Universidad Nacional Autónoma de México

Licenciatura en Tecnologías para la Información en Ciencias

Proyecto de Álgebra Lineal: Automatización con Python del proceso de Balanceo de Reacciones Químicas Usando Conceptos de Álgebra Lineal

Álgebra Lineal

Autor: Arnoldo Fernando Chue Sánchez

Junio 2023

Índice de secciones

1 Introducción

Con base en [4] podemos plantear los conceptos que necesitaremos para este proyecto.
De manera general, una definición simple de una reacción química es que se trata de un proceso en el que una o varias sustancias interactúan entre sí para dar lugar a nuevas sustancias. La o las sustancias con las que se inicia el proceso se denominan reactivos y las que se obtienen al final se conocen como productos.
Existen varias reglas para que se dé una reacción química. Sin embargo, para efectos de este trabajo solamente tomaremos en consideración dos: los elementos que componen a los reactivos deben ser exactamente los mismos que compongan a los productos (las reacciones nucleares son la excepción a esta regla, pero no las consideraremos para este trabajo). La segunda regla es que la misma cantidad de átomos de un elemento se debe tener tanto para los reactivos como para los productos.
Esta última regla la conocemos como la ley de Lavoisier y es la razón por la que balanceamos las reacciones químicas. Por balancear entenderemos entonces que es el proceso en el que agregamos la misma cantidad de átomos de todos los elementos tanto en los reactivos como en los productos. Esto se consigue multiplicando las moléculas de ambos lados de la reacción por ciertos coeficientes que debemos descubrir. Descubrir esos coeficientes es exactamente el proceso de balanceo.
Además, al hacerlo se deben tomar en consideración los subíndices que ya se tienen en la reacción, ya que esa es la cantidad inicial que tenemos de cada elemento. Cuando no se escriben los subíndices se consideran igual a 1. En el caso de que dentro de una molécula haya elementos dentro de un paréntesis y ese paréntesis tenga un subíndice, el subíndice de afuera del paréntesis multiplica a todos los subíndices de adentro.
Por ejemplo: si tenemos una molécula de ácido sulfúrico

$$H_2SO_4 \tag{1}$$

Podemos contar que hay 2 átomos de hidrógeno, 1 de azufre (de manera implícita) y 4 de oxígeno.
Ahora bien, si tenemos el caso de los paréntesis:

$$Al_2(SO_4)_3 \tag{2}$$

Contamos que hay 2 átomos de aluminio, 3 átomos de azufre y 12 de oxígeno. Para poder ingresar las ecuaciones en el código de Python que haremos, las moléculas y elementos que estén de la forma de 2 se deben transformar a la forma de 1
En este caso sería:

$$Al_2S_3O_{12} \tag{3}$$

Finalmente, cuando tenemos coeficientes para algún elemento o molécula se contabiliza de manera similar a 2. Es como si el coeficiente fuera un gran subíndice para toda la sustancia.
Ejemplo:

$$2\,Al_2(SO_4)_3 \tag{4}$$

4 es equivalente a

$$(Al_2(SO_4)_3)_2 \tag{5}$$

Si lo calculamos igual que calculamos 3 tenemos:

$$Al_4S_6O_{24} \tag{6}$$

Ya que tenemos clara la manera en la que se cuenta el número de átomos dentro de una reacción, podemos dar un ejemplo de una ecuación no balanceada y de cómo quedaría en su forma balanceada.

$$H_2 + O_2 \longrightarrow H_2O \tag{7}$$

En 7 podemos contar que en los reactivos hay 2 átomos de hidrógeno y 2 de oxígeno. Mientras que en el lado de los productos hay 2 átomos de hidrógeno y 1 de oxígeno. Por lo tanto no está balanceada.
Si hacemos el proceso de balanceo nos queda de la forma:

$$2\,H_2 + O_2 \longrightarrow 2\,H_2O \tag{8}$$

Aunque 8 es la misma reacción que 7, 8 sí está balanceada porque en los reactivos hay 4 átomos de hidrógeno y 2 de oxígeno, mientras que en los productos hay también 4 átomos de hidrógeno y 2 de oxígeno

2 Objetivos

- Utilizar los conceptos de álgebra lineal aprendidos durante el presente curso para implementar en Python un algoritmo que reciba como entrada una reacción química y la regrese balanceada.
- Programar en Python una clase que permita trabajar con una ecuación química como un objeto con sus respectivos métodos.

