *Archivos adicionales*.

4.2.1. Compuestos de estudio

En el esquema original, la generación de compuestos de estudio ocurre con una serie de bucles "for", sobre los vectores de elementos a considerar para la generación de compuestos, los cuales se denominan `A_cations`, `B_cations` y `X_anions`.

La iteración almacena cada elemento según la posición (A1, A2, B1, B2) del compuesto a formar en un diccionario denominado `candidates_dict`, luego de recorrer todos los elementos base, se obtiene un diccionario de compuestos candidatos, el cual es posteriormente utilizado para conformar los objetos de perovskita `PredictAABBXX6` a analizar.

Primeramente se restringió la generación de perovskitas para trabajar exclusivamente con perovskitas que presenten dopaje, luego se añadió la restricción de fijar lantano como catión base A1. Esto se logró mediante la incorporación de la siguiente línea de código.

Código 4.2: Restricciones incorporadas para la generación de compuestos candidatos.

```
1     if (A1 != A2) and (B1 != B2) and (A1 == 'La'):
2
```

Respecto a la cantidad de compuestos candidatos generados, se identificó un detalle respecto a los compuestos que se obtenían, ya que dependiendo de los intervalos de valores para la variación composicional, existía el caso de generar compuestos duplicados, al intercambiar B1 por B2, a continuación se explica en detalle ambas alternativas que surgen para la generación de compuestos.

- Usar intervalos $0.1 \leq x, y \leq 0.5$ implica que se deberá generar una "copia" de cada compuesto candidato para lograr cubrir todos los casos composicionales, es decir para el candidato LaSrCoFeO6, también se tendrá su homólogo LaSrFeCoO6. Esto se debe repetir para cada compuesto, generando un gran número de perovskitas las cuales analizar.
- Usar intervalos $0.1 \leq x, y \leq 0.9$ lo cual permite generar un único compuesto candidato para cada combinación, dado que se cubren todos las combinaciones de valores composicionales para el compuesto a estudiar.

Pese a que ambos intervalos implican el mismo número de cálculos a realizar, se decide utilizar la segunda opción, dado que la cantidad de compuestos a trabajar será menor. Para evitar entonces la generación de compuestos duplicados, se utiliza un vector auxiliar `candidateDuplicate`, para así llevar un registro de los compuestos ya formados y evitar el almacenamiento del duplicado. A continuación se tiene la sección de código con la cual se generan los compuestos candidatos.

Código 4.3: Generación de compuestos candidatos.

```
#GENERACION DE COMPUESTOS
for A1 in A_cations:
    for A2 in A_cations:
        for B1 in B_cations:
            for B2 in B_cations:
                if (A1 != A2) and (B1 != B2) and (A1 == 'La'):
                    candidate = ''.join([A1, A2, B1, B2, X1, '6'])
                    candidatesReg.append(candidate)
                    candidateDuplicate = ''.join([A1, A2, B2, B1, X1, '6'])
                    if not candidateDuplicate in candidatesReg:
                        tmp_dict = {}
                        tmp_dict['A1'] = A1
                        tmp_dict['A2'] = A2
                        tmp_dict['B1'] = B1
                        tmp_dict['B2'] = B2
                        tmp_dict['X1'] = X1
                        tmp_dict['X2'] = X1
                        tmp_dict['formula'] = candidate
                        candidates_dict[candidate] = tmp_dict

```

Es importante recalcar que para esta etapa aún no se tiene ningún tipo de información de los compuestos generados, no se conoce si son estables, tampoco la configuración electrónica de los compuestos, ni mucho menos los valores de nuevo factor de tolerancia de éstos. Los compuestos generados funcionan como compuestos candidatos de perovskitas, ya que posteriormente se concluirá si es que efectivamente es probable que formen estructuras de tipo perovskita o no.

4.2.2. Variación composicional

Tomando en cuenta los antecedentes que se tienen, respecto las funciones de la clase de `PredictAABBXX6` se decidió incorporar los valores de variación composicional a los atributos de los objetos `PredictPerovskitesAABBXX6`, para que puedan ser accedidas por las demás funciones en cualquier momento. Los inputs de la clase quedó de la siguiente manera:

Código 4.4: Variables de función inicializadora de objetos PredictAABBXX6.

```
def ___init___(self, A1, A2, B1, B2, X1, X2,x_var=0.5,y_var=0.5,delta_var=0):
    """
    Args:
        A1 (str) - element A
        A2 (str) - element A' if applicable, otherwise A
        B1 (str) - element B
        B2 (str) - element B' if applicable, otherwise B
        X1 (str) - element X
        X2 (str) - element X' if applicable, otherwise X
    """
    self.A1 = A1
    self.A2 = A2
    self.B1 = B1
    self.B2 = B2
    self.X1 = X1
    self.X2 = X2
    self.x = x_var
    self.y = y_var
    self.delta = delta_var

```

Las variables mencionadas reciben el nombre de `x_var`, `y_var` y δ`_var`, que toman los valores por defecto de 0.5, 0.5, y 0. A continuación se presenta la incorporación de estos valores en las demás funciones `conc_dict()`, `bal_combos()` y `chosen_ox_states()` que se presentaron en la figura 4.1.

conc_dict()

Para acceder al porcentaje de dopaje de cada elemento en el compuesto, el esquema original se encarga de llamar a los valores que entrega la función *conc_dict()*, por lo tanto para hacer efectiva la variación de composición sobre el esquema completo es necesario modificar los valores que entrega esta función.

Esta modificación se realiza utilizando los valores recién incorporados, de porcentaje composicional entre los atributos de los objetos `PredictAABBXX6`, los cuales son entregados de manera directa como resultados por parte de la función. A continuación se observa el código de la nueva función `conc_dict()` elaborada.

Código 4.5: Nueva función conc_dict().

```
@property
def conc_dict(self):
```

```
"""
returns dictionary of {el (str) : concentration in AxA'1-xByB'1-yXzX'3-z format (float)}
"""
els = self.atom_names
conc = self.frac_atom_nums
natoms = self.num_atoms
A1 = self.A1
A2 = self.A2
B1 = self.B1
B2 = self.B2
X = self.X[0]
x = self.x
y = self.y
delta = self.delta
tmp_dict = {}
for idx in range(len(els)):
    if els[idx] == A1:
        tmp_dict[els[idx]] = 1-x
    elif els[idx] == A2:
        tmp_dict[els[idx]] = x
    elif els[idx] == B1:
        tmp_dict[els[idx]] = 1-y
    elif els[idx] == B2:
        tmp_dict[els[idx]] = y
    elif els[idx] == X:
        tmp_dict[els[idx]] = 3 - delta

return tmp_dict

```

bal_combos()

Esta función se encarga de obtener las combinaciones de estados de oxidación de los elementos, que permiten un balance de cargas en los compuestos. La ecuación con la que se evalúan el balance de cargas de los compuestos se detalla a continuación:

$$\sum[n_{ox,i} \cdot x] = 0 \tag{4.1}$$

Donde:

- $n_{ox,i}$: estado de oxidación del elemento i
- $x - y - \delta$: fracciones composicionales del compuesto

Al expandirla con los términos de los compuestos a analizar $La_{(1-x)}M_xB_{(1-y)}N_yO_6$ queda de la siguiente manera.:

$$n_{ox,La} \cdot (1-x) + n_{ox,M} \cdot x + n_{ox,B} \cdot (1-y) + n_{ox,N} \cdot y + n_{ox,O} \cdot (3-\delta) = 0 \tag{4.2}$$

El esquema original se encarga de evaluar las diferentes combinaciones de estados de oxidación en la ecuación para obtener las combinaciones que permitan satisfacer la ecuación recién presentada. Además, como ya se detalló anteriormente, el esquema original sólo trabaja con composiciones fijas de 0.5 para los elementos dopantes, lo cual implica que en la sección de código que calcula los estados de balances, los valores de fracción composicional x e y estaban fijos en las composiciones de 0.5 mencionadas y sin posibilidad de ser alterados.

Es así como se utilizaron los atributos de composición $x - y - \delta$ de los objetos `PredictAABBXX6` para incluirlos en la expresión realizada por `bal_combos()`, al momento de realizar el cálculo de balance de cargas. El código incorporado que permite este cambio se detalla a continuación:

Código 4.6: Código incorporado en función bal_combos().

```
bal_combos = []
for combo in isovalent_combos:
    cations_sum = round(np.sum([conc_dict[el] * combo[idx_dict[el][0]] for el in cations]),2)
    anion_sum = round(-2 * conc_dict[self.X[0]],2)
    net_charge = round(cations_sum + anion_sum,2)
    if net_charge == 0:
        bal_combos.append(combo)

```

Consiste en utilizar los valores que entrega la función `conc_dict()` recién modificada, para así multiplicarlos por el estado de oxidación tanto en cationes (`cations_sum`) como aniones (`anion_sum`) y finalmente con la variable `net_charge` obtener si efectivamente conforman una combinación de balance de cargas o no.

Producto de los diferentes estados de oxidación de cada elemento, puede ocurrir que la función de `bal_combos()` entregue más de una combinación de valores que cumplen el balance, estos son entregados a una serie de funciones como se observa en el esquema resumen de la figura 4.1, hasta llegar a `chosen_ox_states()`, sin embargo es en dicha función donde finalmente se decide, cuál de las alternativas utilizar.

chosen_ox_states()

Como ya se adelantó, esta función entrega el grupo de estados de oxidación de los elementos en el compuesto de perovskita, si es que se logró encontrar un estado de balance de cargas. La función recibe valores de otras funciones, sin embargo los más importantes son los de las funciones que ya se trabajaron, las concentraciones de *conc_dict()* y las cargas que permiten el balance `bal_combos`.

Esta funciona como un último filtro respecto a la combinación de estados de balance final del compuesto de perovskita a estudiar, dado que llama los estados de oxidación de otra función, y en el caso de que se haya encontrado más de una combinación que permita el balance de cargas, entonces compara ambas funciones y selecciona la que presenta una menor diferencia de electronegatividades entre los elementos enlazantes.

Sin embargo al momento de realizar algunas ejecuciones de prueba, se encontró que existían algunos casos que se escapaban a la programación de la función, generándose un error en en el programa. Este error se daba con compuestos con más de una combinación de balance de cargas, las cuales fueran idénticos en términos de sumatoria de cargas, como diferencia de electronegatividades. Algunas elementos que generaban el error descrito, incluían Ir, Pd, y Ru.

Para resolver este error, se optó a que en el caso de tener dos combinaciones de balances numéricamente iguales, se tome la primera opción, mediante el código que se describe a continuación:

Código 4.7: Modificaciones sobre funcion chosen_ox_states.

```
if len(ox_dict) != len(els):
for idx in range(len(unspec_els)):
    el = unspec_els[idx]
    ox_dict[el] = combo[idx]
    if len(ox_dict) == len(els):
        return ox_dict

```

Esto fue agregado para las situaciones finales de la función, en el caso de que aun no se tuviesen los estados de oxidación de los elementos.

4.2.3. Cálculo de factores de tolerancia

Una vez completa la incorporación de variación composicional en el esquema, se prosigue con lo relacionado al cálculo de los factores de tolerancia finales de cada compuesto.

Para esto se comienza con la incorporación de las ecuaciones 3.2 - 3.4 a las funciones de radio atómico (rA, rB) y estado de oxidación (nX) del esquema original. Esto se resuelve utilizando de nuevo los atributos composicionales agregados a la clase de perovskita y programando las ecuaciones 3.2 - 3.4 en los resultados de las funciones de la siguiente manera:

Código 4.8: Código incorporado para ponderación de propiedades atómicas.

```
return ((1 - self.x) * self.rA1) +  (self.x * self.rA2)

```

La función tendrá sus propias variables A o B según corresponda. Con estos cambios en las propiedades ponderadas de los elementos, las funciones de cálculo de factores de tolerancia (funciones `.t()` `.tau()`) podrán trabajar normalmente con los valores de r_A, r_B y n_X que requieran. Con esto se pasa a la sección final para explicar el formato de los archivos que compondrán la base de datos.

