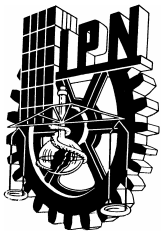

Instituto Politécnico Nacional

Escuela Superior de Ingeniería Química e
Industrias Extractivas
Secretaría de Investigación y Posgrado

Desarrollo de un programa de cómputo para la determinación del equilibrio químico en sistemas complejos

Tesis
que para obtener el grado de
Maestro en Ciencias con especialidad en
Ingeniería Metalúrgica

Presenta
Ing. César Ulises Santiago Cortés

Director
Dr. José Antonio Romero Serrano

México, D. F.
Septiembre 2006

SIP-14

INSTITUTO POLITECNICO NACIONAL

SECRETARIA DE INVESTIGACION Y POSGRADO

ACTA DE REVISION DE TESIS

En la Ciudad de México, D.F., siendo las 12:00 horas del día 18 del mes de Septiembre del 2006 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de La ESIQIE para examinar la tesis de grado titulada:

"DESARROLLO DE UN PROGRAMA DE COMPUTO PARA LA DETERMINACION DEL EQUILIBRIO QUIMICO EN SISTEMAS COMPLEJOS"

Presentada por el alumno:

SANTIAGO	CORTES	CESAR ULISES
Apellido paterno	materno	nombre(s)

Con registro: B | 0 | 3 | 1 | 2 | 7 | 1 |

Aspirante al grado de:

MAESTRO EN CIENCIAS CON ESPECIALIDAD EN INGENIERIA METALURGICA

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***SU APROBACION DE LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISION REVISORA

Director de Tesis
DR. JOSÉ ANTONIO ROMERO SERRANO

Lopez Hirata Victor M.
DR. VICTOR MANUEL LOPEZ HIRATA

DR. ALEJANDRO CRUZ RAMIREZ

DRA. ELSA MIRIAM ARCE ESTRADA

DR. JOSÉ FEDERICO CHÁVEZ ALCALA

DR. JORGE ROBERTO VARGAS GARCIA

EL PRESIDENTE DEL COLEGIO

DR. JOSÉ JAVIER CASTRO ARELLANO

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de **México** el día **29** del mes **Septiembre** del año **2005**, el (la) que suscribe **César Ulises Santiago Cortés** alumna (a) del Programa de **M. en C con Especialidad en Ingeniería Metalúrgica** con número de registro **B031271**, adscrito a **Secretaría de Investigación y Posgrado-ESIQIE** manifiesta que es autor (a) intelectual del presente trabajo de tesis bajo la dirección de **Dr. José Antonio Romero Serrano** y ceden los derechos del trabajo intitulado **Desarrollo de un programa de cómputo para la determinación del equilibrio químico en sistemas complejos**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y del director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección rosario **ulises osa@yahoo.com.mx**. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

César Ulises Santiago Cortés

Nombre y Firma

Dedicatorias

Como diría Don Juan, «...preocúpate y piensa antes de hacer cualquier cosa, pero una vez que la hagas échate a andar libre de preocupaciones y pensamientos...», yo sólo me preocupo y pienso en una: Regina.

Y como diría Don Juan pero no lo dijo, «...lo que nos hace desdichados es la necesidad...», ahora estoy lleno de desdichas: Cynthia, bienvenida.

Agradecimientos

Deseo expresar mi agradecimiento al Dr. Hallen y a su equipo de trabajo (PEP-IPN), al Programa Institucional de Formación de Investigadores (PIFI), al Consejo Nacional de Ciencia y Tecnología (CONACYT) y al Instituto Politécnico Nacional (IPN), por permitirme «comenzar» mis estudios de posgrado y por el apoyo económico para la realización de este trabajo.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos

a_i	actividad de la especie i
a_{ki}	k-ésimo elemento de la fórmula molecular de la especie i, elemento con indice ki de la matriz fórmula
$\underline{\mathbf{a}}$	vector formula a, compuesto de los elementos a_k
$\mathbf{A}$	matriz fórmula
$\mathbf{A}_{m\times n}$	matriz fórmula compuesta de m elementos y n especies
b_k	moles del elemento k en la cantidad base del sistema químico
$\underline{\mathbf{b}}$	vector de abundancia de elementos compuesto de los elementos b_k
$\mathbf{B}$	matriz atómica
${}^nC_m$	número de combinaciones de n objetos diferentes tomados de m en m
C_P	capacidad calorífica a presión constante
C_V	capacidad calorífica a volumen constante
exp	función exponencial base e
g_i	energía libre molar parcial de la especie i, potencial químico
gle	grados de libertad estequiométricos
G	energía libre de Gibbs
H	función de entalpía
H_T°	entalpía estándar a la temperatura T
$\mathbf{I}_m$	matriz identidad de dimensión $m\times m$
J	joules
K	Kelvin
ln	logaritmo base e
$L(\underline{\mathbf{x}})$	función lagrangiana
máx	máximo
máx!	maximizar
mín	mínimo
mín!	minimizar
mol	mol
M	número de elementos
n_i	número de moles de la especie i
δn_i	diferencia de moles entre dos estado de la especie i
n_t	número total de moles
$n_{t\alpha}$	número total de moles de la fase α
$\underline{\mathbf{n}}^\circ$	vector número de moles iniciales

$\underline{\mathbf{n}}$	vector de abundancia de especies con elementos n_i
$\underline{\delta\mathbf{n}}$	dirección de búsqueda, diferencia entre dos vectores de abundancia de especies
N	número de especies
$\mathbf{N}$	matriz estequiométrica completa, espacio nulo
P	presión
P_i	presión parcial de la especie i
S	función entropía
S°	entropía absoluta estándar
r	rango de una matriz
R	constante de los gases ideales, número máximo de ecuaciones linealmente independientes
$\mathfrak{R}$	conjunto de los números reales
$\mathfrak{R}^n$	espacio n-dimensional
T	temperatura
$\underline{\mathbf{v}}$	vector columna v
x_i	fracción mol de la especie i
γ_i	coeficiente de actividad de la especie i
γ_{ij}	coeficiente de actividad del soluto i en el solvente j medido en la escala de fracción mol henriana
γ_i°	coeficiente henriano de la especie i
δ_{ij}	delta generalizada de Kronecker
ϵ_{ij}	parámetro de interacción del soluto i sobre el soluto j
ζ_i	fracción de conversión de la especie i
λ_k	multiplicador k de Lagrange
$\underline{\lambda}$	vector multiplicadores de la Lagrange cuyos elementos son λ_k
μ_i	potencial químico
ν_j	columna j de la matriz estequiométrica, vector estequiométrico de la reacción j
ξ	avance de reacción
$\underline{\xi}$	vector avance de reacción
π	número de fases
π_m	número de fases de especies múltiples
π_u	número de fases de especies únicas
ψ_k	λ_k/RT
$\underline{\psi}$	vector columna cuyos elementos son ψ_k
ω	parámetro de etapa

Resumen

En este trabajo se desarrolló un programa de cómputo denominado `Ceqc` (cálculo del equilibrio químico complejo), creado en lenguaje `Python` para el cálculo del equilibrio químico en sistemas multifásicos y multicomponentes. Los dos objetivos principales del desarrollo del programa `Ceqc` son: 1) servir como material didáctico para la comprensión de los conceptos de equilibrio químico complejo y teoría de soluciones; 2) como herramienta de investigación en estudios de sistemas que contienen escorias, sales fundidas, aleaciones y mezclas de gases entre otras. `Ceqc` determina la composición en equilibrio de un sistema multifásico, basado en la minimización de la energía libre de Gibbs bajo condiciones de presión y temperatura específicas. El algoritmo empleado consiste en la formulación estequiométrica del equilibrio químico. El programa cuenta con una base de datos con información termodinámica de 1881 especies puras. Además, permite la adición de información termodinámica de especies no contempladas en esta base de datos. Para el uso de `Ceqc` sólo se requiere, por parte del usuario, los nombres y moles iniciales de las especies químicas que constituyen el sistema, la temperatura y la presión. `Ceqc`, genera entonces, el balance de masa, por fase, del sistema a la temperatura y presión establecidas. En cada fase puede emplearse un modelo de solución diferente, que puede o no ser ideal, por medio de un módulo o «script» escrito en lenguaje `Python`. Finalmente, se realizan comparaciones, de los cálculos obtenidos, con `Ceqc` y otros programas comerciales de cálculo de equilibrio químico como `Fact` y `Chemsage` obteniéndose resultados satisfactorios. El programa `Ceqc` tiene adaptado el modelo termodinámico de parámetros de interacción para el estudio de las soluciones diluidas como es el caso de los aceros líquidos.

Abstract

In this work a computer program named `Ceqc` (Calculation of the Complex Chemical Equilibrium) was developed. It was created in `Python` language for the calculation of the chemical equilibrium in both multiphasic and multicomponent systems. The main objectives of this software are: 1) to serve as didactic material to understand the complex chemical equilibrium and theory of chemical solutions; 2) as a tool in the research of complex systems containing slags, fused salts, alloys, mixtures of gases, etcetera. `Ceqc` determines the composition in equilibrium of a multiphasic system, based on the minimization of the Gibbs free energy under given constrains of pressure and temperature. The algorithm used in this work consists on the stoichiometric formulation of the chemical equilibrium. The program has a database with thermodynamic information of 1881 pure species. The program allows the addition of thermodynamic information of species not included in this database. The information required to run `Ceqc` are the names and initial composition of the species that constitute the system, temperature and pressure. `Ceqc`generates then the balance of mass, for phase, of the system at the given temperature and pressure. `Ceqc` enables the user to select a different model for each phase, that can or not to be ideal, by means of a module or «script» written in `Python` language. `Ceqc` has been tested by comparison of the results of systems with liquid steel calculated by commercial thermodynamic software such as `Fact` (Facility for the Analysis of Chemical Thermodynamics).

Introducción

El objetivo perseguido al realizar un cálculo de equilibrio químico sobre un sistema es: *encontrar la composición de las especies que componen el sistema cuando éste se encuentra en equilibrio*. Por supuesto, el cálculo de equilibrio químico en un sistema real puede implicar no sólo encontrar la composición de las especies en una fase única (sistemas monofásicos), sino en distintas fases (sistemas multifásicos) de aquí que tenga que determinarse además, la composición del sistema en términos de las fases que lo componen. La importancia de determinar la composición en equilibrio de un sistema químico está dada por la gran cantidad de áreas del conocimiento en las que se ven envueltos tales cálculos, por mencionar algunas, en *química orgánica, inorgánica, ambiental* y *analítica*, en *cinética química*, en procesos de *conversión de energía*, *propulsores para cohetes*, en *metalurgia* y en general en *procesos químicos*.

Los principios teóricos en los que se basa el equilibrio químico, dentro del marco de las definiciones actuales, se establecieron a principios del siglo XX. Hasta antes de 1940 se usaban únicamente dos métodos para calcular el equilibrio químico. Uno de ellos se basaba en la suposición de que pocas especies moleculares estarán presentes en la composición final de equilibrio; el otro, en ensayos de prueba y error. Alrededor de 1940, los algoritmos empleados para el cálculo de equilibrio químico se realizaban teniendo en mente métodos de cálculo manual y giraban alrededor de las expresiones de la constante de equilibrio escritas para un conjunto de ecuaciones estequiométricas en términos de concentraciones. Y casi siempre implicaba la solución de un grupo de ecuaciones no lineales, que por lo general era igual a la diferencia entre el número de especies y el número de elementos. Esto traía como consecuencia, al no haber computadoras electrónicas, cálculos laboriosos y ecuaciones casi imposibles de manipular si la cantidad de especies y elementos involucrados en el cálculo era grande.

El avance en los procedimientos computacionales de cálculo manual para equilibrio se debe principalmente al desarrollo de cohetes en Alemania durante la Segunda Guerra Mundial. Aunque, los procedimientos eran restringidos a sistemas específicos, las manipulaciones numéricas y algebraicas se simplificaron y optimizaron. En la década de 1950 con el rápido desarrollo de las computadoras electrónicas y de los programas espaciales, se generan algoritmos de propósitos generales y se considera el problema como un problema de optimización

no lineal. A partir de ahí se han desarrollado algoritmos *minimizando directamente la energía libre*, es decir, sin utilizar reacciones estequiométricas (*métodos de minimización directa*) en contraposición de los métodos llamados de *constante de equilibrio* o *indirectos*. Pero los métodos de constante de equilibrio no han dejado de desarrollarse, sobre todo relacionados a la minimización de la energía libre. La diferencia esencial con los métodos indirectos es que, como variables independientes se emplean los avances de reacción. Mientras que en los métodos directos, por lo general, las restricciones de balance de materia se incorporan al sistema de ecuaciones no lineales utilizando, por ejemplo, multiplicadores de Lagrange.

Desde sus orígenes, el equilibrio químico ha tenido un objetivo eminentemente práctico. Los cálculos de equilibrio químico respondían en forma directa a problemas de carácter industrial, bélico o científico. Pero, el estudio del equilibrio químico por sí mismo ya tiene interés académico, en él se ven involucrados conceptos termodinámicos de toda índole como puede ser la energía de reacción o la teoría de soluciones. Desarrollar un programa de cómputo que realice cálculos de equilibrio químico como el que se desarrolla en este trabajo persigue dos objetivos. El primero de ellos es que pueda servir como herramienta de investigación, enfocado hacia la metalurgia, en estudios de sistemas que contienen escorias, sales fundidas, aleaciones y mezclas de gases entre otros. El segundo, desde el punto de vista académico, es servir como material didáctico para la comprensión de los conceptos de equilibrio químico complejo y teoría de soluciones, entre otros.

Algunos investigadores han empleado algoritmos desarrollados por ellos mismos o por otros y han escrito algún programa de cómputo que determina principalmente la composición en equilibrio de un sistema químico, empleando desde modelos ideales hasta modelos no ideales de soluciones. Como resultado de ello en la actualidad existen algunos programas comerciales (p. ej., `SolgasMix`, `HSC Chemistry`, `Fact`, `Thermo-CalcSoftware`), que permiten computar el estado en equilibrio de un sistema químico. Todos ellos varían en complejidad, opciones de ejecución, funcionalidades, tamaño de las bases de datos, costos, etcétera. La principal desventaja es su costo; por ejemplo, el costo del sistema `Fact` es del orden de 5000 dólares, y sólo representa una *caja negra* para el usuario, es decir, no se ve y no puede modificarse el algoritmo de cálculo, no pueden hacerse cambios substanciales a la manera de ejecución y no pueden adicionarse características no contempladas por el programador.

En este trabajo se desarrolló un programa para el cálculo del equilibrio químico, escrito en su totalidad en lenguaje `Python`. Este programa, que se ha dado el nombre de `Ceqc`

(Cálculo del equilibrio químico complejo), permite calcular la composición de equilibrio entre fases, condensadas y no condensadas, y entre substancias de un sistema químico. La idea de escribir `Ceqc` es crear un programa fácilmente modificable y extensible a otras aplicaciones o cálculos (p. ej., cinéticos); en el que puedan ser adicionadas características nuevas por el usuario final. El algoritmo que emplea `Ceqc`, VCS, está basado en la formulación estequiométrica del equilibrio químico.