3 Planteamiento del problema

Los conceptos que a continuación se presentarán fueron vistos en clase. El libro donde se puede consultar la información es [2]. La idea original de este problema fue sacada de [1]. En ese libro se plantea que mediante álgera lineal se puede balancear cualquier ecuación química.
El primer paso es asignar una variable a cada molécula dentro de la ecuación. Por ejemplo:

$$\mathrm{HCl} + \mathrm{Na_3PO_4} \longrightarrow \mathrm{H_3PO_4} + \mathrm{NaCl} \tag{9}$$

Sea:

$$\begin{aligned} x_1 &= \mathrm{HCl} \\ x_2 &= \mathrm{Na_3PO_4} \\ x_3 &= \mathrm{H_3PO_4} \\ x_4 &= \mathrm{NaCl} \end{aligned}$$

La idea es encontrar los coeficientes de esas variables mediante la resolución de un sistema lineal y sustituirlos en la ecuación original para que quede balanceada.
El segundo paso es crear una matriz para resolver este sistema lineal. Esta matriz será de n filas por m columnas. Donde n está determinada por el número de elementos que participan en la reacción y m es el número de variables (moléculas) que tenemos. En un inicio la podemos hacer una matriz de ceros
Por lo tanto, el llenado de la matriz se hará de la siguiente forma:

1. Asignamos un elemento a cada fila.
2. Pasaremos por todas las filas iterando por sus entradas

3. Colocamos en esas entradas el número de átomos del elemento que aparecen en esa molécula. Si no aparece mantenemos el cero. Si es una variable que corresponde a un reactivo el número se deja positivo, si corresponde a un producto se pone el número pero negativo.

De esta forma ya tendríamos nuestra matriz para trabajar. Si quisiéramos la matriz aumentada del sistema se llenaría el lado derecho con ceros (vector nulo).
Ejemplo:
Utilizando la ecuación 9 con sus respectivas variables. La matriz (sin el lado derecho) sería:

$$\begin{bmatrix} 1 & 0 & -3 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 3 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 4 & -4 & 0 \end{bmatrix}$$

Donde la fila 1 corresponde al hidrógeno, la 2 al cloro, la 3 al sodio, la 4 al fósforo y la 5 al oxígeno.
Ya con la matriz formada el siguiente paso es llevarla a su forma escalonada reducida. Siguiendo con el ejemplo nos quedaría:

$$\begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1/3 \\ 0 & 0 & 1 & -1/3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Sabemos que el lado derecho de nuestra matriz es igual al vector nulo. Esto es sumamente importante ya que encontrar la solución al sistema lineal por método de Gauss-Jordan da el mismo resultado que tomar lo que ya llevamos y obtener la base del espacio nulo de la matriz.
En la siguiente sección pasaremos toda esta explicación a código de Python. Por lo tanto, optaremos por encontrar la base del espacio nulo porque computacionalmente es más fácil que automatizar la solución de un sistema lineal por Gauss-Jordan.
Además, a nivel de interpretación para este problema de química,

encontrar el espacio nulo tiene más sentido. Lo que estamos buscando son los coeficientes de las moléculas que nos den el mismo número de átomos por elemento de ambos lados de la ecuación.Estos coeficientes deben cumplir dos requisitos: ser distintos de cero y no ser fraccionarios.
Por el primer requisito es precisamente por lo que estamos buscando el espacio nulo: un vector que al multiplicarlo por nuestra matriz obtengamos el vector nulo sin que ese vector sea el mismo vector nulo. En cuanto al segundo requisito, es la razón por la que queremos la base del nulo: para que si llegamos a tener componentes fraccionarios, podamos escalar el vector hasta los valores positivos y enteros más chicos posibles.
Siguiendo con nuestro ejemplo, obtenemos la base del espacio nulo de nuestra matriz en forma escalonada reducida.