4.2.4. Formato de archivos

Entre los archivos del código original, se tiene el archivo de demostración `classify_list_of_formulas.i` el cual genera una planilla de resultados de nombre `classified_formulas.csv` con un listado de compuestos analizados. Para cada uno de los compuestos se calculan los valores de los factores de tolerancia de Goldschmidt, nuevo factor de tolerancia NFT y probabilidad de formación P(τ). En la tabla 4.1 se muestra el formato del archivo generado por el esquema original.

Tabla 4.1: Formato original de archivos generados por *classified_formulas.csv*

formula	tag	tau	tau_pred	tau_prob	t	t_pred	A	B	A1	A2
Compuesto 1										
Compuesto 2										
...										
Compuesto n										

B1	B2	X1	X2	nA	nB	nX	rA	rB	rX

De la tabla 4.1 se identifica que algunas columnas no son aplicables para la investigación, en específico las columnas “A”, “B”, “X1”, y “X2”. También se identifica que es necesario incluir otros campos, tales como las variaciones de composición que se aplicaron sobre el compuesto (valores x y δ), además de los nombres y otras propiedades de los elementos dopantes.

Producto que se tendrán varios compuestos a analizar y a su vez cada de ellos contará con diferentes combinaciones de valores composicionales, se decide que cada compuesto tendrá su propio archivo, con un nombre igual al del compuesto analizado. A continuación, en la tabla 4.2 se tiene un ejemplo del nuevo formato de los archivos a generar.

Tabla 4.2: Formato final de archivos para la base de datos.

Formula	x-y-delta	FTG t	Pred. FTG	NFT tau	Pred. NFT	Prob. formacion	A1	nA1
Compuesto 1								
Compuesto 2								
...								
Compuesto n								

rA1	(1-x)	A2	nA2	rA2	x	nA	rA

B1	nB1	rB1	(1-y)	B2	nB2	rB2	y

nB	rB	d	X	nX	rX	(3-d)

Él código con el cual se logra la creación de cada uno de estos archivos se detalla a continuación:

Código 4.9: Código para generación de resultados.

```
1 with open(fileDir, 'w') as f: #CREACION DE CADA ARCHIVO
2     f.write('Formula, x-y-delta,FTG t, Pred. FTG,NFT tau, Pred. NFT, Prob. formacion,
  ↪ A1,nA1,rA1,(1-x),A2,nA2,rA2,x,nA,rA,B1,nB1,rB1,(1-y),B2,nB2,rB2,y,nB,rB,X,nX,rX
  ↪ ,(3-d),d\n')
3
```

Se observa la variable **fileDir** que corresponde al nombre del compuesto que se está analizando el cual será también el nombre del archivo generado. La variable **fileDir** se define de manera automática en el algoritmo final.

4.2.5. Archivo de porcentajes de formación

Consiste en la programación del archivo *01_PorcentajesFormacion.csv*, el cual funciona como una guía para los resultados obtenidos, ya que contiene información respecto los compuestos analizados de la base de datos generada: número de composiciones balanceadas, fórmula analizada, entre otras.

A continuación, se tiene una tabla ejemplo con el detalle de las diferentes columnas de información que se pueden encontrar en el documento.

Tabla 4.3: Información que se encuentra en el archivo de *01_Porcentajes-Formacion.csv*.

Fórmula	A1	A2	B1	B2
Compuesto 1				
Compuesto 2				
...				
Compuesto n				

Nro. composiciones analizadas (a)	Nro. Composiciones balanceadas (b)	Porcentaje de cargas balanceadas (a/b)	Nro. Perov. Probables P(tau) > 0.5 (c)	Porcentaje form. Perov. De compuesto (c/b)

Esto se logra mediante el la incorporación del código 4.10 a continuación, el cual trabaja con variables que se van actualizando en paralelo con el análisis de los compuestos, donde una vez se termina de analizar todos los compuestos, estas variables ya cuentan con información de todos los resultados, la cual es incorporada al archivo de 01_PorcentajesFormacion.csv.

Código 4.10: Código para generación de 01_PorcentajesFormacion.csv.

```
        with open(DirResultados + formationName, 'w') as f:
            f.write('Formula, A1, A2, B1, B2, Nro. composiciones analizadas (a), Nro.
↪ composiciones balanceados (b), Porcentaje de cargas balanceadas (a/b),\
                Nro. Perov. Probables P(tau)>0.5 ( c), Porcentaje form. perov. de
↪ compuesto (c/b)\n')
                for i in range(len(formulas)):
                    formula = formulas[i]
                    A1 = A1s[i]
                    A2 = A2s[i]
                    B1 = B1s[i]
                    B2 = B2s[i]
                    n_comp = NBalanceados[i]
                    total = NAnalizados[i]
                    percentage = PorcentajeBalanceados[i]
                    n_tau = NPerovskitas[i]
                    tau_per = PorcentajePerovskitas[i]

                    seq = [formula, A1, A2, B1, B2, total, n_comp, percentage, n_tau,
↪ tau_per]
                    seq = [str(val) for val in seq]
                    seq = [val if val != 'nan' else '' for val in seq]
                    f.write(','.join(seq) + '\n')

```

4.2.6. Conclusiones finales de etapa de desarrollo de código

Finalmente, se cuenta con todas las funciones y modificaciones necesarias para obtener los resultados deseados. Por lo tanto, se realiza una script final, denominado `Resultados.py` el cual se encuentra detallado en *Anexo A.1.3 Resultados.py* y el cual al ejecutarse, da lugar a la generación de la base de datos.

4.2.6.1. Multiprocesamiento

Al evaluar la ejecución final del código, se observó que el análisis que se realiza sobre cada compuesto requiere muchas operaciones, lo cual se traduce a elevados tiempos de cómputo. Es por esto que se buscó una alternativa para lograr realizar la generación de los archivos de

manera más rápida, llegando así a la alternativa de multiprocesamiento de datos.

El código de multiprocesamiento permite utilizar múltiples núcleos del procesador del computador de manera simultánea. Mediante la implementación del multiprocesamiento se logró aumentar la velocidad de procesamiento de datos al doble, pasando de aproximadamente 50 $\frac{compuestos}{hora}$ a 100 $\frac{compuestos}{hora}$, en un procesador intel core i7-6700T, utilizando cuatro núcleos.

El código de multiprocesamiento contiene el mismo código del archivo `Resultados.py` ya presentado, sin embargo la estructura del algoritmo es diferente, ya que en vez de tener un único proceso lineal, está divido en secciones. También por el hecho de trabajar en paralelo, se debieron adaptar algunas variables auxiliares para que puedan ser modificadas de manera simultánea por los múltiples procesos de manera segura.

En la primera sección se incluyen todas las variables comunes tales como compuestos a analizar, así como también la creación de variables auxiliares que vayan a ser utilizadas en común por todos los compuestos. En segundo lugar se tiene el trabajo particular con cada compuesto, donde finalmente se generarán los archivos. Es la segunda parte la que luego es llamada por el esquema de asignación de procesos para cada uno de los núcleos del procesador.

El código elaborado recibe el nombre de `ResultadosMP.py` y se incluye entre los archivos finales, además de estar detallado en `A.1.3. ResultadosMP.py`.

4.3. Validación de resultados

Para la validación de los datos se utilizaron dos procedimientos, en primer lugar se realizó una comparación entre los resultados obtenidos con el esquema de funciones original y el esquema de funciones desarrollado.

En segundo lugar, se analizó un set de muestras de composiciones de perovskitas, las cuales han sido estudiadas en distintas publicaciones. Esto para verificar que el algoritmo desarrollado las reconozca como composiciones de perovskitas estables. Ambos procesos se muestran a continuación.

4.3.1. Comparación de resultados

Este análisis se realizó sobre los compuestos que conforman la base de datos final, incluyendo **exclusivamente composiciones de dopaje fijas de 0.5,** para los cationes de sitio A y B. Esto tomando en consideración el hecho de que dicho valor composicional es el único que el esquema original permite.

El código elaborado para la comparación se denomina `Comparación.py` y se basa en el archivo ya realizado de `Resultados.py`, pero en lugar de generar un archivo para cada compuesto, se tiene la generación de un único archivo, titulado `02_Comparacion.csv`..

Formula	A1	A2	B1	B2	nA1_O	nA2_O	nB1_O	nB2_O	nX_O	nA1_N	nA2_N	nB1_N	nB2_N	nX_N	tau N	tau O
LaAgAlBiO	La	Ag	Al	Bi	3	1	3	5	['O']	3	1	3	5	['O']	3.886	3.886
LaAgAlCoO	La	Ag	Al	Co					['O']					['O']		
LaAgAlCrO	La	Ag	Al	Cr	3	1	3	5	['O']	3	1	3	5	['O']	4.176	4.176
LaAgAlCuO	La	Ag	Al	Cu					['O']					['O']		
LaAgAlFeO	La	Ag	Al	Fe					['O']					['O']		
LaAgAlGaO	La	Ag	Al	Ga					['O']					['O']		
LaAgAlGeO	La	Ag	Al	Ge					['O']					['O']		
LaAgAlInO	La	Ag	Al	In					['O']					['O']		
LaAgAlIrO	La	Ag	Al	Ir	3	1	3	5	['O']	3	1	3	5	['O']	4.020	4.020
LaAgAlLiO	La	Ag	Al	Li					['O']					['O']		
LaAgAlMgO	La	Ag	Al	Mg					['O']					['O']		
LaAgAlMnO	La	Ag	Al	Mn	3	1	3	5	['O']	3	1	3	5	['O']	4.708	4.708
LaAgAlMoO	La	Ag	Al	Mo	3	1	3	5	['O']	3	1	3	5	['O']	3.966	3.966
LaAgAlNbO	La	Ag	Al	Nb	3	1	3	5	['O']	3	1	3	5	['O']	3.934	3.934
LaAgAlNiO	La	Ag	Al	Ni					['O']					['O']		
LaAgAlNpO	La	Ag	Al	Np	3	1	3	5	['O']	3	1	3	5	['O']	3.886	3.886
LaAgAlOsO	La	Ag	Al	Os	3	1	3	5	['O']	3	1	3	5	['O']	4.013	4.013
LaAgAlPdO	La	Ag	Al	Pd					['O']					['O']		
LaAgAlPtO	La	Ag	Al	Pt	3	1	3	5	['O']	3	1	3	5	['O']	4.020	4.020

Figura 4.2: Captura de archivo 02_Comparacion.csv generado.

Como se observa en la figura 4.2, el archivo de comparación generado posee en las filas cada uno de los compuestos candidatos de perovskitas, mientras que en las columnas se cuenta

con la información de: elementos del compuesto, estados de oxidación del balance obtenidos tanto por el esquema original como el nuevo y finalmente, los valores de NFT (τ) obtenidos también por el esquema original y el nuevo.

Además, en el archivo se incluye el análisis de los 15470 compuestos que componen la base de datos final, donde se logró la misma cantidad de compuestos balanceados por ambos esquemas, es decir, se obtuvieron los mismos resultados independiente del esquema.

Lo anterior implica que los cambios realizados sobre el nuevo esquema realizado logra exitosamente replicar los resultados que se puedan obtener con el esquema original. Tomando en cuenta el indicadores de eficacia de 92 % del esquema original de [12], el cual se obtuvo al comparar al rededor de 900 y casi 600 muestras de perovskitas reales tipo $A_2BB'X_6$ y ABX_3 respectivamente, se puede establecer que los resultados que se obtienen con el nuevo esquema desarrollado están respaldados gracias a la similitud de los resultados con el método original, recién observada.