`Ceqc` permite utilizar un modelo de solución diferente para cada fase en base a los modelos incluidos dentro del programa, pero es posible utilizar modelos proporcionados directamente por el usuario. Este programa cuenta con una base de datos de casi 1900 especies puras y puede ampliarse según las necesidades del usuario. En este trabajo se muestra el desarrollo e implementación del programa así como todas las características necesarias para emplearlo como un programa de cálculo de equilibrio químico multipropósito.

Antecedentes Teóricos

1 Definiciones

En esta sección se definen algunos términos que se utilizan comúnmente en las exposiciones de equilibrio químico.

1.1 Especie química

Una entidad química que es diferenciable de otras por su estequiometría, estructura molecular[1] o la fase en que existe, se denomina *especie química*. Las siguientes entidades químicas se consideran especies químicas diferentes: $Zn_{sólido}$, Zn_{gas},Fe_{α}, Fe_{γ}, Cu_2O, CuO, H_2SO_4, S, S_2, etcétera.

1.2 Substancia química

Una entidad química que es diferenciable de otras por su estequiometría se denomina *substancia química*. Así, Fe_{α}, Fe_{γ}, $Fe_{líquido}$, Fe_{gas}, se consideran como una misma substancia química.

1.3 Sistema químico

Es una pareja de dos conjuntos ordenados, uno de especies químicas y el otro de elementos químicos; que se representa en la siguiente forma.

$$\{(A_1, A_2, \ldots, A_i, \ldots, A_n),\ (E_1, E_2, \ldots, E_j, \ldots, E_m)\}$$

aquí A_i es la i-ésima especie[2] y E_j es el j-ésimo elemento. El orden elegido es irrelevante, para la definición de sistema químico, pero una vez establecido debe permanecer fijo. Ejemplos de dos sistemas químicos son:

$$\{(H_2SO_4),\ (H, S, O)\}$$

[1] En caso de fases alotrópicas.
[2] Especie química.

$$\{(ZnO, CO, CH_4, H_2, Zn, CO_2, H_2O),\ \ (Zn, O, C, H)\}$$

1.4 Vector fórmula

El *vector fórmula*, $\underline{\mathbf{a}}$, es un vector columna formado con los subíndices de la fórmula molecular de una especie química. De esta manera, para Na_3AlCl_6 se tiene que $\underline{\mathbf{a}} = (3, 1, 6)^{\mathrm{T}}$. El i-ésimo elemento de $\underline{\mathbf{a}}$ se representa como a_i. Casi siempre, el vector fórmula se define en relación a los elementos presentes en algún sistema químico[3], por ejemplo, en el sistema $\{(H_2O, H_2, Fe, FeO, CO, CO_2),\ \ (H, O, Fe, C)\}$ el vector fórmula de cada especie será:

$$\underline{\mathbf{a}}_{H_2O} = \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \underline{\mathbf{a}}_{H_2} = \begin{pmatrix} 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \underline{\mathbf{a}}_{Fe} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \underline{\mathbf{a}}_{FeO} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$\underline{\mathbf{a}}_{CO} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad \underline{\mathbf{a}}_{CO_2} = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 1 \end{pmatrix} \qquad \text{con} \qquad \underline{\mathbf{a}}_{\text{especie}} = \begin{pmatrix} a_H \\ a_O \\ a_{Fe} \\ a_C \end{pmatrix}$$

1.5 Matriz fórmula

Es una matriz $m \times n$ en donde la columna i está formada por los elementos del vector fórmula de la especie i del sistema químico X. Así, en la matiz fórmula $\mathbf{A}_{m\times n}$, del sistema X, m representa el número de elementos y n el número de especies. Para el sistema $\{(H_2O, H_2, Fe, FeO, CO, CO_2),\ \ (H, O, Fe, C)\}$, por ejemplo, se tiene una matriz fórmula dada por:

$$\mathbf{A} = \begin{array}{c} \\ H \\ O \\ Fe \\ C \end{array} \begin{array}{c} \begin{array}{cccccc} H_2O & H_2 & Fe & FeO & CO & CO_2 \end{array} \\ \begin{pmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

[3] En este trabajo, se empleará esta última definición.

1.6 Matriz atómica

Es la matriz traspuesta a la matriz fórmula. La matriz atómica del sistema anterior será:

$$\mathbf{B} = \mathbf{A}^{\mathrm{T}} = \begin{array}{c} \\ \mathrm{H_2O} \\ \mathrm{H_2} \\ \mathrm{Fe} \\ \mathrm{FeO} \\ \mathrm{CO} \\ \mathrm{CO_2} \end{array} \begin{array}{c} \begin{array}{cccc} \mathrm{H} & \mathrm{O} & \mathrm{Fe} & \mathrm{C} \end{array} \\ \begin{pmatrix} 2 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 1 \end{pmatrix} \end{array}$$

1.7 Vector de abundancia de especies

Con referencia a un sistema químico, un *vector de abundancia de especies*, $\underline{\mathbf{n}}$, es un vector columna cuyo elemento, $n_i \geq 0$, representan el número de moles de la especie i en un determinado estado del sistema. Para el sistema $\{(\mathrm{CO_2}, \mathrm{CO}, \mathrm{O_2}, \mathrm{C}),\ (\mathrm{C}, \mathrm{O})\}$, el vector $\underline{\mathbf{n}} = (1.0, 2.0, 5.0, 0.0)^{\mathrm{T}}$ indica que se tiene 1.0 mol de $\mathrm{CO_2}$, 2.0 mol de CO, 5.0 mol de $\mathrm{O_2}$ y 0.0 mol de C.

1.8 Vector de abundancia de elementos

Es un vector columna, de números reales positivos, que representa el número de moles de elementos de sistema químico. Si el sistema no intercambia materia con el medio el *vector de abundancia de elementos*, $\underline{\mathbf{b}}$, permanecerá constante. No cualquier i-upla puede ser un vector de abundancia de elementos. Por ejemplo, para el sistema $\{(\mathrm{SO_2}, \mathrm{SO_3}, \mathrm{O_2}),\ (\mathrm{S}, \mathrm{O})\}$, es válido $\underline{\mathbf{b}} = (2, 8)^{\mathrm{T}}$ pero $\underline{\mathbf{b}} = (3, 2)^{\mathrm{T}}$ no lo es. En general, para este sistema siempre debe cumplirse: $b_{\mathrm{O}} \geq 2.5 b_{\mathrm{S}}$.

1.9 Vector estequiométrico

Es el vector $\underline{\nu}_j$, para el que se cumple, $\mathbf{A}\underline{\nu}_j = \underline{\mathbf{0}}$. Siendo los componentes de $\underline{\nu}_j$ no todos ceros. Por ejemplo, tres los vectores estequiométricos para la matriz fórmula

$$\mathbf{A} = \begin{pmatrix} 0 & 2 & 2 & 4 & 0 & 2 & 2 & 4 & 4 \\ 2 & 6 & 4 & 8 & 2 & 6 & 4 & 8 & 8 \\ 1 & 1 & 2 & 2 & 1 & 1 & 2 & 2 & 4 \end{pmatrix}$$

son:

$$\underline{\nu}_1 = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \underline{\nu}_2 = \begin{pmatrix} 0 \\ 0 \\ -2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \underline{\nu}_3 = \begin{pmatrix} -0.4439 \\ +0.3234 \\ -0.5019 \\ -0.1976 \\ +0.0487 \\ +0.0718 \\ -0.2206 \\ -0.1976 \\ +0.5588 \end{pmatrix}$$

1.10 Matriz estequiométrica completa

A propósito de la terminología empleada en equilibrio químico, recibe el nombre de *matriz estequiométrica completa*, $\mathbf{N}$, el concepto que en álgebra lineal es llamado *espacio nulo* o *nulidad*. El espacio nulo de una matriz $\mathbf{A}_{m\times n}$, contiene todos los vectores n en $\mathbf{N}$, tales que $\mathbf{AN} = \mathbf{0}$. De esta manera se tiene que:

$$\mathbf{N} = \{\underline{\nu}_j\} = (\,\underline{\nu}_1 \quad \underline{\nu}_2 \quad \cdots \quad \underline{\nu}_q\,)$$

donde $\underline{\nu}_j$ es un vector estequiométrico. Cada uno de los vectores, $\underline{\nu}_j$, es linealmente independiente con respecto a los demás vectores que constituyen la matriz estequiométrica completa. Además, $\mathbf{N}$ es una matriz de dimensiones $n \times gle$ donde

$$gle = n - \text{rango}\,(\mathbf{A})$$

el número gle es conocido como *grados de libertad estequiométricos* por ser el número de ecuaciones linealmente independientes El cálculo de $\mathbf{N}$, junto con algunos otros detalles acerca de este concepto se describen en la **sección 2.5**.

1.11 Rango de una matriz

En una matriz $\mathbf{A}$ el número r de vectores columna linealmente independientes es igual al número de vectores fila linealmente independientes. El número r es llamado *rango de la matriz* y se denota por

$$\text{rango}\,(\mathbf{A}) = r$$

1.12 Dependencia lineal

Una combinación lineal de n vectores, $\underline{\mathbf{v}}_1, \ldots, \underline{\mathbf{v}}_k$ es

$$c_1\underline{\mathbf{v}}_1 + c_2\underline{\mathbf{v}}_2 + \cdots + c_k\underline{\mathbf{v}}_k$$

y se llama *no trivial* cuando no todos los coeficientes $c_1, \ldots, c_k$ son iguales a cero. Si todos los coeficientes son iguales a cero se denomina *trivial*.

Un conjunto de n vectores, $\underline{\mathbf{v}}_1, \ldots, \underline{\mathbf{v}}_k$, es *linealmente dependiente* si $\underline{\mathbf{0}}$ es una combinación no trivial de esos vectores. Es decir, la dependencia lineal de dos vectores significa que al menos uno de ellos puede ser expresado en términos del otro.[4]

1.13 Independencia lineal

Una conjunto de n vectores, $\underline{\mathbf{v}}_1, \ldots, \underline{\mathbf{v}}_k$, es *linealmente independiente* si no es linealmente dependiente. Es lo mismo que decir que la única combinación lineal de $\underline{\mathbf{0}}$ en función de $\underline{\mathbf{v}}_1, \ldots, \underline{\mathbf{v}}_k$ es la trivial o que

$$c_1\underline{\mathbf{v}}_1 + c_2\underline{\mathbf{v}}_2 + \cdots + c_k\underline{\mathbf{v}}_k = \underline{\mathbf{0}}$$

se cumple sólo para $c_1 = c_2 = \cdots = c_k = 0$.

[4] Un conjunto $\underline{\mathbf{v}}_1, \ldots, \underline{\mathbf{v}}_k$, de n vectores es linealmente dependiente si lo es como sucesión (se permite repetición de vectores).

2 Estequiometría química

2.1 Restricciones de los sistemas cerrados

Un sistema cerrado no intercambia materia con sus alrededores aunque puede intercambiar energía. Su importancia en los cálculos de equilibrio estriba en que las condiciones de equilibrio termodinámico se aplican principalmente a este sistema.

Un sistema cerrado se define por un conjunto de ecuaciones de *abundancia de elementos* que expresan la conservación de los elementos químicos que constituyen las especies del sistema. Para cada elemento existe una ecuación que adopta la siguiente forma:

$$\sum_{i=1}^{N} a_{ki} n_i = b_k \qquad \text{para cada} \qquad k = 1, 2, \ldots, M \tag{2.1}$$

donde a_{ki} es el índice del k-ésimo elemento de la fórmula molecular de la especie i, es decir cada uno de los componentes del vector fórmula $\underline{\mathbf{a}}$, así n_i es cada uno de los componentes del vector, $\underline{\mathbf{n}}$, de abundancia de especies y b_k cada componente del vector, $\underline{\mathbf{b}}$, de abundancia de elementos. M es el número de elementos del sistema y N es el número de especies. Si el sistema de ecuaciones anterior se escribe de tal manera que exprese el cambio de un estado de composición a otro, debe cumplirse siempre que:

$$\sum_{i=1}^{N} a_{ki} \delta n_i = 0 \qquad \text{para cada} \qquad k = 1, 2, \ldots, M \tag{2.2}$$

donde δn_i es el cambio en el número de moles de la i-ésima especie entre dos estados de composición del sistema. En forma vectorial, las ecuaciones de abundancia de elementos **2.1** y **2.2** se escriben como:

$$\mathbf{A}\underline{\mathbf{n}} = \underline{\mathbf{b}} \tag{2.3}$$

y

$$\mathbf{A}\underline{\delta\mathbf{n}} = \underline{\mathbf{0}} \tag{2.4}$$

El hecho de que **b** esté fijo es lo que caracteriza a un sistema cerrado. Cualquiera de estas últimas cuatro ecuaciones expresa la restricción de un sistema cerrado.

2.2 Grado de conversión de una reacción

El grado de conversión de una reacción química generalmente es representado por el concepto de *avance de reacción*. Este concepto sirve para analizar el equilibrio de las reacciones químicas. Por ejemplo, en un sistema ocurre la reacción:

$$a\mathrm{A}_{(g)} \quad + \quad b\mathrm{B}_{(g)} \qquad \rightleftarrows \qquad c\mathrm{C}_{(g)} \quad + \quad d\mathrm{D}_{(g)}$$

en estado termodinámico inicial llamado E_0 donde se tienen n°_A, n°_B, n°_C, n°_D moles de las especies A, B, C, D. Suponiendo que con el tiempo la reacción ha progresado hasta alcanzar el estado termodinámico E, que no necesariamente es un estado de equilibrio, donde se tienen n_A, n_B, n_C, n_D moles de las especies. El progreso de la reacción del estado E_0 al E puede ser expresado cuantitativamente en términos del *avance de reacción*, ξ, que se define como:

$$\xi = \frac{\Delta n_\mathrm{A}}{a} = \frac{\Delta n_\mathrm{B}}{b} = \frac{\Delta n_\mathrm{C}}{c} = \frac{\Delta n_\mathrm{D}}{d} \tag{2.5}$$

en general

$$\xi = \frac{\Delta n_i}{\nu_i} \tag{2.6}$$

donde $\Delta n_i = n_i - n_i^\circ$ para toda especie i, y ν_i es el coeficiente estequiométrico de la i-ésima especie en la reacción. Nótese que ν_i es positiva para los productos y negativa para los reactantes. En ocasiones, se expresa el progreso de la reacción en términos de *la fracción de conversión* de una especie reactante. Suponiendo que ζ_A representa la fracción de conversión de la especie A debido al avance de reacción ξ, éste estará definido como:

$$\zeta_\mathrm{A} = \frac{\Delta n_\mathrm{A}}{d} = \frac{a\xi}{n^\circ_\mathrm{A}} \tag{2.7}$$

De la ecuación anterior pueden deducirse las moles finales de cada especie

$$n_\mathrm{A} = n^\circ_\mathrm{A}(1 - \zeta_\mathrm{A}) = n^\circ_\mathrm{A} - a\xi$$

$$n_B = n_B^\circ - \left(\frac{b}{a}\right) n_A^\circ \zeta_A = n_B^\circ - b\xi$$

$$n_C = n_C^\circ + \left(\frac{c}{a}\right) n_A^\circ \zeta_A = n_C^\circ + c\xi$$

$$n_D = n_B^\circ + \left(\frac{d}{a}\right) n_A^\circ \zeta_A = n_D^\circ + d\xi$$

El número n de moles totales es

$$n = n^\circ + n_A^\circ \zeta_A \left(\frac{c}{a} + \frac{d}{a} - \frac{b}{a} - 1\right) = n^\circ + \xi(c + d - b - a) \tag{2.8}$$

La fracción de conversión, ζ_A, a diferencia del avance de reacción, ξ, es específica para una especie dada, lo que limita su utilidad.