$$\left[\begin{array}{ccccc} 1 & 0 & 0 & -1 & |0 \\ 0 & 1 & 0 & -1/3 & |0 \\ 0 & 0 & 1 & -1/3 & |0 \\ 0 & 0 & 0 & 0 & |0 \\ 0 & 0 & 0 & 0 & |0 \end{array}\right]$$

La base del espacio nulo sería:

$$\left\{ \begin{pmatrix} 1 \\ 1/3 \\ 1/3 \\ 1 \end{pmatrix} \right\}$$

A continuación lo que se tiene que hacer es escalar la base del espacio nulo para eliminar los valores fraccionarios. La forma de hacer esto y que nos queden los enteros menores posibles es multiplicar el vector por el común denominador de sus componentes.
En el caso de nuestro ejemplo nos quedaría el vector de la forma:

$$\begin{pmatrix} 3 \\ 1 \\ 1 \\ 3 \end{pmatrix}$$

Teniendo ya nuestro vector con todas sus componentes enteras y

positivas, lo único que falta es asignar esos valores a las variables iniciales del problema y sustituirlas en la reacción original.
Concluyendo así con nuestro ejemplo:

$$3\,HCl + 1\,Na_3PO_4 \longrightarrow 1\,H_3PO_4 + 3\,NaCl \tag{10}$$

Nótese que la ecuación 10 ya quedó balanceada: 3 átomos de hidrógeno de cada lado, 3 de cloro de ambos lados, 3 de sodio, 1 de fósforo y 4 de oxígeno.
Será este procedimiento el que pasaremos a código para tener un algoritmo que balancee cualquier ecuación.

4 Desarrollo del código

Para comenzar, se decidió usar Python para este programa por las librerías que se disponen y por ser el lenguaje de programación principal que se usa en la carrera.
Sobre las librerías que se usarán:

- **NumPy.** [3] Usaremos NumPy para crear la matriz con la que vamos a trabajar en la parte de ingresar los valores a la matriz.
- **SymPy.** [5] Es la librería que más usaremos por las funciones que tiene. Nos permite seguir el algoritmo que hicimos lo más fácilmente posile (sin tener que preocuparnos tanto por las cuestiones computacionales).

Ahora bien, el desarrollo del código se abordó en dos partes: la primera fue crear una clase con la que pudiéramos trabajar la reacción como un objeto por sí mismo. En otras palabras, la clase nos permite recibir la ecuación como cadena de texto y los métodos de este nuevo objeto nos preparan para resolver este problema. La segunda parte ya es el desarrollo del problema. Se trata desde que formamos la matriz hasta que conseguimos el vector con los coeficientes.

4.1 Creación de la clase reacción

Lo primero que tenemos que hacer es dar el formato con el que el programa recibirá las ecuaciones:

- En primer lugar, para denotar la flecha de la reacción se deben usar los símbolos: =>

- En segundo lugar, para los símbolos de agregar tanto en la parte de los reactivos como en los productos se debe usar el símbolo: +
- En tercer lugar, las moléculas se deben escribir de la forma en que las describimos en 1, 3 o en 6
- Finalmente, entre cada molécula y su signo se debe dejar un sólo espacio, pero el último caracter no debe ser un espacio

Ejemplo de cómo se deben ingresar las ecuaciones:
HCl + Na3PO4 => H3PO4 + NaCl
Una vez definidas las reglas del input, podemos comenzar con la construcción de la clase.
En primer lugar definimos el constructor:

```
class Reaccion:
    def __init__(self, reaccion: str):
        self.reaccion = reaccion
        self.react = []
        self.prod = []
        self.elem = set()
        self.cantidad = {}
```