4.3.2. Análisis de muestras

Las muestras a analizar en el nuevo esquema desarrollado se listan junto a su referencia a continuación:

1. $La_{0.6}Sr_{0.4}Co_{0.1}Fe_{0.9}O_{3-\delta}$, $La_{0.6}Sr_{0.4}Ni_{0.1}Fe_{0.9}O_{3-\delta}$ [9]
2. $La_{0.6}Sr_{0.4}Co_{0.2}Fe_{0.8}O_{3-\delta}$ [10]
3. $LaAlO_3$, $LaGaO_3$, $LaGaO_3$, $LaFeO_3$ [13]
4. $La_{1-x}Ca_xMnO_3$ $(x = 3/8, 1/2, 2/3)$ [14]
5. $LaREO_3$ $(RE = Dy,\ Ho,\ Er,\ Tm,\ Yb,\ Lu)$ [15]
6. $La_{(1-x)}Ca_xCoO_3$ $(x = 0.2, 0.3)$ [16]

Donde en los casos que se tuviera el valor δ, se utilizaron los valores $\delta = 0, 0.1, 0.2, 0.3$ $0.4, 0.5$.

En total se cuenta con quince muestras base, con la presencia de dopaje y variaciones composicionales en algunas de ellas, como ocurre con las muestras de 1, 2, 4 y 6. Tomando en cuenta las variaciones composicionales, se tiene un total de 33 posibles composiciones de perovskitas.

Las muestras se dividieron en dos grupos: perovskitas con y sin dopaje. A continuación se tienen los resultados de probabilidad de formación de perovskita obtenidos para cada uno de

los grupos de compuestos.

Perovskitas simples

Tabla 4.4: Nuevo Factor de Tolerancia y Probabilidad de formación para muestras de perovskitas simples.

Formula	NFT (τ)	Prob. formacion P(τ)
$LaCoO_3$	1.6372	0.989
$LaAlO_3$	1.7909	0.9862
$LaGaO_3$	1.6355	0.989
$LaFeO_3$	1.65	0.9887
$LaDyO_3$	3.7305	0.7327
$LaHoO_3$	3.5519	0.794
$LaErO_3$	3.3845	0.8404
$LaTmO_3$	3.2414	0.8724
$LaYbO_3$	3.0805	0.9009
$LaLuO_3$	2.9918	0.9139

A partir de la tabla 4.4 se observa que se logró el balance de cargas de las diez muestras de perovskitas simples analizadas.

Perovskitas con dopaje

Por otro lado, respecto a las perovskitas con dopaje presente de la tabla 4.5 a continuación, se logró el balance de sólo siete de las veintitrés posibles combinaciones. Pese a la baja cantidad de balances logrados para el caso de las muestras con dopaje, se observa que para cada compuesto base, se obtuvo a lo menos un estado de balance entre sus variadas composiciones analizadas.

Además, en todos los casos donde se logró un balance de cargas, se obtuvo una probabilidad de formación de perovskita mayor al 50 % (mínimo valor de 77 %) lo cual implica que todas las muestras base analizadas poseen al menos una composición de perovskitas probable. Lo anterior se complementa a los resultados esperados del proceso, ya que las muestras analizadas efectivamente constituyen composiciones existentes de perovskitas, sin embargo en muchas se tenía el valor de delta δ desconocido.

Otro punto a considerar es que en el caso del $La_{(1-x)}Ca_xCoO_3$ $(x = 0.2, 0.3)$ el programa desarrollado no logró ningún estado de balance de cargas, lo cual se contradice con la información que se tenía, dado que dicha composición de perovskita efectivamente existe, y fue

Tabla 4.5: Resultados obtenidos para muestras de perovskitas.

Formula	NFT tau	Prob. formacion
La0.6Sr0.4Co0.2Fe0.8O3-0	NB	NB
La0.6Sr0.4Co0.2Fe0.8O3-0.1	2.7139	0.9443
La0.6Sr0.4Co0.2Fe0.8O3-0.2	2.7056	0.945
La0.6Sr0.4Co0.2Fe0.8O3-0.3	2.7127	0.9444
La0.6Sr0.4Co0.2Fe0.8O3-0.4	NB	NB
La0.6Sr0.4Co0.2Fe0.8O3-0.5	2.8389	0.9323
-	-	-
La0.6Sr0.4Co0.1Fe0.9O3-0	NB	NB
La0.6Sr0.4Co0.1Fe0.9O3-0.1	NB	NB
La0.6Sr0.4Co0.1Fe0.9O3-0.2	2.705	0.9451
La0.6Sr0.4Co0.1Fe0.9O3-0.3	NB	NB
La0.6Sr0.4Co0.1Fe0.9O3-0.4	NB	NB
La0.6Sr0.4Co0.1Fe0.9O3-0.5	NB	NB
-	-	-
La0.6Sr0.4Ni0.1Fe0.9O3-0	NB	NB
La0.6Sr0.4Ni0.1Fe0.9O3-0.1	NB	NB
La0.6Sr0.4Ni0.1Fe0.9O3-0.2	2.7052	0.9451
La0.6Sr0.4Ni0.1Fe0.9O3-0.3	NB	NB
La0.6Sr0.4Ni0.1Fe0.9O3-0.4	NB	NB
La0.6Sr0.4Ni0.1Fe0.9O3-0.5	NB	NB
-	-	-
La0.4Ca0.6Mn0.6Mn0.4O3-0	NB	NB
La0.5Ca0.5Mn0.5Mn0.5O3-0	3.6082	0.776
La0.7Ca0.3Mn0.3Mn0.7O3-0	NB	NB
-	-	-
La0.2Ca0.8Co0.8Co0.2O3-0	NB	NB
La0.3Ca0.7Co0.7Co0.3O3-0	NB	NB

estudiada en un ensayo de esfuerzo - tracción en [16].

Al analizar este punto, se observó que pese a que el código logra replicar los resultados de perovskitas de doble dopaje (como se observó recién en *Comparación*), en el caso de compuestos con dopaje simple ocurre la generación de algunas variables equivocadas en el algoritmo.

Por otro lado, respecto al compuesto $La_{1-x}Ca_xMnO_3$ $(x = 3/8, 1/2, 2/3)$ [14] si se obtuvo una condición de balance, por lo cual esto corresponde a un punto a analizar respecto a las perovskitas con dopaje simple, sin embargo no fue resuelto en su totalidad, ya que escapa a los alcances de la investigación.

4.4. Resultados finales: Base de datos generada

Los elementos utilizados para la generación de la base de datos se especifican según la figura 4.3 demarcados como se especifica en la leyenda, para cationes de sitio A y B respectivamente:

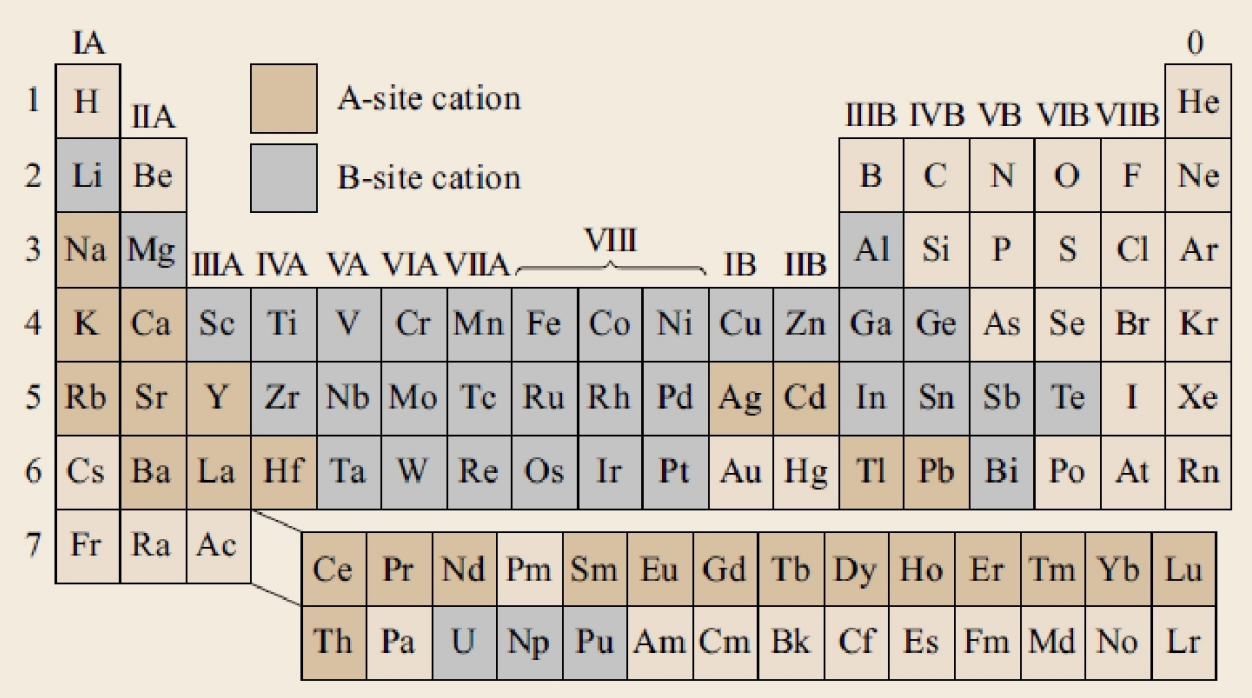


Figura 4.3: Elementos utilizados para generación de base de datos.

Con dichos elementos, se dio lugar a 15 470 compuestos candidatos de perovskitas, además de un archivo índice y dos más de la verificación de resultados.

Los intervalos de variación para las composiciones de los elementos sobre las perovskitas se especifican en la siguiente lista:

- $0.1 \leq x \leq 0.9$
- $0.1 \leq y \leq 0.9$
- $0 \leq \delta \leq 0.5$

4.4.1. Formato y contenidos de archivos

Cada uno de los archivos generados contiene la información que se presentó en la tabla 4.2, cuyos datos se resumen a continuación:

Tabla 4.6: Información entregada por los resultados obtenidos.

Fórmula química	Nombres elementos	Estados oxidación elementos	Radio de elementos	Variaciones composicionales

Factor Goldschmidt	Predicción Goldschmidt	New Tolerance Factor	Predicción New Tolerance Factor	Probabilidad formación New Tolerance Factor

A continuación se tiene un extracto del archivo generado para el compuesto de $La_{1-x}Sr_xCo_{1-y}Fe_yO_{3-\delta}$ (con $0.1 \leq x, y \leq 0.9 \quad \wedge \quad 0 \leq \delta \leq 0.5$) donde se obtuvieron 177 estados de balance, ordenados uno en cada fila:

Tabla 4.7: Extracto de archivo `LaSrCoFeO.csv`.

Formula	x-y-delta	FTG t	Pred. FTG	NFT tau	Pred. NFT	Prob. formacion
La0.9Sr0.1Co0.9Fe0.1O3-0.0	0.1-0.1-0.0	0.975	1	1.939	1	0.983
La0.9Sr0.1Co0.1Fe0.9O3-0.0	0.1-0.9-0.0	0.963	1	1.935	1	0.983
(+174 composiciones)	...	...	...	...	...	...
La0.1Sr0.9Co0.1Fe0.9O3-0.5	0.9-0.9-0.5	0.974	1	3.597	1	0.78

A1	nA1	rA1	(1-x)	A2	nA2	rA2
La	3	1.36	0.9	Sr	2	1.44
La	3	1.36	0.9	Sr	2	1.44
...	...	...	...	...	...	...
La	3	1.36	0.1	Sr	2	1.44

x	nA	rA	B1	nB1	rB1	(1-y)
0.1	2.9	1.368	Co	3	0.61	0.9
0.1	2.9	1.368	Co	4	0.53	0.1
...	...	...	...	...	...	...
0.9	2.1	1.432	Co	2	0.745	0.1

B2	nB2	rB2	y	nB	rB	X
Fe	4	0.585	0.1	3.1	0.6075	O
Fe	3	0.645	0.9	3.1	0.6335	O
...	...	...	...	...	...	...
Fe	3	0.645	0.9	2.9	0.655	O

nX	rX	(3-d)	d
-2	1.4	3	0
-2	1.4	3	0
...	...	...	...
-2	1.4	2.5	0.5

La carpeta con los archivos resultantes contiene la información que se muestra en la tabla

de recién, para cada uno de los compuestos candidatos. Naturalmente algunos compuestos tendrán mayor cantidad de composiciones donde se logró un balance de cargas que otros, así como también se tendrán los casos con compuestos de perovskita donde no se logre ningún estado de balance y por lo tanto estarán en blanco. A continuación se tiene una captura de la carpeta de resultados.