2.3 Reacciones simultáneas

Cuando dos reacciones ocurren simultáneamente en un sistema isotérmico, cada reacción estará asociada con un avance de reacción diferente. Para ilustrar esto, se consideran las siguientes reacciones simultáneas:

$$CH_{4(g)} + H_2O_{(g)} \rightleftarrows CO_{(g)} + 3H_{2(g)} \tag{2.9}$$

$$CO_{2(g)} + H_{2(g)} \rightleftarrows CO_{(g)} + H_2O_{(g)} \tag{2.10}$$

Al alcanzar estas reacciones el estado termodinámico E_1 y ser el número de moles, de cada especie, diferente al número de moles del estado inicial. El número de moles para la i-ésima especie estará dado por:

$$n_i = n_i^\circ + \nu_{i1}\xi_1 + \nu_{i2}\xi_2 \tag{2.11}$$

donde ξ_j es el avance de reacción para la j-ésima reacción y ν_{ij} es el coeficiente estequiométrico para la i-ésima especie en la j-ésima reacción. En el caso general en que, en R reacciones están involucradas N especies, la ecuación tendrá la forma:

$$n_i = n_i^\circ + \sum_{j=1}^{R} \nu_{ij}\xi_j \qquad \text{con} \qquad i = 1, 2, \ldots, N \tag{2.12}$$

en notación vectorial se tiene:

$$\underline{\mathbf{n}} = \underline{\mathbf{n}}^{\circ} + \mathbf{N}\underline{\xi} \tag{2.13}$$

donde $\mathbf{N}$ es la matriz estequiométrica completa y a $\underline{\xi}$ se le conoce como *vector avance de reacción*.

Por ejemplo, si el sistema $\{(CH_4, H_2O, CO_2, CO, H_2),\ (C, H, O)\}$ se considera ser el descrito por las **ecuaciones 2.9** y **2.10** con $\underline{\mathbf{n}}^{\circ} = (1.8, 6.4, 1.0, 0.4, 0.2)^{\mathrm{T}}$ y $\xi = (1.0, -0.5)^{\mathrm{T}}$. Para la primera reacción $\underline{\nu}_1 = (-1, -1, 0, 1, 3)^{\mathrm{T}}$ y para la segunda $\underline{\nu}_2 = (-1, -2, 1, 0, 4)^{\mathrm{T}}$. El número de moles en el estado E_1, de acuerdo a la ecuación anterior, será:

$$\underline{\mathbf{n}} = \begin{pmatrix} n_{CH_4} \\ n_{H_2O} \\ n_{CO_2} \\ n_{CO} \\ n_{H_2} \end{pmatrix} = \begin{pmatrix} 1.8 \\ 6.4 \\ 1.0 \\ 0.4 \\ 0.2 \end{pmatrix} + \begin{pmatrix} -1 & -1 \\ -1 & -2 \\ 0 & 1 \\ 1 & 0 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix}$$

$$= \begin{pmatrix} 1.8 \\ 6.4 \\ 1.0 \\ 0.4 \\ 0.2 \end{pmatrix} + \begin{pmatrix} -0.5 \\ -0.5 \\ 0.0 \\ 1.0 \\ 1.0 \end{pmatrix}$$

$$\begin{pmatrix} n_{CH_4} \\ n_{H_2O} \\ n_{CO_2} \\ n_{CO} \\ n_{H_2} \end{pmatrix} = \begin{pmatrix} 1.3 \\ 6.4 \\ 0.5 \\ 1.4 \\ 1.2 \end{pmatrix}$$

2.4 Reacciones independientes en un sistema químico

La **ecuación 2.13** puede considerarse como una transformación lineal de las N variables independientes n_i a las R variables ξ_i. Las variables $\underline{\mathbf{n}}$ están restringidas por las ecuaciones de abundancia de elementos, mientras que las variables $\underline{\xi}$ no están restringidas, ya que para $\underline{\xi}$, la premultiplicación de la **ecuación 2.13** por $\mathbf{A}$ da como resultado:

$$\mathbf{A}\underline{\mathbf{n}} = \mathbf{A}\underline{\mathbf{n}}^{\circ} + \mathbf{A}\mathbf{N}\underline{\xi} \tag{2.14}$$

el primer término de la derecha es $\underline{\mathbf{b}}$ y el segundo término desaparece a causa de la definición de los vectores estequiométricos. Por ejemplo, para el sistema presentado en la **sección 2.3** se tiene:

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 4 & 2 & 0 & 0 & 2 \\ 0 & 1 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1.3 \\ 6.4 \\ 0.5 \\ 1.4 \\ 1.2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 4 & 2 & 0 & 0 & 2 \\ 0 & 1 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1.8 \\ 6.4 \\ 1.0 \\ 0.4 \\ 0.2 \end{pmatrix}$$

$$+ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 4 & 2 & 0 & 0 & 2 \\ 0 & 1 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} -1 & -1 \\ -1 & -2 \\ 0 & 1 \\ 1 & 0 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix}$$

$$\begin{pmatrix} 3.2 \\ 20.4 \\ 8.0 \end{pmatrix} = \begin{pmatrix} 3.2 \\ 20.4 \\ 8.0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1.0 \\ -0.5 \end{pmatrix}$$

$$\begin{pmatrix} 3.2 \\ 20.4 \\ 8.0 \end{pmatrix} = \begin{pmatrix} 3.2 \\ 20.4 \\ 8.0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 3.2 \\ 20.4 \\ 8.0 \end{pmatrix} = \begin{pmatrix} 3.2 \\ 20.4 \\ 8.0 \end{pmatrix}$$

El significado químico de la **ecuación 2.13** es que cualquier estado composicional del sistema $\underline{\mathbf{n}}$ puede representarse en términos de cualquier estado particular $\underline{\mathbf{n}}^\circ$ y de una combinación lineal de un conjunto de R vectores linealmente independientes $\{\nu_j\}$ que satisfacen la **ecuación 2.4**.

Las **ecuaciones 2.13** y **2.14** conducen naturalmente al concepto de *ecuaciones químicas*. Lo que se llama ecuación química es simplemente una forma de taquigrafía química para representar la ecuación:

$$\mathbf{AN} = \mathbf{0} \tag{2.15}$$

en donde se substituyen las columnas de $\mathbf{A}$ por las fórmulas moleculares correspondientes de las especies. Por ejemplo, para sistema $\{(CO, CH_4, H_2, H_2O, CO_2, C_2H_6),\ \ (C, H, O)\}$, que posee una matriz fórmula, $\mathbf{A}$:

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 1 & 2 & 0 \\ 0 & 4 & 2 & 2 & 0 & 6 \end{pmatrix}$$

se tiene una matriz estequiométrica completa dada por:

$$\mathbf{N} = \begin{pmatrix} -1 & -2 & 0 \\ 1 & 1 & -2 \\ -3 & -2 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

entonces las ecuaciones químicas del sistema se obtienen sustituyendo los vectores de la fórmula por los nombres de las especies respectivas:

$$\begin{aligned} -1CO + 1CH_4 - 3H_2 + 1H_2O + 0CO_2 + 0C_2H_6 &= 0 \\ -2CO + 1CH_4 - 2H_2 + 0H_2O + 1CO_2 + 0C_2H_6 &= 0 \\ 0CO - 2CH_4 + 1H_2 + 0H_2O + 0CO_2 + 1C_2H_6 &= 0 \end{aligned}$$

En forma convencional los nombres de las especies con coeficientes negativos se escriben al lado izquierdo de la ecuación química y los que tienen coeficientes positivos se escriben del lado derecho, de manera que no aparezcan números negativos. Así, eliminando las cantidades que son cero y reordenando de acuerdo a esta convención, se tiene:

$$\begin{aligned} CO + 3H_2 &\rightleftarrows CH_4 + H_2O \\ 2CO + 2H_2 &\rightleftarrows CH_4 + CO_2 \\ 2CH_4 &\rightleftarrows H_2 + C_2H_6 \end{aligned}$$

A partir de las ecuaciones anteriores es posible determinar «cualquier» solución posible $\underline{\mathbf{n}}$ de las ecuaciones de abundancia de elementos especificando, de algún modo distinto a la estequiometría química, un conjunto apropiado de R valores ξ_j, relativos a un $\underline{\mathbf{n}}^\circ$ adecuado, junto con la matriz $\mathbf{A}$.

2.5 Procedimiento estequiométrico

La forma de determinar el conjunto de ecuaciones químicas linealmente independientes, y por extensión el rango, del sistema implica la reducción de la matriz fórmula, $\mathbf{A}$, a la forma de matriz unitaria[5]. La matriz $\mathbf{A}$ tratada de esta forma, $\mathbf{A}^{\text{uni}}$, tendrá la forma:

$$\mathbf{A}^{\text{uni}} = \begin{pmatrix} \mathbf{I}_r & \mathbf{Z}_{r\times gle} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{2.16}$$

donde $\mathbf{I}_r$ es la matriz identidad y $\mathbf{Z}_{r\times gle}$ es una matriz donde al menos uno de sus elementos es distinto de cero. r es el rango de la matiz $\mathbf{A}$ y gle[6] es el número de ecuaciones químicas linealmente independientes.

Una matiz estequiométrica completa se forma a partir de la matriz $\mathbf{A}^{\text{uni}}$ anexándole la matriz identidad $\mathbf{I}_{gle}$, de la siguiente forma:

$$\mathbf{N} = \begin{pmatrix} -\mathbf{Z} \\ \mathbf{I}_r \end{pmatrix} \tag{2.17}$$

La forma de la **ecuación 2.17** se denomina *canónica* o *base racional* del espacio nulo. Existen varios métodos para obtener la matriz $\mathbf{N}$ dos de ellos son la descomposición SVD y el proceso iterativo de ortonormalización de Gram–Schmidt.[3] Estos dos últimos métodos no se describen aquí, pues los algoritmos estequiométricos para el cálculo del equilibrio químico emplean la forma canónica de la matriz estequiométrica completa, $\mathbf{N}$. Para ilustrar el procedimiento estequiométrico para la obtención de la matriz $\mathbf{N}$, considérese el siguiente sistema:

$$\{(\mathrm{CO}, \mathrm{H_2}, \mathrm{FeO}, \mathrm{H_2O}, \mathrm{CO_2}, \mathrm{Fe}),\ \ (\mathrm{C}, \mathrm{H}, \mathrm{Fe}, \mathrm{O})\}$$

para el cual se tienen

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 2 & 0 \\ 0 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

[5] El procedimiento es similar al empleado en por el método Gauss–Jordan para la solución de ecuaciones algebraicas lineales.

[6] Grados de libertad estequiométricos.

y

$$\mathbf{A}^{\text{uni}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & -1 \end{pmatrix}$$

de la **ecuación 2.16** puede verse que:

$$-\mathbf{Z} = \begin{pmatrix} -1 & 0 \\ 1 & -1 \\ 0 & -1 \\ -1 & 1 \end{pmatrix}$$

con $gle = 2$ entonces

$$\mathbf{I}_{gle} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

de la **ecuación 2.17** para este sistema

$$\mathbf{N} = \begin{pmatrix} -1 & 0 \\ 1 & -1 \\ 0 & -1 \\ -1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

3 Principios termodinámicos

Todo cálculo de equilibrio químico se reduce a la resolución de ecuaciones derivadas de la termodinámica clásica. En esta sección se resumen los principios termodinámicos necesarios para el planteamiento del cálculo de equilibrio químico.

3.1 Primera ley de la termodinámica

El enunciado que contiene el principio de la primera ley de la termodinámica es:

> En cualquier proceso la energía se conserva

La translación matemática de este principio es:

$$dE = \delta q + \delta w \tag{3.1}$$

donde E, q y w son, respectivamente, la energía, el calor y el trabajo *absorbido o cedido por el sistema.*

3.1.1 Capacidad calorífica y entalpía

Cuando una substancia absorbe calor su temperatura generalmente se incrementa, y la relación entre calor absorbido dq y el incremento de temperatura dT es de considerable importancia. Esta relación es llamada *capacidad calorífica* y es designada con el símbolo C:

$$C = \frac{\delta q}{dT} \tag{3.2}$$

debido a que δq no es una diferencial exacta, C depende la trayectoria de transformación. Las dos clases de transformación más importantes que toman lugar son a volumen constante y presión constante, simbolizadas por C_V, C_P, respectivamente. Puede ser calculada experimentalmente una relación entre estas dos cantidades.

$$C_P - C_V = \frac{\alpha^2 VT}{\beta} \tag{3.3}$$

donde α y β son los coeficientes de expansión y compresibilidad. V y T son el volumen y la temperatura. α y β están definidos como:

$$\alpha = \frac{1}{V}\left.\frac{\partial V}{\partial T}\right|_P \tag{3.4}$$

$$\beta = -\frac{1}{V}\left.\frac{\partial V}{\partial P}\right|_T \tag{3.5}$$

Si el trabajo de un sistema es hecho sólo contra una presión externa, entonces el calor absorbido por el sistema a volumen constante es idéntico al incremento de la energía. Esto es

$$dE = \delta q - PdV \tag{3.6}$$

y

$$dE|_V = \left.\delta q\right|_V \tag{3.7}$$

entonces

$$C_V = \frac{\delta E}{\delta T} \tag{3.8}$$

A presión constante es conveniente introducir la función *entalpía* H definida como:

$$H \equiv E + PV \tag{3.9}$$

Entonces

$$dH = d(E + PV) = \delta q + \delta w + VdP + PdV \tag{3.10}$$

y cuando $\delta w = -PdV$,

$$dH = \delta q + VdP \tag{3.11}$$

entonces

$$dH|_P = \left.\delta q\right|_P \tag{3.12}$$

en consecuencia

$$C_P = \left.\frac{\partial H}{\partial T}\right|_P \tag{3.13}$$

La dependencia del C_P de una substancia con la temperatura en ocasiones puede ser expresada como:

$$C_P = a + bT - cT^{-2} + dT^2 + eT^3 + fT^{-1/2} + gT^{-3} \tag{3.14}$$

donde a, b, c, d, e, f y g son parámetros específicos de cada substancia. Se obtiene un expresión para H a presión constante de la **ecuación 3.12**.

$$H|_P = \int C_P \, dT \tag{3.15}$$

Por otra parte, la energía de un sistema conteniendo n_1 moles del componente A_1, n_2 moles del componente A_2, ..., y n_r moles del componente A_r es

$$H = \sum_{i=1}^{r} n_i H_i \tag{3.16}$$

a temperatura y presión constante, el incremento de la entalpía debido al progreso de la reacción es

$$H = \sum_{i=1}^{r} H_i dn_i \tag{3.17}$$

3.2 Segunda ley de la termodinámica

La segunda ley de la termodinámica puede ser establecida en una gran variedad de maneras, pero puede describirse a partir de alguna de sus consecuencias más importantes: *la segunda ley de la termodinámica es capaz de predecir la dirección en la cuál se llevará acabo una reacción.*