En self.reaccion almacenaremos la reacción, en self.react y self.prod serán listas donde almacenaremos las moléculas de los reactivos y de los productos, respectivamente. Para el caso de self.elem se decidió tomar una estructura de set para almacenarlos sin repetir y poder acceder rápidamente a ellos. Y en cuanto a self.cantidad será un diccionario donde las llaves serán los elementos y sus valores serán una lista de números. Esa lista de números será exactamente igual a su respectiva fila en la matriz.
Bajo esta lógica las primeras funciones dentro de la clase que debemos hacer son las de los reactivos y los productos:

```
    def reactivos(self):
        self.prod = self.reaccion.strip().split("=>")
        self.prod = self.prod[0].split("+")
        self.prod = [x.strip() for x in self.prod]
        return self.prod

    def productos(self):
        self.react = self.reaccion.strip().split("=>")
        self.react = self.react[1].split("+")
        self.react = [x.strip() for x in self.react]
        return self.react
```

Ambas trabajan y hacen exactamente lo mismo. Únicamente que una se queda con los reactivos y otra con los productos.
A continuación se siguió con la función que nos almacenará los elementos en el set descrito en el constructor:

```
    def elementos(self):
        x = ""
        for e in self.reaccion:
            if e == "=" or e == ">": break
            elif e == "+": pass
            elif e == " " or e.isnumeric():
                self.elem.add(x)
                x = ""
            elif e.isupper():
                self.elem.add(x)
                x = e
            else: x += e
        self.elem.add(x)
        self.elem.discard("")
        return self.elem
```

Ahora bien, antes de seguir con la función que nos construya el diccionario con la cantidad de elementos por molécula, debemos trabajar la reacción. Es cierto que ya le dimos instrucciones específicas al usuario para el momento en que ingrese una reacción. Sin embargo, vamos a hacer un procesamiento para poner los subíndices que son igual a 1 de forma explícita:

```
    def poner_1_explicitos(self):
        cont = 0
        for i, e in enumerate(self.reaccion):
            i += cont
            if i == 0 or e == "+" or e == ">": continue
            if e.isupper() and (self.reaccion[i-1].isupper()
                or self.reaccion[i-1].islower()):
                self.reaccion = self.reaccion[:i]+"1"+self.
                    reaccion[i:]
                cont += 1
                continue
            if e == " " and (self.reaccion[i-1].isupper() or
                self.reaccion[i-1].islower()):
                self.reaccion = self.reaccion[:i]+"1"+self.
                    reaccion[i:]
                cont += 1
                continue
        if not(self.reaccion[-1].isnumeric()): self.reaccion
            += "1"
        return self.reaccion
```

Para que se entienda mejor, un ejemplo de cómo trabaja esta función:
Input:
HCl + Na3PO4 => H3PO4 + NaCl
Output:
H1Cl1 + Na3P1O4 => H3P1O4 + Na1Cl1
Una vez hecho lo anterior, ahora sí podemos contar el numero de veces que aparece cada elemento en cada molécula y almacenarlo en el diccionario que habíamos descrito. Esta función es de las partes más importantes de todo el código porque si falla no podemos construir la matriz con la que trabajamos para llegar al balanceo.

```
    def cantidad_elementos_por_molecula(self,
        reaccion_1_explicitos:str):
        e, num, ladoizquierdo, moleculas = "", "", True, 1
        for x in reaccion_1_explicitos:
            if x.isnumeric(): num += x
            elif x.isalpha():
                if num != "":
                    if ladoizquierdo:
                        try: self.cantidad[e].append(int(num
                            ))
                        except: self.cantidad[e] = [int(num)
                            ]
                    else:
                        try: self.cantidad[e].append(-int(
                            num))
                        except: self.cantidad[e] = [-int(num
                            )]
                    num, e = "", x
                else: e += x
            elif x == ">": ladoizquierdo = False
            elif x == "+" or x == "=":
                if ladoizquierdo:
                    try: self.cantidad[e].append(int(num))
                    except: self.cantidad[e] = [int(num)]
                else:
                    try: self.cantidad[e].append(-int(num))
                    except: self.cantidad[e] = [-int(num)]
                num, e = "", ""
                for y in self.elem:
                    try:
                        if len(self.cantidad[y]) < moleculas
                            :
                            try: self.cantidad[y].append(0)
                            except: self.cantidad[y] = [0]
                    except: self.cantidad[y] = [0]
                moleculas += 1
        try: self.cantidad[e].append(-int(num))
```

```
32          except: self.cantidad[e] = [-int(num)]
33          for y in self.cantidad:
34              if len(self.cantidad[y]) < moleculas:
35                  try: self.cantidad[y].append(0)
36                  except: self.cantidad[y] = [0]
37          return self.cantidad
```