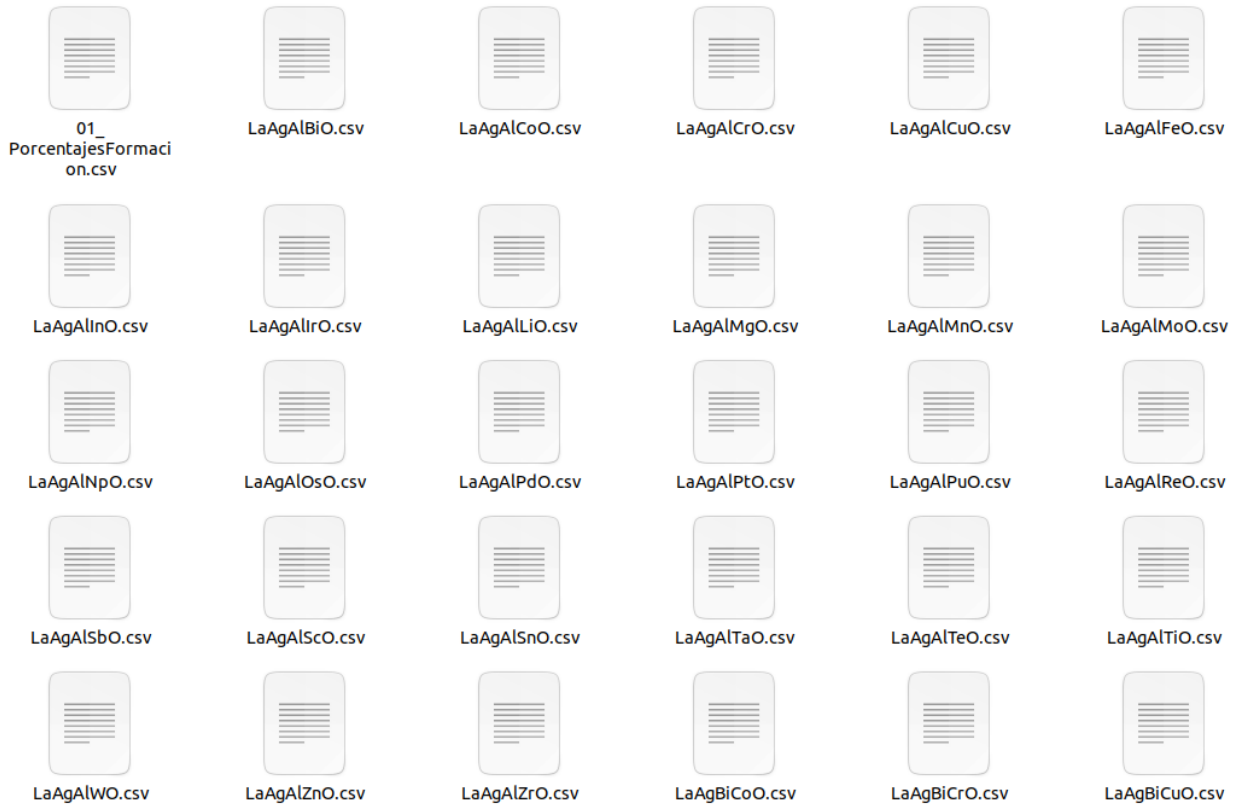


Figura 4.4: Carpeta con archivos de la base de datos generada.

4.4.2. Enlace de repositorio

Para acceder a los datos generados, se deben descargar y extraer los archivos del archivo Estabilidad_Perovskitas.zip que se encuentra en el enlace a continuación:

- https://drive.google.com/drive/folders/16Cz7SwcE4B83-j9krVMnFmEBDgFZ7D1J?usp=sharing [19]

4.4.3. Instrucciones de uso de base de datos

Para la utilización de la base de datos se tienen los siguientes pasos a seguir:

1. Descargar el archivo Estabilidad_Perovskitas.zip desde el enlace de [19] y extraerlo.
2. Localizar la carpeta "Resultados" para luego acceder al archivo "`01_PorcentajesFormacion.csv`", el cual posee un resumen de la base de datos de la carpeta "Perovskitas".
3. Una vez abierto el archivo "`01_PorcentajesFormacion.csv`" activar los filtros de columnas y seleccionar entonces desde las columnas A1, A2, B1 y B2, los elementos que se quieran ver.
4. Al indicar los elementos, se puede observar entonces si se logró algún balance de cargas en el compuesto, revisando que la columna de *Nro. composiciones balanceados (b)* sea distinta a cero.
5. Si se logró realizar un balance de cargas en el compuesto de perovskita a consultar, basta con tomar el nombre del compuesto, desde la primera columna y buscar dicho compuesto en los archivos de la carpeta donde se tengan los resultados.

4.4.4. Indicaciones de archivo Resultados.py

El algoritmo base de `Resultados.py` es el que se utilizó para la obtención de la base de datos. Además funciona como base para el resto de los archivos elaborados de comparación, análisis de muestras y multiprocesamiento. El algoritmo de `Resultados.py` se puede ordenar en tres partes:

- *Inputs*: Corresponde a la primera sección del código, donde se deben ingresar los siguientes datos: "Elementos a considerar para la generación de perovskitas", "Intervalos de porcentajes composicionales" y "Directorios de donde se almacenará la base de datos a generar". Esta sección se observa en el código A.4, de la sección de Anexos.

- *Variables*: Va a continuación de los inputs, y es donde se tienen las variables finales a utilizar para la generación de datos. Esta sección se observa en el código A.5 de Anexos.

- *Generación de datos*: Consta de tres partes principales, en primer lugar se generan los compuestos de perovskitas a evaluar, seguido de la creación de los archivos resultados para luego, finalizar con la creación del archivo guía de los datos generados. Esta sección se observa en el código A.7

4.4.4.1. Consideraciones

- **Formato:** Los archivos se tienen en formato csv, lo que implica que al intentar editar los archivos directamente en google sheets se generan problemas de formato, modificándose así los valores de los archivos. Es por esto que se recomienda que, si se desea trabajar directamente con google sheets, realizar una transformación previa de .csv a cualquier otro formato de planilla de datos, como .xlsx.

Capítulo 5

Conclusión

Se cumplió con el objetivo propuesto, logrando la generación de una base de datos que contiene la información de probabilidad de formación para la familia de perovskitas basadas en lantano con doble dopaje y variación composicional. La base de datos entrega la información para 15 470 perovskitas, utilizando la ecuación del nuevo factor de tolerancia recientemente desarrollada. Para acceder a los resultados se puede visitar el enlace de la referencia de [19].

El trabajo se basa en el estudio "*New tolerance factor to predict the stability of perovskite oxides and halides*" [8], donde se obtuvo la ecuación del Nuevo Factor de Tolerancia mediante el método SISSO de aprendizaje automático. Posterior al descubrimiento de la ecuación, se desarrolló una serie de funciones, para estudiar la efectividad de la ecuación, el conjunto de archivos se titula "*predict-perovskites*" [12] y sirvió como base para el trabajo desarrollado.

El trabajo consistió en ajustar los archivos de [12] a los nuevos requerimientos dados por las características de las perovskitas a estudiar, donde se inició con un estudio detallado del esquema original para así posteriormente trabajar en las modificaciones necesarias para lograr la obtención de los resultados esperados.

Para incluir las variaciones composicionales, se optó por incluir estos valores como atributos de los objetos de la clase `PredictAABBXX6` y posteriormente utilizar estos valores en las diferentes funciones de análisis de cada compuesto a estudiar. Para esto fue necesario consultar a diferentes fuentes para la utilización de las ecuaciones pertinentes al balance químico de compuestos.

Respecto a los resultados finales, se alcanzó un 100 % de similitud respecto a los resultados que se obtienen con el esquema de funciones original, en un espacio de muestras común, lo cual indica que pese a realizar algunas modificaciones sobre el código original, los resultados siguen siendo los mismos. Tomando en cuenta el 92 % de eficacia del esquema original, se puede concluir que este valor se mantendría sobre las muestras analizadas, de perovskitas con doble dopaje fijo de 50 %.

Por otro lado, al analizar un conjunto de 33 composiciones base de perovskitas existentes, se obtuvo que todas las perovskitas (diez en total) simples fueron correctamente detectadas como estructuras estables. Respecto al resto de perovskitas con dopaje, en todas se detectó al menos un estado de balance de cargas, a excepción de un compuesto con dopaje simple, sin embargo esto escapa de los alcances del proyecto.

5.1. Proyecciones y posibles trabajos futuros

Respecto a los posibles trabajos a futuro que se pueden desarrollar en torno a lo presentado, se tiene.

- Realizar un estudio sobre el impacto de ciertos elementos y valores de composición sobre la estabilidad de las estructuras perovskitas, utilizando los resultados que se obtuvieron en esta tesis.
- Generar una nueva base de datos para otros tipos de perovskitas, utilizando los archivos desarrollados en esta tesis.
- Atender al caso de análisis de perovskitas con dopaje simple, en caso que sea necesario. Trabajando sobre el código desarrollado en esta tesis.

Bibliografía

Artículos científicos

[1] Shannon, R. D. (1976). Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides. Acta Crystallographica Section A, 32(5), 751–767. https://doi.org/10.1107/s0567739476001551

[2] Pila de combustible. (2008). Accedido el 07 de Mayo del 2021, en: https://es.wikipedia.org/wiki/Pila_de_combustible#/media/Archivo:Fuel_cell_ES.svg

[3] Professional solid oxide fuel cell stack 5000w 48 number of cells. (n.d.). Accedido el 07 de Mayo del 2021, en: http://www.sofc-fuelcell.com/sale-10440157-professional-solid-oxide-fuel-cell-stack-5000w-48-number-of-cells.html

[4] Ali Akbari. ME7250. ADVANCED MATERIALS FOR ENERGY CONVERSION SOLID CELLS. 5-Electrolyte-perovskites.

[5] Johnsson, M., Lemmens, P. (2007). Crystallography and Chemistry of Perovskites. Handbook of Magnetism and Advanced Magnetic Materials. https://doi.org/10.1002/9780470022184.hmm411

[6] Ou, X., Li, Z., Fan, F., Wang, H.; Wu, H. (2014). Long-range magnetic interaction and frustration in double perovskites Sr2NiIrO6 and Sr2ZnIro6. Scientific Reports, 4(1). doi:10.1038/srep07542

[7] Goldschmidt tolerance factor. Wikizero. (n.d.). https://www.wikizero.com/en/Goldschmidt_tolerance_factor.