La segunda ley de la termodinámica define una función de estado llamada *entropía* S la cuál para procesos reversibles se define como:

$$dS = \frac{\delta q_{\text{reversible}}}{T} \tag{3.18}$$

para procesos irreversibles:

$$dS > \frac{\delta q_{\text{irreversible}}}{T} \tag{3.19}$$

y de aquí puede enunciarse la segunda ley de la termodinámica de la siguiente forma:

> No existe algún proceso en el que la suma de las entropías de un sistema y sus alrededores disminuya

. De las **ecuaciones 3.12** y **3.15** se tiene que para un proceso isobárico reversible:

$$S = \int \frac{C_P}{T}\, dT \tag{3.20}$$

3.3 Energía libre de Gibbs

Para que un proceso sea espontáneo debe aumentar la entropía del universo: $\Delta S_{\text{universo}} > 0$. El cambio de la entropía del universo es la suma de cambio de la entropía del sistema más el cambio de la entropía de medio ambiente local:

$$\Delta S_{\text{universo}} = \Delta S_{\text{sistema}} + \Delta S_{\text{ambiente}} \tag{3.21}$$

Cuando la temperatura y la presión se mantienen constantes y se hable de un proceso reversible, es decir, $\Delta S_{\text{universo}} = 0$, se tiene:

$$\Delta S_{\text{ambiente}} = -\frac{\delta q_{\text{sistema}}}{T} = -\frac{\Delta H_{\text{sistema}}}{T} \tag{3.22}$$

combinando la **ecuación 3.21** y la **3.22** se obtiene:

$$-T\Delta S_{\text{universo}} = \Delta H_{\text{sistema}} - T\Delta S_{\text{sistema}} \tag{3.23}$$

ahora bien, si se hace $\Delta G = -T\Delta S_{\text{universo}}$ y a ΔG se le llama *energía libre de Gibbs*, para un sistema se tiene que $\Delta G = \Delta H_{\text{sistema}} - T\Delta S_{\text{sistema}}$ o de forma más simple:

$$\Delta G = \Delta H - T\Delta S \tag{3.24}$$

considerando que un proceso es espontáneo si $\Delta S_{\text{universo}}$ es positivo, la energía libre de Gibbs puede utilizarse como un criterio de espontaneidad, esto es:

> $\Delta G < 0$ el proceso es espontáneo o factible
> $\Delta G > 0$ el proceso no es espontáneo o factible
> $\Delta G = 0$ el sistema está en equilibrio

3.4 Potencial químico

Cambios infinitesimales en equilibrio químico conduce al correspondiente cambio en la energía libre. La energía libre es una función de todas las variables independientes del sistema. Si se considera $\underline{\mathbf{n}}$ como el número de moles de las especies, $G = G(P, T, \underline{\mathbf{n}})$ y

$$dG = \left.\frac{\partial G}{\partial T}\right|_{P,\underline{\mathbf{n}}} dT + \left.\frac{\partial G}{\partial P}\right|_{T,\underline{\mathbf{n}}} dP + \sum_{i}^{j} \left.\frac{\partial G}{\partial n_i}\right|_{T,P,n_j} dn_i \tag{3.25}$$

y puede demostrase que

$$\left.\frac{\partial G}{\partial T}\right|_{P,\underline{\mathbf{n}}} = -S \qquad \text{y} \qquad \left.\frac{\partial G}{\partial P}\right|_{T,\underline{\mathbf{n}}} = V \tag{3.26}$$

Así que

$$dG = -S\,dT + V dP + \sum_{i}^{j} \left.\frac{\partial G}{\partial n_i}\right|_{T,P,n_j} dn_i \tag{3.27}$$

Las diferenciales parciales $\partial G/\partial n_i|_{T,P,n_j}$son propiedades intensivas del sistema, e indican el efecto en la energía libre de Gibbs del sistema cuando una pequeña cantidad de especies i es adicionada al sistema. La derivada es usualmente llamada *potencial químico* (μ_i) o *energía libre molar parcial* (g_i) de la i-ésima especie; ésta es una función de P, T y de $\underline{\mathbf{n}}$, así esta última ecuación puede reescribirse como:

$$dG = -S\,dT + V dP + \sum_{i}^{j} g_i\, dn_i \tag{3.28}$$

Es útil relacionar los potenciales químicos al potencial químico estándar (a presión de 1 atm), de una manera similar a otras funciones termodinámicas. Por ejemplo, cuando se integra (a temperatura constante y n_i) desde 1 atm hasta P_i, la **ecuación 3.28**:

$$G_T - G_T^\circ = \int_1^{P_i} V\, dP \tag{3.29}$$

donde el símbolo «°» denota la condición estándar. Entonces usando la ecuación de los gases ideales en la forma:

$$V = \frac{n_i RT}{P_i} \tag{3.30}$$

conduce a:

$$G_T - G_T^\circ = n_i RT \ln P_i \tag{3.31}$$

y consecuentemente diferenciando con respecto a n_i se obtiene:

$$g_i = \mu_i = g_i^\circ + RT \ln P_i \tag{3.32}$$

y dado que $G = \sum_i g_i n_i$:

$$G = \sum_i n_i (g_i^\circ + RT \ln P_i) \tag{3.33}$$

3.5 Minimización de la energía libre de Gibbs

Para calcular una composición en equilibrio, es decir, el arreglo de moles no negativos que dan el valor más pequeño posible de la energía libre total del sistema, y que satisfacen las restricciones de balance de masa, se usa un procedimiento iterativo. Primero, el número de moles de las substancias consideradas (y°) son estimadas. Los valores mejorados de los números de moles pueden ser calculados (x), que en turno son usados para los nuevos valores sugeridos (y), y así hasta que la composición en equilibrio es alcanzada. Así, en cada iteración se comienza con un nuevo arreglo de y-valores.

La energía libre del sistema puede ser expresada como

$$G = \sum_i n_i g_i$$

donde n_i denota el número de moles de la especie i y g_i es su potencial químico definido como

$$g_i = g_i^\circ + RT \ln a_i$$

Para especies gaseosas, que son tratadas como ideales, las actividades a_i son iguales a la presión parcial P_i

$$a_i = P_i = x_i P$$

x_i y P denotan la fracción molar de la especie i en de la fase gaseosa y la presión total, respectivamente. Para las substancias condensadas, que se consideran como puras, las actividades son igual a la unidad. Usando las definiciones anteriores, una cantidad no dimensional puede ser usada (G/RT) obteniéndose

$$\frac{G}{RT} = \sum_{i=1}^{m} n_i^g \left[\frac{g_i^{\circ,g}}{RT} + \ln P + \ln x_i^g \right] + \sum_{i=1}^{s} n_i^c \frac{g_i^{\circ,c}}{RT} \tag{3.34}$$

Los índices g y c indican las fases gaseosas y condensadas, respectivamente. El número de substancias en la fase gaseosa es denotado por m y s es el número de substancias en la fase condensada presentes en equilibrio.

Las relaciones de balance de masa pueden escribirse como:

$$\sum_{i=1}^{m} a_{i,j}^g n_i^g + \sum_{i=1}^{m} a_{i,j}^c n_i^c = b_j \quad \text{con} \quad j = 1, 2, \ldots, l \tag{3.35}$$

donde $a_{i,j}$ representa el número de átomos del j-ésimo elemento en la molécula de la i-ésima substancia, b_j es el número total de moles del j-ésimo elemento, l es el número total de elementos.

3.6 Formulaciones de las condiciones de equilibrio

Para que un sistema de una fase o de fases múltiples se encuentre en equilibrio, G está en un mínimo global sujeto a las restricciones del sistema cerrado y a la restricción de no

negatividad a las condiciones termodinámicas establecidas, T y P fijos. En el equilibrio se tiene que:

$$dG\Big|_{T,P} = 0 \tag{3.36}$$

aunque ésta es una condición necesaria, no es suficiente. El problema fundamental es expresar G en función de n_i y buscar los valores de n_i que hagan G un mínimo sujeto a las restricciones. Se supone que se conocen los valores del vector de abundancia de elementos, $(\underline{b})$, de la temperatura T, de la presión P y los datos apropiados de la energía libre. Aquí se describen dos formulaciones del problema de minimización, que se refieren como:

1. formulación estequiométrica en donde las restricciones de un sistema cerrado se tratan por medio de ecuaciones estequiométricas que dan como resultado esencialmente un problema de minimización sin restricciones, y
2. la formulación no estequiométrica en donde las ecuaciones estequiométricas no se utilizan, sino en su lugar la restricción de un sistema cerrado se trata por medio de multiplicadores de Lagrange.

Estas dos formulaciones se describen en las dos secciones siguientes.

3.7 Formulación estequiométrica

Se parte de la relación entre el número de moles, $\underline{\mathbf{n}}$, y los grados de conversión de la reacción $\underline{\xi}$ de las R ecuaciones estequiométricas linealmente independientes

$$\underline{\mathbf{n}} = \underline{\mathbf{n}}^{\circ} + \mathbf{N}^{\mathrm{T}}\underline{\xi} \tag{3.37}$$

o bien

$$\underline{\mathbf{n}} = \underline{\mathbf{n}}^{\circ} + \sum_{j=1}^{R} \underline{\nu}_j^{\mathrm{T}} \xi_j \tag{3.38}$$

El problema consiste en minimizar $G(T, P, \underline{\xi})$ para valores fijos de T y P, en términos de las R ξ_j. Puesto que estas últimas cantidades son independientes, las condiciones necesarias de primer orden para un mínimo en G son:

$$\left.\frac{\partial G}{\partial \underline{\xi}}\right|_{T,P} = \underline{\mathbf{0}} \tag{3.39}$$

o

$$\left.\frac{\partial G}{\partial \xi_j}\right|_{T,P,\xi_{k\neq j}} = 0 \qquad \text{para} \qquad j = 1,2,\ldots R \tag{3.40}$$

Hay $R = N - r$ ecuaciones en el conjunto de **ecuaciones 3.40**. Puesto que

$$\left.\frac{\partial G}{\partial \xi}\right|_{T,P,\xi_{k\neq j}} = \sum_{i=1}^{N} \left.\frac{\partial G}{\partial n_i}\right|_{T,P,\xi_{k\neq i}} \left.\frac{\partial n_i}{\partial \xi_j}\right|_{T,P,\xi_{k\neq j}} \qquad j = 1,2,\ldots,R \tag{3.41}$$

donde

$$\left.\frac{\partial G}{\partial n_i}\right|_{T,P,\xi_{k\neq i}} = g_i \tag{3.42}$$

y

$$\left.\frac{\partial n_i}{\partial \xi_j}\right|_{T,P,\xi_{k\neq j}} = \nu_{ij} \tag{3.43}$$

entonces se tiene la condición de equilibrio

$$\sum_{i=1}^{N} \nu_{i,j} g_i = 0 \qquad \text{para} \qquad j = 1,2,\ldots R \tag{3.44}$$

o bien

$$\underline{\nu}^{\mathrm{T}} \underline{\mathbf{g}} = \underline{\mathbf{0}} \tag{3.45}$$

La cantidad a la izquierda de la **ecuación 3.44** se representa por ΔG_j y su valor negativo se le llama afinidad.

3.8 Formulación no estequiométrica

El problema se formula como una minimización de G, para T y P fijas, en términos de los N números de moles, sujetos a las M restricciones de abundancia de elementos, esto es

$$\underline{\mathbf{n}}\underline{\mathbf{g}} = \text{mín!} \tag{3.46}$$

sujeta a

$$\mathbf{A}\underline{\mathbf{n}} = \underline{\mathbf{b}} \tag{3.47}$$

esta es una forma simple del problema de optimización restringido. Un procedimiento consiste en utilizar el método de multiplicadores de Lagrange para eliminar las restricciones. Para esto primero se escribe el lagrangiano, L,

$$L(\underline{\mathbf{n}}, \underline{\lambda}) = \sum_{i=1}^{N} n_i g_i + \sum_{k=1}^{M} \lambda_k \left(b_k - \sum_{i=1}^{N} a_{ki} n_i \right) \tag{3.48}$$

donde $\underline{\lambda}$ es un vector de M multiplicadores de Lagrange desconocidos. Entonces las condiciones necesarias proporcionan el siguiente conjunto de $(N + M)$ ecuaciones en las $(N + M)$ incógnitas $(n_1, n_2, \ldots, n_N, \lambda_1, \lambda_2, \ldots, \lambda_M)$:

$$\left.\frac{\partial L}{n_i}\right|_{n_j \neq i, \underline{\lambda}} = g_i - \sum_{k=1}^{M} a_{ki}\lambda_k = 0, \qquad (n_i > 0) \tag{3.49}$$

$$\left.\frac{\partial L}{\lambda_k}\right|_{\underline{\mathbf{n}}, \lambda_j \neq k} = b_k - \sum_{i=1}^{N} a_{ki} n_i = 0 \tag{3.50}$$

Como en la formulación estequiométrica, la solución de estas ecuaciones implica la introducción de una expresión apropiada para g_i.

3.9 Actividad

La actividad química de la especie i, a_i, es la medida de la facilidad que tiene la especie i para reaccionar con otras especies.

3.10 Solución ideal

Una solución ideal se considera que la actividad de la especie i es igual a su fracción molar:

$$a_i = x_i \tag{3.51}$$

Así el potencial químico para una especie en una solución ideal, por ejemplo,

$$g_i(T,P,x_i) = g_i^{\circ}(T,P) + RT\ln x_i \tag{3.52}$$

depende sólo de su propia concentración.

3.11 Solución no ideal

El potencial químico para una especie en una solución no ideal representado, por ejemplo, por

$$g_i(T,P,\underline{\mathbf{x}}) = g_i^{\circ}(T,P) + RT\ln \gamma_i(T,P,\underline{\mathbf{x}})x_i \tag{3.53}$$

depende de la composición en general, según se refleja en el coeficiente de actividad γ_i

3.12 Ley de Raoult y Ley de Henrry

La actividad de la especie i puede relacionarse con su fracción molar mediante un parámetro adimensional denominado coeficiente de actividad γ_i

$$a_i = \gamma_i x_i \tag{3.54}$$

Cuando una especie en una solución obedece la *ley de Raoult* su actividad y su fracción mol son iguales, esto es:

$$a_i = x_i \qquad \text{en consecuencia} \qquad \gamma_i = 1 \tag{3.55}$$

En los sistemas reales la ley de Raoult se cumple cuando x_i tiende a la unidad, es decir, en soluciones muy concentradas. En soluciones muy diluidas se cumple que el coeficiente de actividad permanece constante, esto es: cuando x_i tiende a cero γ_i tiende a ser constante, a esto se le llama *ley de Henry*. En tal caso el coeficiente de actividad se escribe γ_i°. En la **figura 3.1** se muestra el ejemplo de una solución líquida de Cd-Pb a 500°C donde se representa la variación de la actividad en función de la fracción mol del cadmio.