Como podemos ver, es la parte más extensa de todo el programa, ya que maneja muchas excepciones y es muy minuciosa para que todo el conteo se realice correctamente.
Con todo lo anterior ya podríamos comenzar a trabajar con nuestro problema. Sin embargo, vale la pena hacer una función adicional para nuestra clase que se encargue de validar si la ecuación que nos proporcionan ya está balanceada desde un inicio. Ésto con la finalidad de ahorrarnos todo el proceso del método que describimos que seguiremos y evitar errores por ciertas excepciones.

```
1       def esta_balanceada_inicialmente(self):
2           for x in self.cantidad:
3               if sum(self.cantidad[x]) != 0: return False
4           return True
```

Con esto podemos concluir nuestra clase reacción. Es cierto que podríamos agregar funciones adicionales que resultarían muy útiles para trabajos de química. Sin embargo, para efectos de este proyecto esta clase es más que suficiente.

4.2 Resolución del problema. Álgebra lineal aplicada

Lo primero que tenemos que hacer para trabajar en la parte principal de nuestro programa es tener un mensaje con las instrucciones para el input para los usuarios almacenada en una varible. Esta variable la podemos imprimir justo antes de solicitar el input.

```
1   if __name__ == '__main__':
2       aclaracionParaInputs = '''\nIndicaciones para los inputs
            :
3       Los subindices de los elementos van en tamanio normal
            despues del elemento en el que aparecen y deben ser
            estrictamente menores a 10.
4       Dejar espacio entre cada compuesto y simbolo de la
            reaccion.
5       Poner el simbolo => para indicar que se realiza la
            reaccion.
6   Ejemplo:
7       HCl + Na3PO4 => H3PO4 + NaCl\n'''
8       print(aclaracionParaInputs)
```

```
    reaccion = reaccion(input("Ingrese la reaccion:\n"))
```

Ahora lo que sigue es ejecutar todas nuestras funciones que hicimos para la clase reacción:

```
    reactivos = reaccion.reactivos()
    productos = reaccion.productos()
    elementos = reaccion.elementos()
    reaccion_explicita = reaccion.poner_1_explicitos()
    cantidad = reaccion.cantidad_elementos_por_molecula(
        reaccion_explicita)
```

Teniendo esto lo primero es validar si la reacción no está balanceada desde un inicio. En caso de que sí lo esté no sería necesario ejecutar el resto del programa. Pero si no lo está el primer paso es construir nuestra matriz.

```
    if reaccion.esta_balanceada_inicialmente(): print("\
        nEsta balanceada desde el inicio")
    else:
        M = np.matrix(len(elementos)*[([0]*(len(productos)+
            len(reactivos)))])
        for i, c in enumerate(cantidad): M[i] = cantidad[c].
            copy()
```

Nótese que primero creamos una matriz del tamaño que necesitamos pero con puros ceros. Hasta después es que la llenamos con los datos obtenidos del conteo.
El siguiente paso es llevarla a su forma escalonada reducida. SymPy tiene un método que hace esto en automático y que además nos da las columnas pivote. Por lo que primero debemos pasar nuestra matriz de NumPy a SymPy para poder aplicar el método

```
        f_e_r = sympy.Matrix(M)
        f_e_r, pivotes = f_e_r.rref()
```