[8] Bartel, C. J., Sutton, C., Goldsmith, B. R., Ouyang, R., Musgrave, C. B., Ghiringhelli, L. M., Scheffler, M. (2019). New tolerance factor to predict the stability of perovskite oxides and halides. Science Advances, 5(2). https://doi.org/10.1126/sciadv.aav0693

[9] Rojas R. T., (2018) Estudio del comportamiento ferroelastico de perovskitas basadas en lantano. http://repositorio.uchile.cl/bitstream/handle/2250/159292/Estudio-del-comportamiento-ferroel%C3%A1stico-de-perovskitas-basadas-en-lantano.pdf?sequence=1&isAllowed=y

[10] Akbari-Fakhrabadi, A., Sathishkumar, P., Ramam, K., Palma, R.; Mangalaraja, R. V. (2015). Low frequency ultrasound assisted synthesis of La0.6Sr0.4Co0.2Fe0.8O3−δ (LSCF)

perovskite nanostructures. Powder Technology, 276, 200–203. https://doi.org/10.1016/j.powtec.2015.02.043

[11] Li, W., Ionescu, E., Riedel, R.; Gurlo, A. (2013). Can we predict the formability of perovskite oxynitrides from tolerance and octahedral factors. Journal of Materials Chemistry A, 1(39), 12239. https://doi.org/10.1039/c3ta10216e

[12] CJBartel (2017). CJBartel/perovskite-stability. Accedido el 11 de Abril del 2021, en: https://github.com/CJBartel/perovskite-stability

[13] Araki, W., Takeda, K., amp; Arai, Y. (2016). Mechanical behaviour of ferroelastic lanthanum metal oxides LaMO3 (M = co, Al, Ga, Fe). Journal of the European Ceramic Society, 36(16), 4089–4094. https://doi.org/10.1016/j.jeurceramsoc.2016.07.006

[14] He, J., Wang, R., Gui, J., amp; Dong, C. (2002). Orthorhombic to cubic phase transition in la1?xcaxmno3 perovskites. Physica Status Solidi (b), 229(3), 1145–1154. https://doi.org/10.1002/1521-3951(200202)229

[15] Artini, C., Costa, G. A., amp; Masini, R. (2010). Study of the formation temperature of mixed Lareo3 (RE = Dy, Ho, Er, TMm Yb, Lu) and NDGDO3 oxides. Journal of Thermal Analysis and Calorimetry, 103(1), 17–21. https://doi.org/10.1007/s10973-010-0973-8

[16] Faaland, S., Grande, T., Einarsrud, M.-A., Vullum, P. E.; Holmestad, R. (2005). Stress-strain behavior during compression of polycrystalline La1-xCaxCoO3 Ceramics. Journal of the American Ceramic Society, 88(3), 726–730. https://doi.org/10.1111/j.1551-2916.2005.00165.x

[17] Shannon, R. D. (1976). Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides. Acta Crystallographica Section A, 32(5), 751–767. Accedido el 15 de Mayo del 2021. https://doi.org/10.1107/S0567739476001551

[18] Horn D. V. (2018) David's Virtual Lab. Radii.xlsx Accedido el 07 de Octubre del 2021, en: v.web.umkc.edu/vanhornj/Radii.xls

[19] Repositorio final de resultados. Accedido el 16 de Noviembre del 2021 en: https://drive.google.com/drive/folders/16Cz7SwcE4B83-j9krVMnFmEBDgFZ7D1J?usp=sharing

[20] Lufaso, M. W., Woodward, P. M. (2001). Prediction of the crystal structures of perovskites using the software program SPuDS. Acta Crystallographica Section B Structural Science, 57(6), 725-738. https://doi.org/10.1107/s0108768101015282

Anexo A

Códigos computacionales python

A.1. Resultados y discusión

A.1.1. Funciones editadas de PredictPerovskites.py

Código A.1: Modificaciones sobre funcion conc_dict.

```
        @property
    def conc_dict(self):
        """
        returns dictionary of {el (str) : concentration in AxA'1-xByB'1-yXzX'3-z format (float) }
        """
        els = self.atom_names
        conc = self.frac_atom_nums
        natoms = self.num_atoms
        A1 = self.A1
        A2 = self.A2
        B1 = self.B1
        B2 = self.B2
        X = self.X[0]
        x = self.x
        y = self.y
        delta = self.delta
        tmp_dict = {}
        for idx in range(len(els)):
            if els[idx] == A1:
                tmp_dict[els[idx]] = 1-x
            elif els[idx] == A2:
                tmp_dict[els[idx]] = x
            elif els[idx] == B1:
                tmp_dict[els[idx]] = 1-y
            elif els[idx] == B2:
```

```
            tmp_dict[els[idx]] = y
        elif els[idx] == X:
            tmp_dict[els[idx]] = 3 - delta

    return tmp_dict

```

bla combos

Código A.2: Función bal_combos final.

```
    @property
def bal_combos(self):
    """
    returns dictionary of {ox state combo (tup) : {el : [ox state by site (float)]}}
    """
    X_charge = self.X_charge
    allowed_ox = self.allowed_ox
    idx_dict = self.idx_dict
    cations = self.cations
    conc_dict = self.conc_dict
    lists = [allowed_ox[key][site]['oxs'] for key in cations for site in list(allowed_ox[key].
 ↪ keys())]
    combos = list(product(*lists))
    isovalent_combos = []
    suitable_combos = []
    for combo in combos:
        iso_count = 0
        suit_count = 0
        for key in idx_dict:
            curr_oxs = [combo[idx] for idx in idx_dict[key]]
            if np.min(curr_oxs) == np.max(curr_oxs):
                iso_count += 1
            if np.min(curr_oxs) >= np.max(curr_oxs) - 1:
                suit_count += 1
        if iso_count == len(cations):
            isovalent_combos.append(combo)
        if suit_count == len(cations):
            suitable_combos.append(combo)
    #bal_combos = [combo for combo in isovalent_combos if np.sum(combo) == -
 ↪ X_charge]
    bal_combos = []
    for combo in isovalent_combos:
        cations_sum = round(np.sum([conc_dict[el] * combo[idx_dict[el][0]] for el in
 ↪ cations]),2)
        anion_sum = round(-2 * conc_dict[self.X[0]],2)
```

```
            net_charge = round(cations_sum + anion_sum,2)
            if net_charge == 0:
                bal_combos.append(combo)
        if len(bal_combos) > 0:
            combo_to_idx_ox = {}
            for combo in bal_combos:
                idx_to_ox = {}
                for key in idx_dict:
                    idx_to_ox[key] = sorted([combo[idx] for idx in idx_dict[key]])
                    if idx_to_ox not in list(combo_to_idx_ox.values()):
                        combo_to_idx_ox[combo] = idx_to_ox
                combo_to_idx_ox[combo] = idx_to_ox
            return combo_to_idx_ox
        else:
            #bal_combos = [combo for combo in suitable_combos if combo not in
    ↪ isovalent_combos
             #           if np.sum([conc_dict[el]*combo[idx_dict[el]] for el in cations]) == -
    ↪ X_charge*conc_dict[self.X[0]]]
            for combo in suitable_combos:
                if combo not in isovalent_combos:
                    cations_sum = round(np.sum([conc_dict[el] * combo[idx_dict[el][0]] for el in
    ↪ cations]),2)
                    anion_sum = round(-2 * conc_dict[self.X[0]],2)
                    if cations_sum == -anion_sum:
                        bal_combos.append(combo)
            combo_to_idx_ox = {}
            for combo in bal_combos:
                idx_to_ox = {}
                for key in idx_dict:
                    idx_to_ox[key] = sorted([combo[idx] for idx in idx_dict[key]])
                    if idx_to_ox not in list(combo_to_idx_ox.values()):
                        combo_to_idx_ox[combo] = idx_to_ox
            return combo_to_idx_ox

```

Código A.3: Función chosen_ox_states final.

```
        @property
    def chosen_ox_states(self):
        """
        returns dictionary of {el (str) : chosen ox state (float)}
        """
        cations = self.cations
        conc_dict = self.conc_dict
        els = self.els
```

```
        choices = self.choice_dict
        if isinstance(choices, float):
            return np.nan
        X_ox_dict = self.X_ox_dict
        ox_dict = {}
        ox_dict[self.X1] = X_ox_dict[self.X1]
        if self.X1 != self.X2:
            ox_dict[self.X2] = X_ox_dict[self.X2]
        for cation in cations:
            if len(choices[cation]) == 1:
                ox_dict[cation] = choices[cation][0]
        if len(ox_dict) == len(els):
            return ox_dict
        else:
            unspec_els = [el for el in els if el not in ox_dict]
            cationsAux = cations
            for el in unspec_els:
                cationsAux.remove(el)
            unspec_charge = np.sum([conc_dict[el]*ox_dict[el] for el in cationsAux])
            if len(unspec_els) == 1:
                unspec_combos = list(product(choices[unspec_els[0]]))
            elif len(unspec_els) == 2:
                unspec_combos = list(product(choices[unspec_els[0]], choices[unspec_els[1]]))
            elif len(unspec_els) == 3:
                unspec_combos = list(product(choices[unspec_els[0]], choices[unspec_els[1]],
    ↪ choices[unspec_els[2]]))
            elif len(unspec_els) == 4:
                unspec_combos = list(product(choices[unspec_els[0]], choices[unspec_els[1]],
    ↪ choices[unspec_els[2]], choices[unspec_els[3]]))
            elif len(unspec_els) == 5:
                unspec_combos = list(product(choices[unspec_els[0]], choices[unspec_els[1]],
    ↪ choices[unspec_els[2]], choices[unspec_els[3]], choices[unspec_els[4]]))
            elif len(unspec_els) == 6:
                unspec_combos = list(product(choices[unspec_els[0]], choices[unspec_els[1]],
    ↪ choices[unspec_els[2]], choices[unspec_els[3]], choices[unspec_els[4]], choices[
    ↪ unspec_els[5]]))
            good_combos = []
            for combo in unspec_combos:
                amt = 0
                for idx in range(len(unspec_els)):
                    amt += conc_dict[unspec_els[idx]]*combo[idx]
                cero = amt + unspec_charge - (2*(3-self.delta))
                if cero == 0:
                    good_combos.append(combo)
            if len(good_combos) == 0:
```

```
            return np.nan
        biggest_spread = np.max([np.max(combo) - np.min(combo) for combo in
    ↪ good_combos])
        smallest_spread = np.min([np.max(combo) - np.min(combo) for combo in
    ↪ good_combos])
        spread_combos = [combo for combo in good_combos if np.max(combo) - np.min(
    ↪ combo) == biggest_spread]
        tight_combos = [combo for combo in good_combos if np.max(combo) - np.min(
    ↪ combo) == smallest_spread]
        chi_dict = self.chi_dict
        chis = [chi_dict[el] for el in unspec_els]
        maxdex = chis.index(np.max(chis))
        mindex = chis.index(np.min(chis))
        if np.min(chis) <= 0.9*np.max(chis):
            if len(spread_combos) > 1:
                for combo in spread_combos:
                    if combo[mindex] == np.max(combo):
                        if combo[maxdex] == np.min(combo):
                            for idx in range(len(unspec_els)):
                                el = unspec_els[idx]
                                ox_dict[el] = combo[idx]
                            return ox_dict
                min_ox_most_elec = np.min([combo[maxdex] for combo in spread_combos])
                for combo in spread_combos:
                    if (combo[maxdex] == min_ox_most_elec):
                        for idx in range(len(unspec_els)):
                            el = unspec_els[idx]
                            ox_dict[el] = combo[idx]
                        return ox_dict
            else:
                combo = spread_combos[0]
                for idx in range(len(unspec_els)):
                    el = unspec_els[idx]
                    ox_dict[el] = combo[idx]
        else:
            if len(tight_combos) > 1:
                for combo in tight_combos:
                    if combo[mindex] == np.max(combo):
                        if combo[maxdex] == np.min(combo):
                            for idx in range(len(unspec_els)):
                                el = unspec_els[idx]
                                ox_dict[el] = combo[idx]
                for idx in range(len(unspec_els)):
                    el = unspec_els[idx]
                    ox_dict[el] = combo[idx]
```

```
                    if len(ox_dict) == len(els):
                        return ox_dict
            else:
                combo = tight_combos[0]
                for idx in range(len(unspec_els)):
                    el = unspec_els[idx]
                    ox_dict[el] = combo[idx]
        if len(ox_dict) != len(els):
            for idx in range(len(unspec_els)):
                el = unspec_els[idx]
                ox_dict[el] = combo[idx]
                if len(ox_dict) == len(els):
                    return ox_dict
    return ox_dict