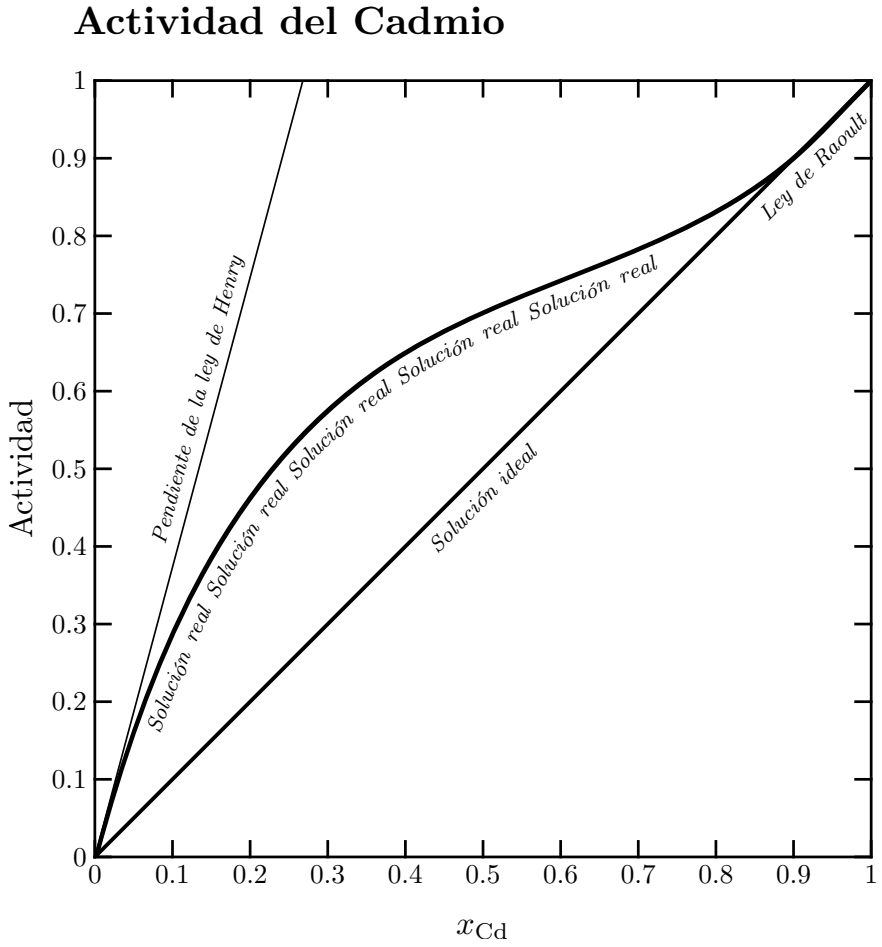

Figura 3.1 Curva esquemática de la actividad del cadmio en el sistema líquido Cd-Pb a 500 °C.

3.13 Modelo de parámetros de interacción

El modelo de «parámetros de interacción» o «coeficientes de interacción» puede emplearse en sistemas de soluciones diluidas, es decir, en los que la cantidad del solvente es mucho mayor a la cantidad de todos los solutos juntos. Una solución diluida puede ser, por ejemplo, un acero al carbono líquido.

Para explicar la forma en que se desarrolla el modelo de parámetros de interacción, considérese el sistema formado, en principio, por dos componentes, i y j. Donde el soluto j se

encuentra infinitamente diluido. El coeficiente de actividad para j en cualquier fracción mol x_j se designa por γ_{jj}, donde el segundo subíndice indica que la solución sólo contiene un soluto. Ahora, suponiendo que una pequeña cantidad de un tercer elemento t se adiciona a la solución binaria mientras se mantiene la actividad de j constante, en ocasiones esta última condición es satisfecha si la concentración de j se mantiene constante. El nuevo valor del coeficiente de actividad para j en la solución ternaria se designa por γ_j. Estos dos valores son relacionados como:

$$\gamma_j = \gamma_{jj}\gamma_{jt} \tag{3.56}$$

o

$$\ln \gamma_j = \ln \gamma_{jj} + \ln \gamma_{jt} \tag{3.57}$$

la cantidad γ_{jt} puede entenderse como el efecto del elemento t sobre el coeficiente de actividad del soluto j. Si el tercer elemento tiene un efecto finito en el coeficiente de actividad del soluto j, entonces $\gamma_{jt} \neq 1$. Así, el factor γ_{jt} es la medida de la interacción entre el tercer elemento y la solución primaria.

Lo anterior puede generalizarse para incluir una solución multicomponente que contenga al solvente 1 y los elementos solutos 1, 2, 3, ... en las fracciones molares x_1, x_2, x_3, ... en la cual el coeficiente de actividad del soluto 2 se puede expresar como:

$$\ln \gamma_2 = \ln \gamma_{22} + \ln \gamma_{23} + \ln \gamma_{24} + \cdots \tag{3.58}$$

En esta expresión, γ_{22} es el coeficiente de actividad del soluto 2 en la solución binaria 1-2 con la fracción mol de x_2. γ_{23}, γ_{24}, ... son los respectivos efectos de los elementos 3, 4, ... en el coeficiente de actividad del componente 2. Se ha desarrollado un método para tratar estas interacciones desarrollándolas en una expansión de Taylor para γ_2 dando:

$$\begin{aligned} \ln \gamma_2 = \ln \gamma_2^\circ + \left[x_2 \left(\frac{\partial \ln \gamma_2}{\partial x_2} \right) + x_3 \left(\frac{\partial \ln \gamma_2}{\partial x_3} \right) + x_4 \left(\frac{\partial \ln \gamma_2}{\partial x_4} \right) + \cdots \right] \\ + \left[\frac{1}{2} x_2 x_2 \left(\frac{\partial^2 \ln \gamma_2}{\partial x_2^2} \right) + x_2 x_3 \left(\frac{\partial^2 \ln \gamma_2}{\partial x_2 \partial x_3} \right) + \cdots \right] \end{aligned} \tag{3.59}$$

donde las derivadas parciales se toman para el caso en que todos los solutos tienden a la concentración cero. Los términos:

$$\epsilon_{22} = \left.\frac{\partial \ln \gamma_2}{\partial x_2}\right|_{x_2 \to 0}, \quad \epsilon_{23} = \left.\frac{\partial \ln \gamma_2}{\partial x_3}\right|_{x_3 \to 0}, \quad \epsilon_{24} = \left.\frac{\partial \ln \gamma_2}{\partial x_3}\right|_{x_4 \to 0}, \quad \ldots \tag{3.60}$$

se conocen como «parámetros de interacción». Haciendo algunas otras consideraciones como que el parámetro de interacción del soluto i sobre el soluto j es igual al parámetro de interacción del soluto j sobre el soluto i:

$$\epsilon_{ij} = \epsilon_{ji} \tag{3.61}$$

y sustituyendo estas expresiones en la **ecuación 3.59** se tiene una expresión para los solutos de la forma:

$$\ln \gamma_i = \ln \gamma_i^\circ + \ln \gamma_1 + \sum_{j=2}^{m} \epsilon_{ij} x_j \qquad \text{para} \qquad i = 2, 3, 4, \ldots \tag{3.62}$$

donde

$$\ln \gamma_1 = -\frac{1}{2} \sum_{i=2}^{m} \sum_{j=2}^{m} \epsilon_{ij} x_i x_j \tag{3.63}$$

que es la expresión para el solvente.

3.13.1 Ejemplo de cálculo

El objetivo que se persigue al realizar un cálculo con el modelo de parámetros de interacción para una solución es: determinar la actividad de cada especie participante. En el siguiente ejemplo se describe el cálculo de la actividad química de una aleación a la temperatura de 1873 K y cuya composición se muestra en la **tabla 3.1**.

Tabla 3.1 Composición de una aleación férrica a 1873 K.

Especie	**Fracción mol,** x
Fe	0.95797
Al	0.00978
C	0.01955
Ca	0.00489
Si	0.00782

De la **ecuación 3.62** se obtiene para los solutos:

$$\ln\gamma_{Al} = \ln\gamma^{\circ}_{Al} + \ln\gamma_{Fe} + \left[\epsilon_{AlAl}x_{Al} + \epsilon_{AlC}x_{C} + \epsilon_{AlCa}x_{Ca} + \epsilon_{AlSi}x_{Si}\right] \tag{3.64}$$

$$\ln\gamma_{C} = \ln\gamma^{\circ}_{C} + \ln\gamma_{Fe} + \left[\epsilon_{CC}x_{C} + \epsilon_{CAl}x_{Al} + \epsilon_{CCa}x_{Ca} + \epsilon_{CSi}x_{Si}\right] \tag{3.65}$$

$$\ln\gamma_{Ca} = \ln\gamma^{\circ}_{Ca} + \ln\gamma_{Fe} + \left[\epsilon_{CaCa}x_{Ca} + \epsilon_{CaAl}x_{Al} + \epsilon_{CaC}x_{C} + \epsilon_{CaSi}x_{Si}\right] \tag{3.66}$$

$$\ln\gamma_{Si} = \ln\gamma^{\circ}_{Si} + \ln\gamma_{Fe} + \left[\epsilon_{SiSi}x_{Si} + \epsilon_{SiAl}x_{Al} + \epsilon_{SiC}x_{C} + \epsilon_{SiCa}x_{Ca}\right] \tag{3.67}$$

y de la **ecuación 3.63** para el solvente:

$$\begin{aligned}\ln\gamma_{Fe} = -\frac{1}{2}\Big[&\epsilon_{AlAl}x^2_{Al} + \epsilon_{CC}x^2_{C} + \epsilon_{SiSi}x^2_{Si} \\ &+2\epsilon_{AlC}x_{Al}x_{C} + 2\epsilon_{AlCa}x_{Al}x_{Ca} + 2\epsilon_{AlSi}x_{Al}x_{Si} \\ &+2\epsilon_{CaC}x_{Ca}x_{C} + 2\epsilon_{CSi}x_{C}x_{Si} + 2\epsilon_{CaSi}x_{Ca}x_{Si}\Big]\end{aligned} \tag{3.68}$$

En la segunda columna de la **tabla 3.2** se muestran las expresiones termodinámicas que contienen a $\ln\gamma^{\circ}_{i}$ con i = Al, C, Ca, Si. Empleando estas ecuaciones se calcula el valor de $\ln\gamma^{\circ}_{i}$ para cada especie y el resultado se muestra en la tercera columna de la misma tabla.

Tabla 3.2 Datos termodinámicos para el sistema Fe, Al, C, Ca, Si. Parte 1.

Especie, i	**Expresión de** $\ln\gamma^{\circ}_{i}$	$\ln\gamma^{\circ}_{i}$ a 1873 K
Al	$RT\ln\gamma^{\circ}_{Al} = -71097.93 + 12.8861T$	-3.01580
C	$RT\ln\gamma^{\circ}_{C} = +17234.09 - 14.3576T$	-0.62020
Ca	$RT\ln\gamma^{\circ}_{Ca} = -39398.16 + 84.8902T$	7.68047
Si	$RT\ln\gamma^{\circ}_{Si} = -131496.21 + 15.3303T$	-6.60040

Aquí, los parámetros a y b del coeficiente de actividad henriano, γ°_{i}, expresado como:

$$RT\ln\gamma^{\circ}_{i} = a + bT. \tag{3.69}$$

se agrupan con los parámetros A y B de la expresión de energía libre estándar de los elementos puros:

$$g_i^\circ = A + BT + CT\ln T + DT^2 + ET^{-3} + FT^{-1} + GT^4 + HT^5 + IT^{-9} + JT^{-7} + K\ln T + LT^{-2} + MT^{1/2} \tag{3.70}$$

De la misma forma se tienen las expresiones que contienen a ϵ_{ij} con $i = j =$ Al, C, Ca, Si se muestran en la **tabla 3.3**, en la tercera columna se encuentra el cálculo de ϵ_{ij} a la temperatura de 1873 K.

Tabla 3.3 Datos termodinámicos para el sistema Fe, Al, C, Ca, Si. Parte 2.

Especies ij	**Expresión de** ϵ_{ij}	ϵ_{ij} a 1873 K
AlAl	$RT\epsilon_{\mathrm{AlAl}} = -58577.63 + 28.5988T$	7.2015
AlC	$RT\epsilon_{\mathrm{AlC}} = 77067.07$	4.9490
AlCa	$RT\epsilon_{\mathrm{AlCa}} = -117222$	−7.5277
AlSi	$RT\epsilon_{\mathrm{AlSi}} = 108999.6$	6.9997
CC	$RT\epsilon_{\mathrm{CC}} = 199310.2$	12.7992
CCa	$RT\epsilon_{\mathrm{CCa}} = -246024$	−15.7990
CSi	$RT\epsilon_{\mathrm{CSi}} = 155796.9 - 3.5582T$	9.5769
CaCa	$RT\epsilon_{\mathrm{CaCa}} = 0.0$	0.0000
CaSi	$RT\epsilon_{\mathrm{CaSi}} = -167236$	−10.7395
SiSi	$RT\epsilon_{\mathrm{SiSi}} = 33470.55 + 85.2144T$	12.3989

Empleando las **ecuaciones 3.64–3.67** y las **tablas 3.2** y **3.3** se construye la **tabla 3.4**, en ella se muestran los resultados de todos los cálculos para obtener la actividad de cada especie.

Tabla 3.4 Resultados del cálculo de actividades del sistema Fe, Al, C, Ca, Si. a 1873K.

i	x_i	$\ln\gamma_i^\circ$	$\ln\gamma_i$	γ_i	$a_i = x_i\gamma_{\mathrm{Fe}}$
Fe	0.95797		−0.00383	0.99617	0.95430
Al	0.00978	−3.01580	−2.83453	0.05875	5.74252×10^{-5}
C	0.01955	−0.62020	−0.32775	0.72054	0.0140869
Ca	0.00489	7.68047	7.21019	1353.14833	6.61363
Si	0.00782	−6.60040	−6.30413	0.00183	1.43009×10^{-5}

4 Programación Matemática

Los procedimientos que se tratan para resolver el problema del equilibrio químico implica la consideración de dos clases de problemas numéricos. Estos son: 1) la minimización de una función $f(\underline{\mathbf{x}})$ quizá sujeta a ciertas restricciones y 2) la solución de un conjunto de ecuaciones algebraicas no lineales. En esta sección se desarrollan los conceptos básicos de los métodos numéricos necesarios para el desarrollo de cualquier algoritmo para resolver el problema de equilibrio químico.

4.1 Optimización

La programación matemática es una rama de las matemáticas de la teoría de la optimización en la cual una *función objetivo* f de n variables reales $x_1, \ldots, x_n$ es minimizada (o maximizada), posiblemente sujeta a un número finito de *restricciones* que son escritas como desigualdades o igualdades.