Teniendo lo anterior, lo que sigue es sacar el espacio nulo, el cual almacenaremos como una matriz de SymPy

```
        espacio_nulo_fraccionario = sympy.Matrix(f_e_r.
            nullspace())
```

Hay que recordar que en este punto el espacio nulo que obtenemos con este método puede tener valores fraccionarios. Esto sería un problema a nivel computacional porque sería muy complicado escalarlo hasta su entero positivo más cercano. Sin embargo, SymPy

lo deja literalmente como fracciones, es decir, podemos acceder a los denominadores:

```
        denominadores = espacio_nulo_fraccionario.applyfunc(
            lambda x: x.as_numer_denom()[1])
        denominadores = set(denominadores)
```

Nótese cómo almacenamos los denominadores en un set para eliminar los repetidos y que al momento de buscar el denominador común nos sea más fácil.
Ahora bien, existe un caso que nos podría dar problemas al buscar un denominador común: que todos los denominadores sean iguales (que nuestro set sólo almacene un valor). Por lo que filtraremos ese caso y lo solucionaremos aumentando el mismo valor una vez mientras que transformamos el set en una lista para almacenar los dos. Esto no representa un problema para las personas cuando hacemos este ejercicio, pero sí para la computadora y para la función lcm() de SymPy

```
        if len(denominadores) == 1:
            denominadores = list(denominadores)
            denominadores.append(denominadores[0])
            comun_denominador = sympy.lcm(denominadores)
        else: comun_denominador = sympy.lcm(*set(
            denominadores))
```

Teniendo el común denominador es sólo cuestión de escalar nuestro espacio nulo fraccionario y transformar nuestros datos en enteros

```
        coeficientes_enteros = (comun_denominador *
            espacio_nulo_fraccionario).applyfunc(lambda x:
            int(x))
```

Por último, ya con el vector de los coeficientes de las moléculas, sólo debemos construir la cadena de la salida del algoritmo con nuestros datos y concluiremos nuestro programa exitosamente.

```
        reaccion_balanceada = ""
        moleculas = reactivos + productos
        for i, m in enumerate(moleculas):
            reaccion_balanceada += str(coeficientes_enteros[
                i]) + m + " "
            if i == len(reactivos) - 1: reaccion_balanceada
                += "=> "
            else: reaccion_balanceada += "+ "
        print("\nLa ecuacion balanceada es:")
        print(reaccion_balanceada[:-2])
```

5 Pruebas

A continuación se presentan algunas pruebas que se le hicieron al programa de este trabajo. Cada una de estas pruebas consiste en una reacción química y se presentan también los resultados que nos dio el programa.

Inputs:

$$\mathrm{H_2 + O_2 \longrightarrow H_2O} \tag{11}$$

$$\mathrm{HCl + Na_3PO_4 \longrightarrow H_3PO_4 + NaCl} \tag{12}$$

$$\mathrm{Na + H_2O \longrightarrow NaOH + H_2} \tag{13}$$

$$\mathrm{CH_4 + O_2 \longrightarrow CO_2 + H_2O} \tag{14}$$

$$\mathrm{C_6H_{12}O_6 + O_2 \longrightarrow CO_2 + H_2O} \tag{15}$$

$$\mathrm{AgNO_3 + NaCl \longrightarrow AgCl + NaNO_3} \tag{16}$$

$$\mathrm{HCl + NaHCO_3 \longrightarrow H_2O + CO_2 + NaCl} \tag{17}$$

$$\mathrm{H_2S + FeCl_3 \longrightarrow HCl + S + FeCl_2} \tag{18}$$

$$\mathrm{HNO_3 + Cu \longrightarrow H_2O + NO_2 + CuN_2O_6} \tag{19}$$

$$\mathrm{Cl_2 + Mg \longrightarrow MgCl_2} \tag{20}$$

Outputs:

$$\mathrm{2\,H_2 + 1\,O_2 \longrightarrow 2\,H_2O} \tag{21}$$

$$\mathrm{3\,HCl + 1\,Na_3PO_4 \longrightarrow 1\,H_3PO_4 + 3\,NaCl} \tag{22}$$

$$\mathrm{2\,Na + 2\,H_2O \longrightarrow 2\,NaOH + 1\,H_2} \tag{23}$$

$$\mathrm{1\,CH_4 + 2\,O_2 \longrightarrow 1\,CO_2 + 2\,H_2O} \tag{24}$$

$$\mathrm{1\,C_6H_{12}O_6 + 6\,O_2 \longrightarrow 6\,CO_2 + 6\,H_2O} \tag{25}$$

$$\mathrm{1\,AgNO_3 + 1\,NaCl \longrightarrow 1\,AgCl + 1\,NaNO_3} \tag{26}$$

$$\mathrm{1\,HCl + 1\,NaHCO_3 \longrightarrow 1\,H_2O + 1\,CO_2 + 1\,NaCl} \tag{27}$$

$$\mathrm{1\,H_2S + 2\,FeCl_3 \longrightarrow 2\,HCl + 1\,S + 2\,FeCl_2} \tag{28}$$

$$\mathrm{4\,HNO_3 + 1\,Cu \longrightarrow 2\,H_2O + 2\,NO_2 + 1\,CuN_2O_6} \tag{29}$$

$$\mathrm{1\,Cl_2 + 1\,Mg \longrightarrow 1\,MgCl_2} \tag{30}$$

Como se puede ver por los resultados, todos los balanceos se hicieron correctamente. El programa trabaja perfectamente porque el planteamiento con álgebra lineal para balancear las ecuaciones es correcto. En los casos de 26, 27 y 30 la ecuación inicial ya estaba balanceada y el programa lo identificó.

6 Conclusiones

Como se habló a lo largo de este curso, las aplicaciones del álgebra lineal en distintas áreas del conocimiento son muy extensas. En este proyecto el enfoque fue en una tarea repetitiva de química y quedó demostrado con nuestros resultados que la automatización de ese proceso fue un éxito. También es importante destacar y agradecer el enorme avance que hay hasta el día de hoy en cuanto a funciones y librerías que disponemos para llevar a cabo tareas matemáticas complejas. Sin las herramientas que usamos de las librerías NumPy y SymPy, la tarea a nivel computacional hubiera sido infinitamente más difícil.
Para trabajos futuros valdría la pena hacer más pruebas para tener un muestreo significativo sobre la tasa de efectividad de este algoritmo. Además, hay dos caminos claros para la continuidad de este proyecto a nivel computacional: el primero es trabajar para que el programa acepte las ecuaciones en distintos formatos (mejorar el proceso de limpieza de los inputs para ser menos estrictos). Buscar validar las ecuaciones mediante expresiones regulares sería un tema interesante para mejorar el programa.
El otro camino es ampliar la clase reacción que ya tenemos. Hay más propiedades e información sobre las reacciones que les serían de muchísima utilidad a los especialistas en esta área. Mientras más funciones se puedan automatizar con este programa les estaríamos ahorrando tiempo y esfuerzo dentro y fuera de los laboratorios.

7 Bibligrafía

References

[1] Chris Rorres Howard Anton. *Elementary Linear Algebra*. 2014. WILEY.

[2] David C. Lay. *Álgebra lineal y sus aplicaciones*. 2012. PEARSON EDUCACIÓN.

[3] Travis Oliphant. Numpy, 2005. `https://numpy.org/`.

[4] Bruce E. Bursten Julia R. Burdge Theodore L. Brown, H. Eugene LeMay. *Química. La ciencia central*. 2004. PEARSON EDUCACIÓN.

[5] Ondřej Čertík. Sympy, 2006. `https://www.sympy.org/en/index.html`.

[1] [2] [3] [4] [5]