```

A.1.2. Resultados.py

Código A.4: Inputs para generación de resultados.

```
        from PredictPerovskites import *
        import numpy as np
        import time
        import itertools

        #PARAMETROS A DEFINIR
    ↪ ----------------------------------------------------------------------------------------
        ##ELEMENTOS A UTILIZAR PARA LA GENERACION DE ARCHIVOS
        X_anions = ['O']

        A_cations = ['Ag', 'Ba', 'Ca', 'Cd', 'Ce', 'Dy', 'Er', 'Eu', 'Gd', 'Hf', 'Ho', 'K', 'La', '
    ↪ Lu',
                     'Na', 'Nd', 'Pb', 'Pr', 'Rb', 'Sm', 'Sr', 'Tb', 'Th', 'Tl', 'Tm', 'Y', 'Yb']

        B_cations = ['Al', 'Bi', 'Co', 'Cr', 'Cu', 'Fe', 'Ga', 'Ge', 'In', 'Ir', 'Li', 'Mg', 'Mn', '
    ↪ Mo',
                     'Nb', 'Ni', 'Np', 'Os', 'Pd', 'Pt', 'Pu', 'Re', 'Rh', 'Ru', 'Sb', 'Sc', 'Sn', 'Ta',
                     'Te', 'Ti', 'U', 'V', 'W', 'Zn', 'Zr']


        #PARAMETROS VARIACIÓN COMPOSICIONAL
        # VECTOR = [INICIO INTERVALO, FINAL INTERVALO, VARIACIÓN
    ↪ INTERVALO]
        X = [0.1, 0.9, 0.1]
        Y = [0.1, 0.9, 0.1]
        DELTA = [0, 0.5, 0.1]


        #DIRECTORIOS DE ARCHIVOS
        DirResultados = 'Resultados/' #Directorio resultados
        formationName = '01_PorcentajesFormacion.csv' # Nombre archivo
    ↪ PorcentajesFormacion
```

Código A.5: Variables auxiliares del algoritmo.

```

    # VARIABLES AUTOMATICAS
```

```
    ↪ ----------------------------------------------------------------------------------------
##Son variables auxiliares que se utilizan en el resto del programa
## INICIALIZACION DE VARIABLES
X1 = X_anions[0]
candidatesReg = []
formation_percentages = []
candidates = {}
candidates_dict = {}
iteracion = 0
count = 0
clf = PredictABX3('').calibrate_tau
start = time.time()

#Variables para archivo PorcentajeFormacion
formulas = []
A1s = []
A2s = []
B1s = []
B2s = []
NAnalizados = []
NBalanceados = []
NPerovskitas = []
PorcentajeBalanceados = []
PorcentajePerovskitas = []



## CREACION DE VARIABLES
x_vector = np.linspace(X[0], X[1], int(((X[1] - X[0]) / X[2]) + 1))
y_vector = np.linspace(Y[0], Y[1], int(((Y[1] - Y[0]) / Y[2]) + 1))
delta_vector = np.linspace(DELTA[0], DELTA[1], int(((DELTA[1] - DELTA[0]) / DELTA
    ↪ [2]) + 1))
comp_combinations = np.array(np.meshgrid(x_vector, y_vector, delta_vector)).T.
    ↪ reshape(-1, 3)
n_compositions = (len(x_vector) * len(y_vector) * len(delta_vector))
formationDir = DirResultados + formationName
properties = ['t', 't_pred',
              'tau', 'tau_pred',
              'A1', 'nA1', 'rA1',
              'A2', 'nA2', 'rA2',
              'nA', 'rA',
              'B1', 'nB1', 'rB1',
              'B2', 'nB2', 'rB2',
              'nB', 'rB',
              'nX', 'rX']
```

```

```

Código A.6: Código de generación de compuestos candidatos.

```
##GENERACION DE DICCIONARIO DE COMPUESTOS CANDIDATOS
for A1 in A_cations:
    for A2 in A_cations:
        for B1 in B_cations:
            for B2 in B_cations:
                if (A1 != A2) and (B1 != B2) and (A1 == 'La'):
                    candidate = ''.join([A1, A2, B1, B2, X1, '6'])
                    candidatesReg.append(candidate)
                    candidateDuplicate = ''.join([A1, A2, B2, B1, X1, '6'])
                    if not candidateDuplicate in candidatesReg:
                        tmp_dict = {}
                        tmp_dict['A1'] = A1
                        tmp_dict['A2'] = A2
                        tmp_dict['B1'] = B1
                        tmp_dict['B2'] = B2
                        tmp_dict['X1'] = X1
                        tmp_dict['X2'] = X1
                        tmp_dict['formula'] = candidate
                        candidates_dict[candidate] = tmp_dict

```

Código A.7: Código para generación de archivos resultados.

```

#GENERACION DE ARCHIVOS RESULTADOS
for candidate in candidates_dict:
    count += 1

    print('%s/%s' %(count, len(candidates_dict)))

    # Variables base de cada compuesto
    tmp_dict = candidates_dict[candidate]
    A1 = tmp_dict['A1']
    A2 = tmp_dict['A2']
    B1 = tmp_dict['B1']
    B2 = tmp_dict['B2']
    X1 = tmp_dict['X1']
    X2 = tmp_dict['X2']
    compoundName = ''.join([A1, A2, B1, B2, X1])
```

```
17
18      # Variables para PorcentajesFormacion
19      globalIndex = count - 1
20      print(globalIndex)
21      formulas.append( compoundName)
22      A1s.append( A1)
23      A2s.append( A2)
24      B1s.append( B1)
25      B2s.append( B2)
26      NAnalizados_local = len(comp_combinations)
27      NBalanceados_local = 0
28      NPerovskitas_local = 0
29
30
31      fileName = compoundName + '.csv'
32      fileDir = DirResultados + fileName
33
34
35      with open(fileDir, 'w') as f: #CREACION DE CADA ARCHIVO
36          f.write('Formula, x-y-delta,FTG t, Pred. FTG,NFT tau, Pred. NFT, Prob.
    ↪ formacion,A1,nA1,rA1,(1-x),A2,nA2,rA2,x,nA,rA,B1,nB1,rB1,(1-y),B2,nB2,rB2,y,nB,
    ↪ rB,X,nX,rX,(3-d),d\n')
37          for set in comp_combinations: #Iteracion sobre valores composicionales
38
39              #Valores composicionales
40              x = round(set[0], 1)
41              y = round(set[1], 1)
42              delta = round(set[2], 1)
43              xydelta = str(x) + '-' + str(y) + '-' + str(delta)
44
45              tmp_dict = candidates_dict[candidate]
46              tmp_dict['formula'] = ('%s %s %s %s %s %s %sO3-%s' % \
47                              (A1, round((1 - x), 1), A2, round(x, 1), \
48                               B1, round((1 - y), 1), B2, round(y, 1), \
49                               round(delta, 1)))
50              tmp_dict['xydelta'] = xydelta
51
52              classifier = PredictAABBXX6(A1, A2, B1, B2, X1, X2,x,y,delta) #creación de
    ↪ objeto PredictAABBXX6
53
54              for prop in properties:
55                  value = getattr(classifier, prop)
56                  if prop == 't' or prop == 'tau':
57                      value = round(getattr(classifier, prop),3)
58                  tmp_dict[prop] = value
```

```
            tmp_dict['tau_prob'] = round(classifier.tau_prob(clf),3)

            if not math.isnan(tmp_dict['tau']):
                NBalanceados_local += 1
                if tmp_dict['tau_pred'] == 1:
                    NPerovskitas_local += 1
                d = tmp_dict
                seq = [d['formula'], d['xydelta'],
                       d['t'], d['t_pred'],
                       d['tau'], d['tau_pred'], d['tau_prob'],
                       d['A1'], d['nA1'], d['rA1'], 1 - x,
                       d['A2'], d['nA2'], d['rA2'], x,
                       d['nA'], d['rA'],
                       d['B1'], d['nB1'], d['rB1'], 1 - y,
                       d['B2'], d['nB2'], d['rB2'], y,
                       d['nB'], d['rB'],
                       d['X1'][0], d['nX'], d['rX'], 3 - delta, delta]
                seq = [str(val) for val in seq]
                seq = [val if val != 'nan' else '' for val in seq]
                f.write(','.join(seq) + '\n')
        f.close()
    if NAnalizados_local == 0:
        PorcentajeBalanceados_local = 0
    else:
        PorcentajeBalanceados_local = NBalanceados_local / NAnalizados_local

    if NBalanceados_local == 0:
        PorcentajePerovskitas_local = 0
    else:
        PorcentajePerovskitas_local = NPerovskitas_local / NBalanceados_local

    NAnalizados.append( NAnalizados_local)
    NBalanceados.append( NBalanceados_local)
    NPerovskitas.append( NPerovskitas_local)

    PorcentajeBalanceados.append( PorcentajeBalanceados_local)
    PorcentajePerovskitas.append( PorcentajePerovskitas_local)


```

Código A.8: Código para generación de archivo de revisión $01_porcentajesFormacion.csv$

```
## GENERACION DE ARCHIVO PORCENTAJESFORMACION
```

```
    with open(DirResultados + formationName, 'w') as f:
        f.write('Formula, A1, A2, B1, B2, Nro. composiciones analizadas (a), Nro. composiciones balanceados (b), Porcentaje de cargas balanceadas (a/b),\
            Nro. Perov. Probables P(tau)>0.5 ( c), Porcentaje form. perov. de compuesto (c/b)\n')
        for i in range(len(formulas)):
            formula = formulas[i]
            A1 = A1s[i]
            A2 = A2s[i]
            B1 = B1s[i]
            B2 = B2s[i]
            n_comp = NBalanceados[i]
            total = NAnalizados[i]
            percentage = PorcentajeBalanceados[i]
            n_tau = NPerovskitas[i]
            tau_per = PorcentajePerovskitas[i]

            seq = [formula, A1, A2, B1, B2, total, n_comp, percentage, n_tau, tau_per]
            seq = [str(val) for val in seq]
            seq = [val if val != 'nan' else '' for val in seq]
            f.write(','.join(seq) + '\n')