4.2 Optimización no–lineal

Un problema de optimización no lineal tiene la forma general

$$f(\underline{\mathbf{x}}) = \text{mín!} \qquad \text{sujeto a} \qquad \underline{\mathbf{x}} \in \mathfrak{R}^n \qquad \text{con} \tag{4.1}$$

$$g_i(\underline{\mathbf{x}}) \leq 0, \qquad i \in I = \{1, \ldots m\} \qquad \text{y} \tag{4.2}$$

$$h_j(\underline{\mathbf{x}}) = 0, \qquad j \in J = \{1, \ldots n\} \tag{4.3}$$

donde una de las funciones f, g_i, o h_j es no–lineal, m es el número de restricciones escritas como desigualdades y n es el número de ecuaciones escritas como igualdades. El conjunto de las posibles soluciones es denotado por

$$M = \{\underline{\mathbf{x}} \in \mathfrak{R}^n : g_i(\underline{\mathbf{x}}) \leq 0, i \in I, h_j(\underline{\mathbf{x}}) = 0, j \in J\} \tag{4.4}$$

El problema es determinar los puntos mínimos.

4.3 Puntos mínimos

Un punto $\underline{\mathbf{x}}^*$ se llama *punto mínimo global* if $f(\underline{\mathbf{x}}^*) \leq f(\underline{\mathbf{x}})$ para cualquier $\underline{\mathbf{x}} \in M$

4.4 Problema de minimización

Un problema de minimización $f(\underline{\mathbf{x}}) = \text{mín!}$ es equivalente a un problema de maximización multiplicando la función objetivo por (-1)

$$-f(\underline{\mathbf{x}}) = \text{máx!} \tag{4.5}$$

4.5 Solución general del problema de optimización

Los problemas de minimización no–lineales no se pueden resolver en forma cerrada, excepto en los casos más simples. La mayoría de los algoritmos numéricos para resolver problemas no lineales procede por la solución de una secuencia de problemas (iteración). Los componentes de $\underline{\mathbf{x}}$ cambian de iteración en iteración en menos de cierta cantidad pequeña especificada. Así, los algoritmos deben partir de una estimación inicial $\underline{\mathbf{x}}_\circ$ y calcular una secuencia por medio de:

$$\underline{\mathbf{x}}_{n+1} = \underline{\mathbf{x}}_n + \omega_n \underline{\delta \mathbf{x}}_n \tag{4.6}$$

ω es una cantidad escalar llamada *parámetro de etapa* o *tamaño de paso*, $\underline{\delta \mathbf{x}}$ es llamado *dirección de búsqueda*. ω determina la distancia que debe avanzarse en la dirección $\underline{\delta \mathbf{x}}$ en cada iteración.

Todo algoritmo aplica la **ecuación 4.6** y

> Los algoritmos sólo difieren por la forma en que determinan $\underline{\delta \mathbf{x}}$

El parámetro de etapa se selecciona de manera que $f(\underline{\mathbf{x}}_n + \omega_n \underline{\delta \mathbf{x}}_n)$ sea más pequeño que $f(\underline{\mathbf{x}}_n)$, es decir, se requiere evaluar:

$$\text{mín!} = f(\underline{\mathbf{x}}_n + \omega_n \underline{\delta\mathbf{x}}_n) \qquad \text{con} \qquad \omega_n \geq 0 \tag{4.7}$$

4.6 Métodos de minimización

Existen muchos métodos de minimización restringidos , no restringidos, de proyección, entre otros, algunos de ellos son los siguientes.

4.6.1 No restringidos

1. Método del descenso más rápido o método de la primera variación

$$\underline{\delta\mathbf{x}} = -\left.\frac{\partial f}{\partial \underline{\delta\mathbf{x}}}\right|_{\underline{\mathbf{x}}} = -\left.\nabla f\right|_{\underline{\mathbf{x}}} \tag{4.8}$$

2. Método de la segunda variación

$$\underline{\delta\mathbf{x}} = -\left.\frac{\partial^2 f}{\partial \underline{\delta\mathbf{x}}^2}\right|_{\underline{\mathbf{x}}}^{-1} \left.\frac{\partial f}{\partial \underline{\delta\mathbf{x}}}\right|_{\underline{\mathbf{x}}} \tag{4.9}$$

4.6.2 Restringidos

1. Multiplicadores de Lagrange. Este método emplea una función auxiliar definida por

$$L(\underline{\mathbf{x}}) = f(\underline{\mathbf{x}}) - \sum_{k=1}^{m} \lambda_k (g_k(\underline{\mathbf{x}}) - b_k) \tag{4.10}$$

denominada *función lagrangiana*. Las funciones $g_k(x)$ son las restricciones del problema y los escalares b_k son los parámetros que definen esas restricciones. Las nuevas variables, λ_k, son llamadas multiplicadores de Lagrange. La función lagrangiana es, entonces, la función a minimizar por cualquier otro método.

4.6.3 Proyección

1. Método de proyección de gradiente. Se establece

$$\underline{\delta \mathbf{x}} = -\nabla f + \mathbf{A}^{\mathrm{T}} \underline{\lambda} \tag{4.11}$$

o bien

$$\underline{\delta \mathbf{x}} = -\left[\mathbf{I} - \mathbf{A}^{\mathrm{T}} \left(\mathbf{A}\mathbf{A}^{\mathrm{T}}\right)^{-1} \mathbf{A}\right] \nabla f \tag{4.12}$$

donde $\underline{\lambda}$ contiene a los multiplicadores de Lagrange.

4.7 Cálculo del tamaño de etapa

Todos los métodos de minimización y resolución de conjuntos de ecuaciones no lineales implican el cálculo de nuevos valores de $\underline{\mathbf{x}}$ a partir de otros, por medio de la **expresión 4.6**. En esta sección se estudia el cálculo de ω_n. Si es posible plantear el problema en la forma de un problema de minimización para $\underline{\mathbf{x}}$ de la forma

$$G(\underline{\mathbf{x}}) = \text{mín!} \tag{4.13}$$

La forma conveniente de seleccionar ω_n es encontrar el valor de ω que minimiza aproximadamente a $G(\underline{\mathbf{x}}_n + \omega \underline{\delta \mathbf{x}}_n)$ en cada iteración. Se ha visto que

$$G(\underline{\mathbf{x}}) = \sum_{i=1}^{N} g_i^2(\underline{\mathbf{x}}) \tag{4.14}$$

es una función cuyo mínimo determina la solución de las ecuaciones no lineales

$$\underline{\mathbf{g}}(\underline{\mathbf{x}}) = \underline{\mathbf{0}} \tag{4.15}$$

Así, la determinación de un parámetro de etapa tiene importancia general en la aplicación práctica de la mayoría de los métodos numéricos. El concepto de un método descendiente es de especial importancia, ya que para este método G es una función decreciente de ω_n en $\underline{\mathbf{x}}$; o sea, el método satisface

$$\left(\frac{dG}{d\omega_n}\right) \equiv \sum_{i=1}^{N} \left(\frac{\partial G}{\partial x_i}\right)_{\underline{\mathbf{x}}_n} \delta x_i < 0 \tag{4.16}$$

siempre que $\partial G/\partial x = 0$. la ecuación anterior asegura que se puede encontrar un valor positivo de ω_n de manera que $G(\underline{\mathbf{x}}_{n+1}) < G(\underline{\mathbf{x}}_n)$. La determinación del parámetro de etapa óptimo en cada iteración es equivalente al problema de optimización unidimensional.

$$G(\underline{\mathbf{x}}_n + \omega \underline{\delta \mathbf{x}}_n) = \text{mín!} \tag{4.17}$$

Por lo general es preferible determinar el valor sólo aproximado y después proceder con la siguiente iteración. El siguiente procedimiento se puede aplicar cuando se sabe que $\omega = 1$ para proporcionar una nueva estimación del valor óptimo. Primero, se calcula el valor de

$$\left(\frac{dG}{d\omega}\right) = \sum_{i=1}^{N} \left(\frac{\partial G}{\partial x_i}\right)_{\omega=1} \delta x_i \tag{4.18}$$

Si esta cantidad es negativa o cero, se supone que no se ha pasado el valor del minimización de ω se procede a la siguiente iteración, con $\omega_n = 1$. Si la cantidad es positiva, se establece que

$$\omega_n = \frac{dG/d\omega|_{\omega=0}}{dG/d\omega|_{\omega=0} - dG/d\omega|_{\omega=1}} \tag{4.19}$$

Esta ecuación asegura que $0 < \omega_n < 1$ ya que se supone que no se ha pasado un mínimo en $G(\omega)$ en $\omega = 1$, y $\underline{\delta \mathbf{x}}_n$ define un método descendiente.

4.8 Descomposición LU

Aunque, la descomposición LU se encuentra dentro del álgebra lineal más que dentro de la programación matemática, esta sección se incluye aquí, pues en algunos problemas de optimización no lineal es necesario resolver sistemas de ecuaciones lineales. Un método de resolución de ecuaciones lineales es la descomposición LU que se describe a continuación. Si se conoce como factorizar una matriz $\mathbf{A}_{m\times n}$, en la forma

$$\mathbf{A} = \mathbf{LU} \tag{4.20}$$

donde $\mathbf{L}$ es una matriz triangular inferior $m\times m$ y $\mathbf{U}$ es una matriz escalonada $m\times n$. Entonces el sistema

$$\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}} \tag{4.21}$$

puede resolverse en dos pasos más fáciles. Primero, se despeja $\underline{\mathbf{y}}$ de la ecuación

$$\mathbf{L}\underline{\mathbf{y}} = \underline{\mathbf{b}} \tag{4.22}$$

y a continuación se despeja $\underline{\mathbf{x}}$ de la ecuación

$$\mathbf{U}\underline{\mathbf{x}} = \underline{\mathbf{y}} \tag{4.23}$$

De hecho, la solución de estos dos sistemas equivale a resolver el sistema original, porque

$$\mathbf{LU}\underline{\mathbf{x}} = \mathbf{L}(\mathbf{U}\underline{\mathbf{x}}) = \mathbf{L}\underline{\mathbf{y}} = \underline{\mathbf{b}} \tag{4.24}$$

la ventaja de no resolverlo en forma directa es que la matriz $\mathbf{L}$ en la **ecuación 4.22** es un sistema triangular inferior que puede determinarse con facilidad mediante una sustitución directa, y la matriz $\mathbf{U}$ es la **ecuación 4.23** es triangular superior, que puede resolverse fácilmente son una solución por sustitución hacia atrás.

Una matriz $\mathbf{A}$ se puede factoriza como

$$\mathbf{A} = \mathbf{E}_1^{-1} \cdots \mathbf{E}_k^{-1}\mathbf{U} \tag{4.25}$$

donde $\mathbf{U}$ es una forma de escalón de $\mathbf{A}$, y $\mathbf{E}_1, \ldots \mathbf{E}_k$ son las matrices elementales que corresponden a las operaciones elementales de renglón usadas para reducir $\mathbf{A}$ a $\mathbf{U}$.

Cualquier matriz $\mathbf{A}$ puede reducirse a la forma de escalón sin operaciones de escalamiento, sólo con intercambios y eliminaciones. Si la matriz $\mathbf{A}$ se puede reducir empleando sólo eliminaciones, entonces las matrices $\mathbf{E}_1, \ldots \mathbf{E}_k$ son triangulares inferiores y contienen al numero uno en la diagonal principal. Si se define

$$\mathbf{L} = \mathbf{E}_1^{-1} \cdots \mathbf{E}_k^{-1} \tag{4.26}$$

entonces $\mathbf{A} = \mathbf{LU}$ es una descomposición LU de $\mathbf{A}$. No hay necesidad de calcular las inversas ni los productos de $\mathbf{L}$ ya que el (i, j)-ésimo elemento l_{ij} $(i > j)$ de $\mathbf{L}$ proviene de la operación $R_i - l_{ij}R_j \rightarrow R_i$ que se uso para obtener 0 en esta posición durante la reducción. A los elementos de $\mathbf{L}$ situados debajo de la diagonal principal se les llama multiplicadores de Gauss.

Este algoritmo es muy útil cuando hay que solucionar varios sistemas con la misma matriz $\mathbf{A}$ de coeficientes. Para obtener una idea de la utilidad de que tiene LU, se tiene que si se

necesita resolver dos sistemas con 500 ecuaciones y 500 incógnitas, y con la misma matriz **A** con la eliminación de Gauss se emplearían 160 millones de operaciones mientras que con LU sólo 83 millones.

5 Algoritmos de equilibrio químico

En esta sección se describen los algoritmos generales para tratar con problemas con cualquier número de fases, especies y elementos. Se supone que la ecuación

$$g_i = g_i^\circ(T,P) + RT\ln\frac{n_i}{n_t} \qquad (5.1)$$

es válida para cada especie. Todos los algoritmos se basan en la suposición de que existe una solución al problema y que ésta es única.

5.1 Clasificación de los algoritmos

Los algoritmos desarrollados para tratar el problema de equilibrio químico pueden clasificarse en dos grandes grupos: los no estequiométricos y los estequiométricos. La diferencia principal entre estos dos tipos es la manera como utilizan las constantes de restricción de elementos en los cálculos. Los métodos que eliminan estas restricciones son conocidos como *algoritmos estequiométricos*, los algoritmos que utilizan las restricciones de abundancia explícitamente se conocen como *algoritmos no estequiométricos*. Las **figuras 5.1** y **5.2** esquematizan esta clasificación presentando algunos de los algoritmos dentro de ella; la mayoría de los algoritmos presentados ahí se resumen en esta sección.

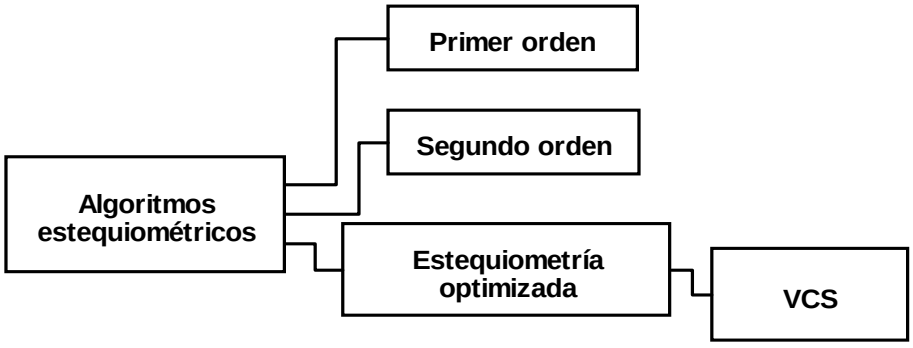

Figura 5.1 Algoritmos estequiométricos para obtener el equilibrio químico.