```

A.1.3. ResultadosMP.py

Código A.9: Código de multiprocesamiento.

```
from multiprocessing import Process, Value, Pool, Semaphore, Array
from PredictPerovskites import *
import multiprocessing
import numpy as np
import math
from ctypes import c_wchar
from multiprocessing.sharedctypes import RawArray, Array
import itertools
import time

#Se necesitan dos funciones, la de calculation, que luego se ejecuta desde main()
#Funcion calculation: para creacion de cada archivo resultados de los compuestos
def calculation(tmp_dict, clf, iteracion, properties, comp_combinations, DirResultados,
    ↪ AVAILABLE_P,
            formulas,
            A1s, A2s, B1s, B2s,
            NAnalizados, NBalanceados, NPerovskitas,
            PorcentajeBalanceados, PorcentajePerovskitas):

    #valores para indicar en que compuesto va calculation()
    local_iteracion = iteracion.value
    iteracion.value+=1
    print('empezó el proceso %s/15470' %local_iteracion)

    #Variables base de cada compuesto
    A1 = tmp_dict['A1']
    A2 = tmp_dict['A2']
    B1 = tmp_dict['B1']
    B2 = tmp_dict['B2']
    X1 = tmp_dict['X1']
    X2 = tmp_dict['X2']
    compoundName = ''.join([A1, A2, B1, B2, X1])

    #Variables para PorcentajesFormacion
    globalIndex = iteracion.value - 2
    formulas[globalIndex].value = compoundName
    A1s[globalIndex].value = A1
    A2s[globalIndex].value = A2
    B1s[globalIndex].value = B1
    B2s[globalIndex].value = B2
    NAnalizados_local = len(comp_combinations)
```

```
    NBalanceados_local = 0
    NPerovskitas_local = 0


    fileName = compoundName + '.csv'
    fileDir = DirResultados + fileName

    with open(fileDir, 'w') as f:
        f.write(
            'Formula, x-y-delta,FTG t, Pred. FTG,NFT tau, Pred. NFT, Prob. formacion,A1,
    ↪ nA1,rA1,(1-x),A2,nA2,rA2,x,nA,rA,B1,nB1,rB1,(1-y),B2,nB2,rB2,y,nB,rB,X,nX,rX,(3-
    ↪ d),d\n')
        for set in comp_combinations:

            x = round(set[0], 1)
            y = round(set[1], 1)
            delta = round(set[2], 1)
            xydelta = str(x) + '-' + str(y) + '-' + str(delta)


            tmp_dict['formula'] = ('%s%s%s%s%s%s%s%sO3-%s' % \
                                (A1, round((1 - x), 1), A2, round(x, 1), \
                                 B1, round((1 - y), 1), B2, round(y, 1), \
                                 round(delta, 1)))
            tmp_dict['xydelta'] = xydelta

            classifier = PredictAABBXX6(A1, A2, B1, B2, X1, X2, x, y, delta)

            for prop in properties:
                value = getattr(classifier, prop)
                if prop == 't' or prop == 'tau':
                    value = round(getattr(classifier, prop), 3)
                tmp_dict[prop] = value
            tmp_dict['tau_prob'] = round(classifier.tau_prob(clf), 3)

            if not math.isnan(tmp_dict['tau']):
                NBalanceados_local += 1
                if tmp_dict['tau_pred'] == 1:
                    NPerovskitas_local += 1
                d = tmp_dict
                seq = [d['formula'], d['xydelta'],
                       d['t'], d['t_pred'],
                       d['tau'], d['tau_pred'], d['tau_prob'],
                       d['A1'], d['nA1'], d['rA1'], 1 - x,
                       d['A2'], d['nA2'], d['rA2'], x,
```

```
                    d['nA'], d['rA'],
                    d['B1'], d['nB1'], d['rB1'], 1 - y,
                    d['B2'], d['nB2'], d['rB2'], y,
                    d['nB'], d['rB'],
                    d['X1'][0], d['nX'], d['rX'], 3 - delta, delta]
            seq = [str(val) for val in seq]
            seq = [val if val != 'nan' else '' for val in seq]
            f.write(','.join(seq) + '\n')

        f.close()

    if NAnalizados_local == 0:
        PorcentajeBalanceados_local = 0
    else:
        PorcentajeBalanceados_local = NBalanceados_local / NAnalizados_local

    if NBalanceados_local == 0:
        PorcentajePerovskitas_local = 0
    else:
        PorcentajePerovskitas_local = NPerovskitas_local / NBalanceados_local

    NAnalizados[globalIndex] = NAnalizados_local
    NBalanceados[globalIndex] = NBalanceados_local
    NPerovskitas[globalIndex] = NPerovskitas_local

    PorcentajeBalanceados[globalIndex] = PorcentajeBalanceados_local
    PorcentajePerovskitas[globalIndex] = PorcentajePerovskitas_local

    print('termino el proceso %s/15470' %local_iteracion)
    AVAILABLE_P.release()


#Funcion main(): creacion de variables y preparacion de procesos a iterar
def main():
    # Valores string
    global formulas
    global A1s
    global A2s
    global B1s
    global B2s

    # Valores int
    global NAnalizados
    global NBalanceados
    global NPerovskitas
```

```
129
130     # Valores float
131     global PorcentajeBalanceados
132     global PorcentajePerovskitas
133
134     AVAILABLE_P = Semaphore(4)
135
136     X_anions = ['O']
137
138     A_cations = ['Ag', 'Ba', 'Ca', 'Cd', 'Ce', 'Dy', 'Er', 'Eu', 'Gd', 'Hf', 'Ho', 'K', 'La', 'Lu',
139             'Na', 'Nd', 'Pb', 'Pr', 'Rb', 'Sm', 'Sr', 'Tb', 'Th', 'Tl', 'Tm', 'Y', 'Yb']
140
141     B_cations = ['Al', 'Bi', 'Co', 'Cr', 'Cu', 'Fe', 'Ga', 'Ge', 'In', 'Ir', 'Li', 'Mg', 'Mn', 'Mo',
142             'Nb', 'Ni', 'Np', 'Os', 'Pd', 'Pt', 'Pu', 'Re', 'Rh', 'Ru', 'Sb', 'Sc', 'Sn', 'Ta',
143             'Te',  'Ti', 'U', 'V', 'W', 'Zn', 'Zr']
144
145     X = [0.1, 0.9, 0.1]
146     Y = [0.1, 0.9, 0.1]
147     DELTA = [0, 0.5, 0.1]
148
149     DirResultados = 'Resultados/'
150     formationName = '01_PorcentajesFormacion.csv'
151
152     # VARIABLES AUXILIARES PARA ITERACIONES
153     X1 = X_anions[0]
154     candidatesReg = []
155     formation_percentages = []
156     candidates = {}
157     candidates_dict = {}
158     iteracion = Value('i', 1)
159     count = Value('i', 0)
160     clf = PredictABX3('').calibrate_tau
161
162     #Variables para archivo PorcentajesFormacion
163     formulas = [RawArray(c_wchar, 10) for _ in range(15470)]
164     A1s = [RawArray(c_wchar, 3) for _ in range(15470)]
165     A2s = [RawArray(c_wchar, 3) for _ in range(15470)]
166     B1s = [RawArray(c_wchar, 3) for _ in range(15470)]
167     B2s = [RawArray(c_wchar, 3) for _ in range(15470)]
168     NAnalizados = multiprocessing.Array('i', 15470)
169     NBalanceados = multiprocessing.Array('i', 15470)
170     NPerovskitas = multiprocessing.Array('i', 15470)
171     PorcentajeBalanceados = multiprocessing.Array('d', 15470)
172     PorcentajePerovskitas = multiprocessing.Array('d', 15470)
173
```

```
174
175
176 # VARIABLES SECUNDARIAS CREADAS
177 x_vector = np.linspace(X[0], X[1], int(((X[1] - X[0]) / X[2]) + 1))
178 y_vector = np.linspace(Y[0], Y[1], int(((Y[1] - Y[0]) / Y[2]) + 1))
179 delta_vector = np.linspace(DELTA[0], DELTA[1], int(((DELTA[1] - DELTA[0]) / DELTA
    ↪ [2]) + 1))
180 comp_combinations = np.array(np.meshgrid(x_vector, y_vector, delta_vector)).T.
    ↪ reshape(-1, 3)
181 n_compositions = (len(x_vector) * len(y_vector) * len(delta_vector))
182 formationDir = DirResultados + formationName
183 properties = ['t', 't_pred',
184             'tau', 'tau_pred',
185             'A1', 'nA1', 'rA1',
186             'A2', 'nA2', 'rA2',
187             'nA', 'rA',
188             'B1', 'nB1', 'rB1',
189             'B2', 'nB2', 'rB2',
190             'nB', 'rB',
191             'nX', 'rX']
192
193
194 #GENERACION DE COMPUESTOS
195 for A1 in A_cations:
196     for A2 in A_cations:
197         for B1 in B_cations:
198             for B2 in B_cations:
199                 if (A1 != A2) and (B1 != B2) and (A1 == 'La'):
200                     candidate = ''.join([A1, A2, B1, B2, X1, '6'])
201                     candidatesReg.append(candidate)
202                     candidateDuplicate = ''.join([A1, A2, B2, B1, X1, '6'])
203                     if not candidateDuplicate in candidatesReg:
204                         tmp_dict = {}
205                         tmp_dict['A1'] = A1
206                         tmp_dict['A2'] = A2
207                         tmp_dict['B1'] = B1
208                         tmp_dict['B2'] = B2
209                         tmp_dict['X1'] = X1
210                         tmp_dict['X2'] = X1
211                         tmp_dict['formula'] = candidate
212                         candidates_dict[candidate] = tmp_dict
213
214
215 p_arr = []
216 for candidate in candidates_dict:
```

```
        tmp_dict = candidates_dict[candidate]

        p = Process(target=calculation, args=(tmp_dict, clf, iteracion, properties,
                                              comp_combinations, DirResultados, AVAILABLE_P,
                                              formulas,
                                              A1s, A2s, B1s, B2s,
                                              NAnalizados, NBalanceados, NPerovskitas,
                                              PorcentajeBalanceados, PorcentajePerovskitas))
        AVAILABLE_P.acquire()
        p.start()
        p_arr.append(p)

    for process in p_arr:
        process.join()


#ejecucion final del algoritmo completo
if ___name___ == '___main___':

    start = time.time()  # Tiempo de inicio del programa

    main() #Ejecucion funciones para generacion de resultados

    formation_percentages = [formulas,
                             A1s, A2s, B1s, B2s,
                             NAnalizados, NBalanceados, NPerovskitas,
                             PorcentajeBalanceados, PorcentajePerovskitas]


    with open('Resultados/01_PorcentajesFormacion.csv', 'w') as f:
        f.write('Formula, A1, A2, B1, B2, Nro. composiciones analizadas (a), Nro.
    ↪ composiciones balanceados (b), Porcentaje de cargas balanceadas (a/b),\
            Nro. Perov. Probables P(tau)>0.5 ( c), Porcentaje form. perov. de compuesto (c/
    ↪ b)\n')
        for i in range(len(formulas)):
            formula = formulas[i].value
            A1 = A1s[i].value
            A2 = A2s[i].value
            B1 = B1s[i].value
            B2 = B2s[i].value
            n_comp = NBalanceados[i]
            total = NAnalizados[i]
            percentage = PorcentajeBalanceados[i]
            n_tau = NPerovskitas[i]
            tau_per = PorcentajePerovskitas[i]
```

```
260
261         seq = [formula, A1, A2, B1, B2, total, n_comp, percentage, n_tau, tau_per]
262         seq = [str(val) for val in seq]
263         seq = [val if val != 'nan' else '' for val in seq]
264         f.write(','.join(seq) + '\n')
265
266 end = time.time()
267
268 print('Tiempo total: %s segundos ' % (end - start))
269 print('Tiempo total: %s minutos ' % ((end - start) / 60))
270 print('Tiempo total: %s horas ' % (((end - start) / 60) / 60))
271
272
273
274
275
276
```

A.1.4. Comparacion.py

Código A.10: Archivo para verificación de comparación de resultados entre esquemas.

```
from PredictPerovskites import *
from PredictPerovskitesO import *
import numpy as np
import time
import itertools

# PARAMETROS A DEFINIR
# ----------------------------------------------------------------------------------------------
# ELEMENTOS A UTILIZAR PARA LA GENERACION DE ARCHIVOS
X_anions = ['O']

A_cations = ['Ag', 'Ba', 'Ca', 'Cd', 'Ce', 'Dy', 'Er', 'Eu', 'Gd', 'Hf', 'Ho', 'K', 'La', 'Lu',
          'Na', 'Nd', 'Pb', 'Pr', 'Rb', 'Sm', 'Sr', 'Tb', 'Th', 'Tl', 'Tm', 'Y', 'Yb']

B_cations = ['Al', 'Bi', 'Co', 'Cr', 'Cu', 'Fe', 'Ga', 'Ge', 'In', 'Ir', 'Li', 'Mg', 'Mn', 'Mo',
          'Nb', 'Ni', 'Np', 'Os', 'Pd', 'Pt', 'Pu', 'Re', 'Rh', 'Ru', 'Sb', 'Sc', 'Sn', 'Ta',
          'Te', 'Te', 'Ti', 'U', 'V', 'W', 'Zn', 'Zr']


X = [0.5, 0.5, 0.1]
Y = [0.5, 0.5, 0.1]
DELTA = [0, 0, 0.1]

# VARIABLES DE DIRECTORIOS DE ARCHIVOS
DirResultados = 'Resultados/'
formationName = '02_Comparacion.csv'

# VARIABLES AUTOMATICAS
# ----------------------------------------------------------------------------------------------
# VARIABLES AUXILIARES PARA ITERACIONES
X1 = X_anions[0]
candidatesReg = []
formation_percentages = []
candidates = {}
candidates_dict = {}
results_N = {}
results_O = {}
iteracion = 0
count = 0
clf = PredictABX3('').calibrate_tau
```

```
start = time.time() # starting time

# VARIABLES SECUNDARIAS CREADAS
x_vector = np.linspace(X[0], X[1], int(((X[1] - X[0]) / X[2]) + 1))
y_vector = np.linspace(Y[0], Y[1], int(((Y[1] - Y[0]) / Y[2]) + 1))
delta_vector = np.linspace(DELTA[0], DELTA[1], int(((DELTA[1] - DELTA[0]) / DELTA
 ↪ [2]) + 1))
comp_combinations = np.array(np.meshgrid(x_vector, y_vector, delta_vector)).T.
 ↪ reshape(-1, 3)
n_compositions = (len(x_vector) * len(y_vector) * len(delta_vector))
fileDir = DirResultados + formationName
properties = ['t', 't_pred',
            'tau', 'tau_pred',
            'A1', 'nA1', 'rA1',
            'A2', 'nA2', 'rA2',
            'nA', 'rA',
            'B1', 'nB1', 'rB1',
            'B2', 'nB2', 'rB2',
            'nB', 'rB',
            'X',
            'nX', 'rX',
            't', 't_pred',
            'tau', 'tau_pred']
ver_count = 1

# CREACION DE ARCHIVOS
 ↪ ----------------------------------------------------------------------------------------------
##GENERACION DE COMPUESTOS CANDIDATOS
##GENERACION DE ARCHIVOS DE CADA COMPUESTO

for A1 in A_cations:
    for A2 in A_cations:
        for B1 in B_cations:
            for B2 in B_cations:
                if (A1 != A2) and (B1 != B2) and (A1 == 'La'):
                    candidate = ''.join([A1, A2, B1, B2, X1, '6'])
                    candidatesReg.append(candidate)
                    candidateDuplicate = ''.join([A1, A2, B2, B1, X1, '6'])
                    if not candidateDuplicate in candidatesReg:
                        tmp_dict = {}
                        tmp_dict['A1'] = A1
                        tmp_dict['A2'] = A2
                        tmp_dict['B1'] = B1
                        tmp_dict['B2'] = B2
                        tmp_dict['X1'] = X1
```

```
                    tmp_dict['X2'] = X1
                    tmp_dict['formula'] = candidate
                    candidates_dict[candidate] = tmp_dict
                    count += 1


with open(fileDir, 'w') as f:
    f.write('Formula, A1, A2, B1, B2,\
        nA1_O, nA2_O, nB1_O,nB2_O,nX_O,\
        nA1_N, nA2_N, nB1_N, nB2_N, nX_N,\
        tau N, \
        tau O, \n')
    for candidate in candidates_dict:
        # solo extraer el nombre y cosas
        tmp_dict = candidates_dict[candidate]
        A1 = tmp_dict['A1']
        A2 = tmp_dict['A2']
        B1 = tmp_dict['B1']
        B2 = tmp_dict['B2']
        X1 = tmp_dict['X1']
        X2 = tmp_dict['X2']
        compoundName = ''.join([A1, A2, B1, B2, X1])
        fileName = compoundName + '.csv'
        fileDir = DirResultados + fileName

        for set in comp_combinations:
            x = round(set[0], 1)
            y = round(set[1], 1)
            delta = round(set[2], 1)
            xydelta = str(x) + '-' + str(y) + '-' + str(delta)

            tmp_dict = candidates_dict[candidate]
            tmp_dict['formula'] = compoundName
            tmp_dict['xydelta'] = xydelta
            classifier = PredictAABBXX6(A1, A2, B1, B2, X1, X2, x, y, delta)
            for prop in properties:
                tmp_dict[prop] = getattr(classifier, prop)
            tmp_dict['tau_prob'] = classifier.tau_prob(clf)
            results_N[candidate] = tmp_dict

        tmp_dict = candidates_dict[candidate]
        tmp_dict['formula'] = compoundName
        classifier = PredictAABBXX6O(A1, A2, B1, B2, X1, X2)
        for prop in properties:
            tmp_dict[prop] = getattr(classifier, prop)
```

```
126         tmp_dict['tau_prob'] = classifier.tau_prob(clf)
127         results_O[candidate] = tmp_dict
128 
129         n = results_N[candidate]
130         o = results_O[candidate]
131         print('%s    (%s / %s)' % (n['formula'], ver_count, len(candidates_dict) ))
132         seq = [n['formula'], n['A1'], n['A2'], n['B1'], n['B2'],
133               n['nA1'], n['nA2'], n['nB1'], n['nB2'], n['nX'],
134               o['nA1'], o['nA2'], o['nB1'], o['nB2'], o['nX'],
135               n['tau'],
136               o['tau']]
137 
138         seq = [str(val) for val in seq]
139         seq = [val if val != 'nan' else '' for val in seq]
140         f.write(','.join(seq) + '\n')
141         ver_count += 1
142 
143 
144 
145 end = time.time()
146 
147 # total time taken
148 print('Tiempo total: %s segundos ' %(end-start))
149 print('Tiempo total: %s minutos ' %((end-start)/60))
150 print('Tiempo total: %s horas ' %(((end-start)/60)/60))
151 
152 
153 
```

A.1.5. Muestras.py

Código A.11: Archivo para verificación con muestras de perovskitas.

```
from PredictPerovskites import *

#VARIABLES DE DIRECTORIOS DE ARCHIVOS
DirResultados = 'Resultados/'
formationName = '03_Muestras.csv'
fileDir = DirResultados + formationName
clf = PredictABX3('').calibrate_tau


# MUESTRAS -------------------------------------
LaCo = PredictAABBXX6('La', 'La', 'Co', 'Co', 'O', 'O')
LaAl = PredictAABBXX6('La', 'La', 'Al', 'Al', 'O', 'O')
LaGa = PredictAABBXX6('La', 'La', 'Ga', 'Ga', 'O', 'O')
LaFe = PredictAABBXX6('La', 'La', 'Fe', 'Fe', 'O', 'O')
LaDy = PredictAABBXX6('La', 'La', 'Dy', 'Dy', 'O', 'O')
LaHo = PredictAABBXX6('La', 'La', 'Ho', 'Ho', 'O', 'O')
LaEr = PredictAABBXX6('La', 'La', 'Er', 'Er', 'O', 'O')
LaTm = PredictAABBXX6('La', 'La', 'Tm', 'Tm', 'O', 'O')
LaYb = PredictAABBXX6('La', 'La', 'Yb', 'Yb', 'O', 'O')
LaLu = PredictAABBXX6('La', 'La', 'Lu', 'Lu', 'O', 'O')
Singles = [LaCo, LaAl, LaGa, LaFe, LaDy, LaHo, LaEr, LaTm, LaYb, LaLu]

LaSrCoFe0 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.8, 0) #no
LaSrCoFe1 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.8, 0.1)
LaSrCoFe2 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.8, 0.2)
LaSrCoFe3 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.8, 0.3)
LaSrCoFe4 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.8, 0.4) #no
LaSrCoFe5 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.8, 0.5)
LaSrCoFe = [LaSrCoFe0, LaSrCoFe1, LaSrCoFe2, LaSrCoFe3, LaSrCoFe4, LaSrCoFe5]

LaSrCoFe90 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.9, 0)
LaSrCoFe91 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.9, 0.1)
LaSrCoFe92 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.9, 0.2)
LaSrCoFe93 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.9, 0.3)
LaSrCoFe94 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.9, 0.4)
LaSrCoFe95 = PredictAABBXX6('La', 'Sr', 'Co', 'Fe', 'O', 'O', 0.4, 0.9, 0.5)
LaSrCoFe9 = [LaSrCoFe90, LaSrCoFe91, LaSrCoFe92, LaSrCoFe93, LaSrCoFe94,
 ↪ LaSrCoFe95]

LaSrNiFe0 = PredictAABBXX6('La', 'Sr', 'Ni', 'Fe', 'O', 'O', 0.4, 0.9, 0)
LaSrNiFe1 = PredictAABBXX6('La', 'Sr', 'Ni', 'Fe', 'O', 'O', 0.4, 0.9, 0.1)
```

```
LaSrNiFe2 = PredictAABBXX6('La', 'Sr', 'Ni', 'Fe', 'O', 'O', 0.4, 0.9, 0.2)
LaSrNiFe3 = PredictAABBXX6('La', 'Sr', 'Ni', 'Fe', 'O', 'O', 0.4, 0.9, 0.3)
LaSrNiFe4 = PredictAABBXX6('La', 'Sr', 'Ni', 'Fe', 'O', 'O', 0.4, 0.9, 0.4)
LaSrNiFe5 = PredictAABBXX6('La', 'Sr', 'Ni', 'Fe', 'O', 'O', 0.4, 0.9, 0.5)
LaSrNiFe = [LaSrNiFe0, LaSrNiFe1, LaSrNiFe2, LaSrNiFe3, LaSrNiFe4, LaSrNiFe5]


LaCaMn0 = PredictAABBXX6('La', 'Ca', 'Mn', 'Mn', 'O', 'O', 1 - (3 / 8), 3 / 8, 0)
LaCaMn1 = PredictAABBXX6('La', 'Ca', 'Mn', 'Mn', 'O', 'O', 1 - (1 / 2), 1 / 2, 0)
LaCaMn2 = PredictAABBXX6('La', 'Ca', 'Mn', 'Mn', 'O', 'O', 1 - (2 / 3), 2 / 3, 0)
LaCaMn = [LaCaMn0, LaCaMn1, LaCaMn2]

LaCaCo0 = PredictAABBXX6('La', 'Ca', 'Co', 'Co', 'O', 'O', 0.8, 0.2, 0)
LaCaCo1 = PredictAABBXX6('La', 'Ca', 'Co', 'Co', 'O', 'O', 0.7, 0.3, 0)
LaCaCo = [LaCaCo0, LaCaCo1]

candidate_list = []
for el in Singles:
    candidate_list.append(el)
for el in LaSrCoFe:
    candidate_list.append(el)
for el in LaSrCoFe9:
    candidate_list.append(el)
for el in LaSrNiFe:
    candidate_list.append(el)
for el in LaCaMn:
    candidate_list.append(el)
for el in LaCaCo:
    candidate_list.append(el)

count = len(candidate_list)
print(count)


with open(fileDir, 'w') as f:
    f.write(
        'Formula, x-y-delta,FTG t, Pred. FTG,NFT tau, Pred. NFT, Prob. formacion,A1,
↪ nA1,rA1,(1-x),A2,nA2,rA2,x,nA,rA,B1,nB1,rB1,(1-y),B2,nB2,rB2,y,nB,rB,X,nX,rX,(3-
↪ d),d\n')

    for d in candidate_list:
        x = round(d.x, 2)
        y = round(d.y, 2)
        delta = round(d.delta, 2)

```

```
84          t = round(d.t, 4)
85          t_pred = d.t_pred
86
87          tau = round(d.tau, 4)
88          tau_pred = d.tau_pred
89          tau_prob = round(d.tau_prob(clf), 4)
90
91          nombrePerovskita = ('%s%s%s%s%s%s%s%sO3-%s' % \
92                          (d.A1, round((1 - x), 1), d.A2, round(x, 1), \
93                           d.B1, round((1 - y), 1), d.B2, round(y, 1), \
94                           round(delta, 1)))
95
96          xydelta = str(x) + '-' + str(y) + '-' + str(delta)
97          seq = [nombrePerovskita, xydelta,
98                t, t_pred,
99                tau, tau_pred, tau_prob,
100               d.A1, d.nA1, d.rA1, 1 - x,
101               d.A2, d.nA2, d.rA2, x,
102               d.nA, d.rA,
103               d.B1, d.nB1, d.rB1, 1 - y,
104               d.B2, d.nB2, d.rB2, y,
105               d.nB, d.rB,
106               d.X[0], d.nX, d.rX, 3 - delta, delta
107               ]
108
109         seq = [str(val) for val in seq]
110         seq = [val if val != 'nan' else '' for val in seq]
111         f.write(','.join(seq) + '\n')
112
113
114
115
```