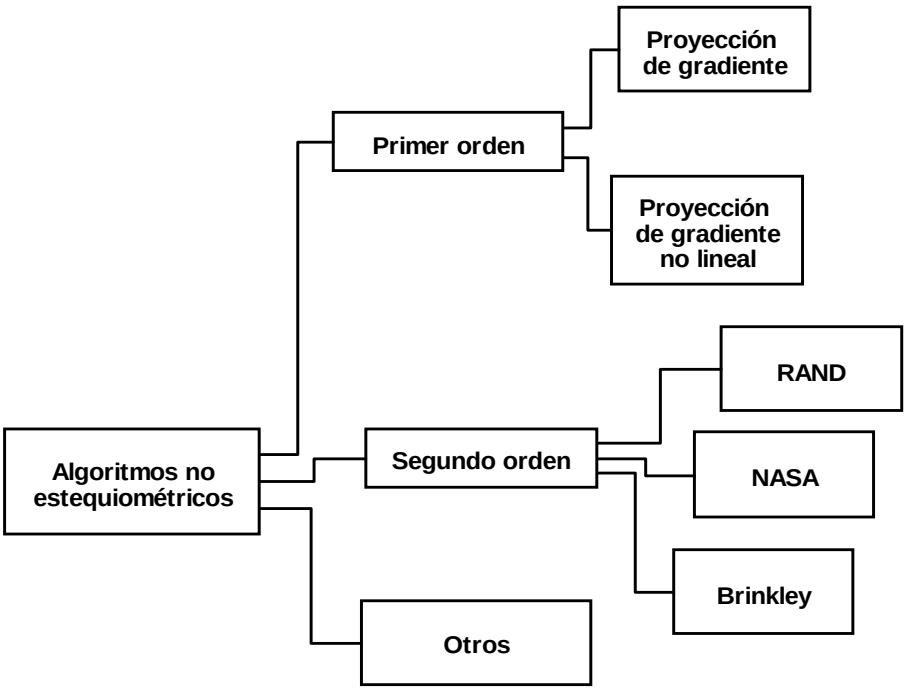

Figura 5.2 Algoritmos no estequiométricos para obtener el equilibrio químico

5.2 Algoritmos no estequiométricos

5.2.1 Proyección de gradiente

Los cambios en los números de moles a partir de una estimación dada n_i se calculan por medio de

$$\underline{\delta \mathbf{n}} = \mathbf{P} \left. \frac{\partial G}{\partial \underline{\mathbf{n}}} \right|_{\underline{\mathbf{n}}} = -\mathbf{P}\underline{\mathbf{g}} \tag{5.2}$$

$$\underline{\mathbf{n}}^{\text{nuevo}} = \underline{\mathbf{n}} + \omega \underline{\delta \mathbf{n}} \tag{5.3}$$

la matriz de proyección $\mathbf{P}$ está dada por

$$\mathbf{P} = \mathbf{I} - \mathbf{A}^{\mathrm{T}} \left(\mathbf{A}\mathbf{A}^{\mathrm{T}} \right)^{-1} \mathbf{A} \tag{5.4}$$

Estas ecuaciones se utilizan de manera iterativa para minimizar la función de Gibbs del sistema.

5.2.2 Proyección del gradiente no lineal

Las restricciones de no negatividad en el número de moles se incorporan por medio de la transformación logarítmica

$$\underline{\mathbf{y}} = \ln \underline{\mathbf{n}} \tag{5.5}$$

esto resulta en el problema transformado

$$G(\underline{\mathbf{y}}) = \text{mín!} \tag{5.6}$$

de manera que

$$\mathbf{A} \exp(\underline{\mathbf{n}}) = \underline{\mathbf{b}} \tag{5.7}$$

Si se utiliza la serie de aproximación local de Taylor para las restricciones se obtiene

$$\mathbf{A}\underline{\mathbf{n}}\underline{\delta \mathbf{y}} = 0 \tag{5.8}$$

El método de proyección de gradiente se puede utilizar para minimizar $G(\underline{\delta \mathbf{y}})$, que se puede expresar en la forma

$$\mathbf{A}\mathbf{D}\underline{\delta \mathbf{y}} = \underline{\mathbf{0}} \tag{5.9}$$

donde $\mathbf{D}$ es la matriz diagonal con elementos n_i. El algoritmo resultante calcula los cambios en $\underline{\mathbf{y}}$ por medio de

$$\underline{\delta \mathbf{y}} = -\mathbf{P} \left. \frac{\partial G}{\partial \underline{\mathbf{y}}} \right|_{\underline{\mathbf{y}}} = -\mathbf{P}\mathbf{D}\underline{\mathbf{g}} \tag{5.10}$$

La matriz de proyección $\mathbf{P}$ se calcula en cada iteración

$$\mathbf{P} = \mathbf{I} - \mathbf{A}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\left(\mathbf{A}\mathbf{D}\mathbf{D}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\right)^{-1}\mathbf{A}\mathbf{D} \tag{5.11}$$

Una dificultad práctica con el método es satisfacer las restricciones de abundancia de elementos de la **ecuación 5.7** tiende a deteriorarse a medida que proceden las iteraciones, a menos que el parámetro de etapa sea muy pequeño. El fenómeno de desviación y la minimización de los errores por redondeo se pueden tratar para dar la ecuación

$$\underline{\delta\mathbf{n}} = -\mathbf{P}\underline{\mathbf{g}} + \beta\mathbf{A}^{\mathrm{T}}\left(\mathbf{A}\mathbf{A}^{\mathrm{T}}\right)^{-1}\underline{\delta\mathbf{b}} \tag{5.12}$$

donde β es un parámetro de etapa , que casi siempre se hace igual a la unidad y

$$\underline{\delta\mathbf{b}} = \underline{\mathbf{b}} - \mathbf{A}\underline{\mathbf{n}} \tag{5.13}$$

5.2.3 Algoritmos Brinkley–NASA–RAND (BNR)

Estos, contemplan el problema como la resolución de un conjunto de ecuaciones no lineales.

Las condiciones de equilibrio, en que se incorpora el potencial químico, son

$$\frac{g_i^\circ}{RT} + \ln\frac{n_i}{n_t} - \sum_{k=1}^{M}\psi_k a_{ki} \qquad i = 1, 2, \ldots, N \tag{5.14}$$

donde

$$\psi_k = \frac{\lambda_k}{RT} \tag{5.15}$$

y

$$n_t = \sum_{i=1}^{N} n_i \tag{5.16}$$

la restricción de abundancia de elementos

$$\sum_{i=1}^{N} a_{ki} n_i - b_k = 0 \qquad k = 1, 2, \ldots, M \tag{5.17}$$

Las tres variaciones (Brinkley, NASA y RAND) difirieren sólo en la forma de tratamiento numérico de las variables de los números de moles. La versión RAND utiliza n_t como variable

y emplea el método de Newton-Raphson en las **ecuaciones 5.14**, **5.16** y **5.17**, lo que es equivalente a linealizar los términos logarítmicos en las **ecuaciones 5.14**. Las versiones Brinkley y NASA utilizan «$\ln n_i$» como variables y emplean el método de Newton-Raphson en el mismo conjunto de ecuaciones, lo que equivale a linealizar los términos exponenciales resultantes en las **ecuaciones 5.16** y **5.17**.

5.2.4 La variación RAND

Primero se consideran los problemas compuestos por una sola fase i con especies múltiples y después se generaliza al problema de fases múltiples. La linealización de la **ecuación 3.49** respecto a una estimación arbitraria de la solución $(\underline{\mathbf{n}}, \underline{\psi})$ produce, después de ordenar

$$-\frac{1}{RT}\sum_{j=1}^{N}\left.\frac{\partial g_i}{\partial n_j}\right|_{\underline{\mathbf{n}}}\delta n_j + \sum_{k=1}^{M} a_{ki}\delta\psi_k = \frac{g_i}{RT} - \sum_{k=1}^{M} a_{ki}\psi_k \qquad i = 1, 2, \ldots, N \tag{5.18}$$

donde

$$\delta\psi_k = \psi_k^{\text{anterior}} - \psi_k \tag{5.19}$$

y

$$\delta n_j = n_j^{\text{anterior}} - n_j \tag{5.20}$$

las cantidades $\underline{\mathbf{n}}$ y $\underline{\mathbf{n}}^{\text{anterior}}$ están relacionadas a través de las restricciones de abundancia de elementos por

$$\mathbf{A}\delta\underline{\mathbf{n}} = \underline{\delta\mathbf{b}} \tag{5.21}$$

donde $\underline{\delta\mathbf{b}} = \underline{\mathbf{b}}^{\text{anterior}} - \underline{\mathbf{b}}$ y

$$\underline{\mathbf{b}} = \mathbf{A}\underline{\mathbf{n}} \tag{5.22}$$

las incógnitas son $\underline{\delta\mathbf{n}}$ y $\underline{\delta\psi}$. Para sistemas ideales, el número de ecuaciones lineales que deben resolverse en cada iteración del procedimiento se puede reducir de $(N + M)$ hasta $(M + 1)$ eliminando las variables $\underline{\delta\mathbf{n}}$ en las **ecuaciones 5.18** y **5.21**. La **ecuación 5.1** resulta en

$$\frac{1}{RT}\frac{\partial g_i}{\partial n_j} = \frac{\delta_{ij}}{n_j} - \frac{1}{n_t} \tag{5.23}$$

donde δ_{ij} es la delta generalizada de Kronecker, es decir,

$$\delta_{ij} = \begin{cases} 1 & \text{para } i = j \\ 0 & \text{de otra forma.} \end{cases} \tag{5.24}$$

Al sustituir la **ecuación 5.23** en la **ecuación 5.18** es posible obtener explícitamente $\underline{\delta \mathbf{n}}$ en términos de ψ en la **ecuación 5.19**

$$\delta n_j = n_j \left(\sum_{k=1}^{M} a_{kj}\psi_k + u - \frac{g_j}{RT} \right) \qquad j = 1, 2, \ldots, N \tag{5.25}$$

donde

$$u = \frac{\sum_{k=1}^{N} \delta n_j}{n_t} = \frac{\delta n_t}{n_t} \tag{5.26}$$

Al sustituir la **ecuación 5.25** en la **5.21** se obtienen las M ecuaciones lineales

$$\sum_{i=1}^{M} \left(\sum_{k=1}^{N} a_{ik} a_{jk} n_k \right) \psi_i + b_j u = \sum_{k=1}^{N} a_{jk} n_k \frac{g_k}{RT} + \delta b_j \qquad j = 1, 2, \ldots, M \tag{5.27}$$

Ahora utilizando la **ecuación 5.26** y sumando la **ecuación 5.25**

$$\sum_{i=1}^{M} b_i \psi_i - n_z u = \sum_{k=1}^{N} n_k \frac{g_k}{RT} \tag{5.28}$$

Cada iteración consiste en resolver el conjunto de $(M + 1)$ ecuaciones lineales **5.27** y **5.28**. En el caso general, cuando hay π_m fases de especies múltiples y π_s fases de especies únicas, las **ecuaciones 5.25**, **5.27** y **5.28** se transforman respectivamente en,

$$\delta n_j = \begin{cases} n_j \left(\sum_{i=1}^{M} a_{ij}\psi_i + u_\alpha - \frac{g_i}{RT} \right) & \text{para especies en fases de especies múltiples,} \\ u_\alpha n_j & \text{para especies en fases de especies únicas.} \end{cases} \tag{5.29}$$

$$\sum_{i=1}^{N} \sum_{k=1}^{N} a_{ik} a_{jk} n_k \psi_i + \sum_{\alpha=1}^{\pi} b_{j\alpha} u_\alpha = \sum_{k=1}^{N} a_{jk} n_k \frac{g_i}{RT} + \delta b_j \qquad j = 1, 2, \ldots, M \tag{5.30}$$

$$\sum_{i=1}^{M} b_{i\alpha}\psi_i - n_z u_\alpha = \sum_{k=1}^{N} n_{k\alpha}\frac{g_{k\alpha}}{RT} \qquad \alpha = 1,2,\ldots \pi_s + \pi_m \tag{5.31}$$

donde el subíndice α se refiere a una fase. El algoritmo RAND consiste en resolver el conjunto de $(M + \pi)$ ecuaciones lineales **5.30** y **5.31**. Donde

$$\pi = \pi_u + \pi_m \tag{5.32}$$

Algunas veces es posible encontrar problemas numéricos no triviales en la aplicación de las **ecuaciones 5.30** y **5.31**. Estos problemas surgen cuando la matriz de coeficientes lineales se hace singular en algún punto del cálculo. Este algoritmo es un método de minimización. En cada iteración se satisfacen las restricciones de abundancia de elementos e iterativamente el algoritmo minimiza la energía libre de Gibbs.

5.2.5 La variación Brinkley

Puesto que «$\ln n_i$» son la variables independientes, se establece

$$y_i = \ln n_i \tag{5.33}$$

entonces para una sola fase ideal las **ecuaciones 5.14** y **5.17** se convierte en

$$\exp y_i = n_i = \exp\left(\sum_{k=1}^{M} a_{ki}\psi_k - \frac{g_i^\circ}{RT} + \ln n_t\right) \qquad i = 1,2,\ldots,N \tag{5.34}$$

$$\sum_{i=1}^{N} a_{ji}\exp(y_i) = b_j \qquad j = 1,2,\ldots,M \tag{5.35}$$

al sustituir la **ecuación 5.34** en la **5.35** se obtiene

$$\sum_{i=1}^{N} \exp\left(\sum_{k=1}^{M} a_{ki}\psi_k - \frac{g_i^\circ}{RT} + \ln n_t = b_j\right) \qquad j = 1,2,\ldots,M \tag{5.36}$$

Finalmente

$$\sum_{i=1}^{N} \exp\left(\sum_{k=1}^{M} a_{ki}\psi_k - \frac{g_i^\circ}{RT}\right) = 1 - \frac{n_z}{n_t} \tag{5.37}$$

Si se eligen las estimaciones $\underline{\psi}$, n_t y se determina $\underline{\mathbf{n}}$ de la **ecuación 5.34**, las ecuaciones de la iteraciones de Newton-Raphson que se obtienen de linearizar las **ecuaciones 5.35** y **5.37** son

$$\sum_{i=1}^{M}\left(\sum_{k=1}^{N} a_{ik}a_{jk}n_k\right)\delta\psi_i + b_j v = \delta b_j \qquad j = 1,2,\ldots,M \tag{5.38}$$

y

$$\sum_{i=1}^{M} b_i\delta\psi_i - n_z v = n_t - \sum_{k=1}^{N} n_k - n_z \tag{5.39}$$

donde

$$v = \ln\frac{n_t}{n_t^{\text{anterior}}} \tag{5.40}$$

y $\underline{\delta\psi}$ está definida por la **ecuación 5.19**. Los nuevos valores de $\underline{\psi}$ y $\ln n_t$ se obtiene de

$$\underline{\psi} = \underline{\psi}^{\text{anterior}} + \omega\underline{\delta\psi} \tag{5.41}$$

y

$$\ln n_t = \ln n_t^{\text{anterior}} + \omega v \tag{5.42}$$

5.2.6 La variación NASA

En la variación NASA las restricciones de abundancia de elementos y de números de moles (logaritmos), presentes en la variación Brinkley, se relajan.

Considerando el caso de una solución ideal y utilizando las variables logarítmicas para el número de moles, se linealizan las **ecuaciones 5.14** y **5.17** en las estimaciones de $(\underline{\mathbf{n}}, \underline{\psi})$ para dar,

$$\delta(\ln n_k) - \delta(n_t) = \sum_{j=1}^{M} a_{jk}\left(\psi_j + \delta\psi_j\right) - \frac{\mathcal{G}_k}{RT} \qquad k = 1,2,\ldots,N \tag{5.43}$$

$$\sum_{k=1}^{N} a_{jk} n_k \delta(\ln n_k) = \delta b_j \qquad j = 1, 2, \ldots, M \tag{5.44}$$

La variación NASA siempre incluye los elementos atómicos como especies en los cálculos. Al numerar las especies de tal manera que los primeros M son elementos, se tiene

$$\psi_i = \frac{g_j}{RT} \qquad j = 1, 2, \ldots, M \tag{5.45}$$

por lo tanto, para las especies elementales de la **ecuación 5.43** se tiene

$$\psi_j + \delta\psi_j = \delta(\ln n_j) - \delta(\ln n_t) + \frac{g_j}{RT} \qquad j = 1, 2, \ldots, M \tag{5.46}$$

cuando k no representa una especie elemental, la **ecuación 5.43** se representa utilizando la **ecuación 5.46**, con $k = M+1, \ldots, N$, como

$$\delta(\ln n_k) = \delta(\ln n_t)\left(1 - \sum_{i=1}^{M} a_{ik}\right) + \sum_{i=1}^{M} a_{ik}\delta(\ln n_i) + \sum_{i=1}^{M} a_{ik}\frac{g_i}{RT} - \frac{g_k}{RT} \tag{5.47}$$

La sustitución de la **ecuación 5.47** en la **5.44** da como resultado un conjunto de M ecuaciones lineales en que intervienen $\delta(\ln n_j)$ para cada $j = 1, 2, \ldots M$,

$$\begin{aligned} n_j\delta(\ln n_j) + \sum_{i=1}^{M} \delta(\ln n_i)\left(\sum_{k=M+1}^{N} a_{ik}a_{jk}n_k\right) & \\ + \delta(\ln n_t)\sum_{k=M+1}^{N} a_{jk}n_k\left(1 - \sum_{i=1}^{M} a_{ik}\right) & \\ = \sum_{k=M+1}^{N} -a_{jk}n_k\left(\sum_{i=1}^{M} a_{ik}\frac{g_i}{RT} + \frac{g_k}{RT}\right) + \delta b_j & \end{aligned} \tag{5.48}$$

Se obtiene una ecuación final linealizando

$$\sum_{i=1}^{N} \exp(\ln n_t) = \exp(\ln n_t) - n_z \tag{5.49}$$

respecto a $(\underline{\mathbf{n}}, n_t)$, dando

$$\sum_{i=1}^{M} \delta(\ln n_i)\left(n_i + \sum_{k=M+1}^{N} a_{ik} n_k\right) + \delta(\ln n_t)\left[-n_t + \sum_{k=M+1}^{N} n_k\left(1 - \sum_{i=1}^{M} a_{ik}\right)\right] = n_t - \sum_{k=1}^{N} n_k - n_z - \sum_{k=M+1}^{N} n_k\left(-\sum_{i=1}^{M} a_{ik}\frac{g_i}{RT} + \frac{g_k}{RT}\right) \tag{5.50}$$

Las **ecuaciones 5.48** y **5.50** son un conjunto de $(M+1)$ ecuaciones lineales con $(M+1)$ incógnitas $\delta(\ln n_j)$ para cada $j = 1, 2, \ldots M$ y $\delta(\ln n_t)$. El algoritmo NASA es un método de ecuaciones no lineales. No satisfacen ni la abundancia de elementos ni las condiciones de equilibrio en cada iteración.

5.3 Algoritmos estequiométricos

Los algoritmos estequiométricos eliminan las restricciones de abundancia de elementos del problema de minimización, lo que da como resultado una formulación sin restricciones. Esto se lleva acabo transformando las N incógnitas correspondientes a los números de moles $\underline{\mathbf{n}}$, que están restringidos por las M ecuaciones de abundancia de elementos a un nuevo conjunto de variables del «avance de reacción» iguales en número a $R = (N - M)$. Los cambios en los números de moles $\underline{\delta\mathbf{n}}$ a partir de cualquier estimación de $\underline{\mathbf{n}}$ que satisfagan las restricciones de abundancia de elementos están relacionados con las nuevas variables $\underline{\xi}$ por

$$\underline{\delta\mathbf{n}} = \mathbf{N}\underline{\delta\xi} \tag{5.51}$$

Considerando la función de Gibbs como una función de las variables de grado de reacción $\underline{\xi}$, el problema del equilibrio químico es el de minimizar $\underline{\mathbf{G}}(\xi)$. Esta formulación es equivalente a la formulación química clásica de las condiciones de equilibrio

$$\Delta\underline{\mathbf{G}} = \mathbf{N}^{\mathrm{T}}\underline{\mathbf{g}}(\xi) = 0 \tag{5.52}$$

Una de las principales diferencias entre los algoritmos estequiométricos y los no estequiométricos se refiere al número total de variables independientes que se deben determinar esencialmente. Utilizando las **ecuaciones 5.14** y **5.17** en forma directa los algoritmos no

estequiométricos incorporan las restricciones de abundancia de elementos por la introducción de un conjunto adicional de M variables. Se ha visto que en varios de estos algoritmos también se introduce una nueva variable por cada fases del sistema. Esto resulta en un total de $(N+M+\pi)$ variables conjuntas. En los algoritmos estequiométricos el número de variables siempre es $(N-M)$, sin importar si las fases son ideales. Así para los sistemas no ideales, los algoritmos estequiométricos siempre tienen menos variables. Para los sistemas ideales con un número de fases pequeño, los algoritmos no estequiométricos tienen por lo general menos variables.

5.3.1 Algoritmo de primer orden

Las variables $\underline{\xi}$ en cada iteración son ajustadas por las cantidades $\underline{\delta\xi}$, donde

$$\delta\xi = -\frac{\partial G}{\partial \xi_j} = -\Delta G_j = -\underline{\nu}_j \underline{\mathbf{g}} \qquad j = 1, 2, \ldots, R \tag{5.53}$$

Los números de moles se ajustan por medio de la **ecuación 5.51**. Este algoritmo converge con bastante lentitud.

5.3.2 Algoritmo de segundo orden

Al aplicar el método de Newton-Raphson a las **ecuaciones 5.52** se obtiene

$$\underline{\xi} = \left.\frac{\partial^2 G}{\partial \xi^2}\right|^{-1}_{\underline{\mathbf{n}}} \left.\frac{\partial G}{\partial \xi}\right|_{\underline{\mathbf{n}}} \tag{5.54}$$

Puesto que N por lo general es grande comparado con M, la solución numérica de estas ecuaciones lineales puede ser una parte del algoritmo que requiera mucho tiempo. Este algoritmo por lo general se emplea en sistemas relativamente simples.

5.3.3 Algoritmo Villars-Cruise-Smith (VCS)

Este algoritmo se encuentra entre los algoritmos de primer y segundo orden. El algoritmo principia con la **ecuación 5.54**. En el caso de una sola fase ideal, la matriz Hessiana $(\partial^2 G/\partial \xi^2)$ está dada por

$$\frac{\partial^2 G}{\partial \xi_i \partial \xi_j} = \frac{\partial}{\partial \xi_j}\left(\sum_{k=1}^{N} \nu_{ki} g_k\right) = RT \sum_{k=1}^{N}\sum_{l=1}^{N} \nu_{ki}\nu_{lj}\left(\frac{\delta_{kl}}{n_k} - \frac{1}{n_t}\right) \qquad i, j = 1, 2, \ldots, R \tag{5.55}$$

La **ecuación 5.55** puede volverse a escribir como

$$\frac{1}{RT}\frac{\partial^2 G}{\partial \xi_i \xi_j} = \sum_{k=1}^{N} \frac{\nu_{ki}\nu_{kj}}{n_k} - \frac{\overline{\nu}_i \overline{\nu}_j}{n_t} \qquad i, j = 1, 2, \ldots, R \tag{5.56}$$

donde

$$\overline{\nu}_i = \sum_{k=1}^{N} \nu_{ki} \tag{5.57}$$

Si la matriz estequiométrica es de forma canónica, el producto $\nu_{ki}\nu_{kj}$ para $i \neq j$ es cero cuando k se refiere a una especie no componente ($> M$) puesto que cada especie no componente tiene un coeficiente estequiométrico distinto de cero sólo en un vector estequiométrico. Cuando $i = j$, $\nu_{ki}\nu_{kj} = 1$ para estos valores de k. Los elementos de la matriz Hessiana, numerando las especies componentes desde 1 hasta M y las especies no componentes desde $(M + 1)$ hasta N se expresan como

$$\frac{1}{RT}\frac{\partial^2 G}{\partial \xi_i \xi_j} = \frac{\delta_{ij}}{n_{j+M}} + \sum_{k=1}^{M} \frac{\nu_{ki}\nu_{kj}}{n_k} - \frac{\overline{\nu}_i \overline{\nu}_j}{n_t} \qquad i, j = 1, 2, \ldots, R \tag{5.58}$$

Si se escoge que las especies componentes sean aquellas que tengan los números de moles más altos, esto tiende a hacer que el segundo término del miembro de la derecha sea pequeño y el primer término sea grande. El último término desaparece si cualquiera de los valores $\overline{\nu}_i$ o $\overline{\nu}_j$ desaparecen y en cualquier caso a menudo es pequeño en comparación con el primer término debido a la presencia de n_t en el denominador.

Si se forma la matriz **N** en esta forma, se hace la suposición de que la matriz Hessiana es diagonal y se invierte

$$RT\left(\frac{\partial G}{\partial \xi_i \xi_j}\right)^{-1} \approx \left(\frac{1}{n_{i+M}} + \sum_{k=1}^{M} \frac{\nu_{ki}^2}{n_k} - \frac{\overline{\nu}_i^2}{n_t}\right)^{-1} \delta_{ij}. \tag{5.59}$$

Así, el algoritmo VCS para una sola fase ideal consiste en utilizar la **ecuación 5.54** con la **5.59** e iterativamente se ajusta cada ecuación estequiométrica por una cantidad

$$\delta\xi_j = -\left(\frac{1}{n_{j+M}} + \sum_{k=1}^{M}\frac{\nu_{ki}^2}{n_k} - \frac{\overline{\nu}_j^2}{n_t}\right)^{-1}\frac{\Delta G_j}{RT} \qquad j = 1, 2, \ldots, R. \tag{5.60}$$

En el caso de un sistema ideal con fases múltiles, la **ecuación 5.60** se transforma en

$$\delta\xi_j = \begin{cases} -\left(\dfrac{\delta^*_{j+M,\alpha}}{n_{j+M}} + \displaystyle\sum_{k=1}^{M}\frac{\nu_{kj}^2\delta_{k\alpha}}{n_k} - \sum_{\alpha=1}^{\pi}\sum_{k=1}^{N}\frac{(\nu_{kj}\delta_{k\alpha})^2}{n_{t\alpha}}\right)^{-1}\dfrac{\Delta G_j}{RT} & \text{Siempre que cuando menos una especie para la cual } \nu_{kj} \neq 0 \text{ se encuentre en una fase multicomponente,} \\ -\dfrac{\Delta G_j}{RT} & \text{en caso contrario.} \end{cases} \tag{5.61}$$

Donde α representa a la fase α. El valor de $\delta^*_{k\alpha}$ es la unidad si la especie k está en cualquier fase multicomponente α y en caso contrario es cero, y $\delta_{k\alpha}$ es la unidad si la especie k se encuentra en la fase particular α multicomponente y cero en caso contrario.

5.4 Número de moles iniciales

Una estimación del número de moles iniciales puede generarse por un procedimiento de programación lineal. Debido a la presencia del término logarítmico en la expresión del potencial químico, n_i° en una fase de especies múltiples debe escogerse como positivo, pero para una fase de una sola especies puede considerarse como cero.

El procedimiento de programación lineal se basa en el uso de la expresión de potencial químico sin el término logarítmico. Esto implica la resolución de cierto problema de programación lineal y el proceder de acuerdo a la naturaleza de la solución.

El problema de programación lineal es:

$$\text{mín!} = \underline{\mathbf{n}}^{\mathrm{T}}\underline{\mathbf{g}}^\circ \qquad \text{tal que} \quad \mathbf{A}\underline{\mathbf{n}} = \underline{\mathbf{b}},\ \underline{\mathbf{n}} \geq \underline{\mathbf{0}} \tag{5.62}$$

El primer tipo se caracteriza por los valores de r, rango de la matriz fórmula, representados por: $(n_1^{\mathrm{PL}}, n_2^{\mathrm{PL}}, \ldots, n_r^{\mathrm{PL}})$, siendo el resto cero.[7] En este caso el número de moles total, $n_{t\alpha}$, en cada fase de especies múltiples está determinado y la matriz estequiométrica $\mathbf{N}$ se calcula en forma canónica, utilizando la especie r como especie componente. Entonces, haciendo las

[7] El exponente PL indica un valor de programación lineal.

aproximaciones de que $n_{t\alpha}$ y $(n_1^{\mathrm{PL}}, n_2^{\mathrm{PL}}, \ldots, n_r^{\mathrm{PL}})$ permanecen constantes, se calcula la g_k para estas especies componentes y haciendo los cambios de energía libre de reacción iguales a cero, también se hace que:

$$\xi_i = \delta_{i\alpha} n_{t\alpha} \exp\left\{\frac{1}{RT}\left(-g_k^\circ - \sum_{k=1}^{r} g_k(\underline{\mathbf{n}}^{\mathrm{PL}})\nu_{kj}\right)\right\} \qquad \text{para} \qquad i = r+1, r+2, \ldots, N \tag{5.63}$$

donde $j = i - r$ y $\delta_{i\alpha}$ es la unidad para una especie en una fase de especies múltiples y cero en caso contrario. Finalmente, se introduce un parámetro de etapa ω y se hace:

$$n_i^\circ = n_i^{\mathrm{PL}} + \omega \sum_{j=1}^{R} \nu_{ij}\xi_j \qquad \text{para} \qquad i = 1, 2, \ldots, N \tag{5.64}$$

donde ω se escoge de manera que todos los números permanezcan positivos.

El segundo tipo se caracteriza por haber menos valores de n_i^{PL} positivos que r, siendo el resto cero. En este caso se hace n_i° igual al mayor de los valores n_i^{PL} y a un número positivo arbitrariamente pequeño, como 10^{-15}.

5.5 Selección del algoritmo

De entre los algoritmos disponibles para calcular la composición en equilibrio de un sistema químico se ha elegido el algoritmo VCS pues es adecuado para manejar problemas, especialmente en aquellos que intervienen fases con una sola especie, como los que surgen en aplicaciones metalúrgicas. Esto se debe al hecho de que las restricciones de no negatividad en el número de moles de la especie son fácilmente manejadas en este algoritmo. En el algoritmo VCS, además, no hay ecuaciones lineales que resolver en cada iteración: en cualquiera de los algoritmos tipo BNR para un sistema ideal se requiere de la solución de $(M + \pi)$ ecuaciones lineales en cada iteración. Estas ecuaciones pueden tener una matriz de coeficientes singular para ciertos problemas y esto ocasiona problemas prácticos, el algoritmo VCS lo evita. El número de variables independientes desconocidas en el algoritmo VSC, y en todos los algoritmos estequiométricos, siempre es $(N-M)$ mientras que en los algoritmos no estequiométricos es $(N + M + \pi)$ aunque, para sistemas ideales casi siempre se reduce a $(M + \pi)$. En suma, el algoritmo VCS requiere de menos recursos computacionales y es más estable para sistemas de interés metalúrgico.

Desarrollo

6 Programa de cómputo

El programa `Ceqc` *determina la composición en equilibrio de un sistema multifásico, basado en la minimización de la energía libre de Gibbs bajo condiciones de presión y temperatura específicas. El algoritmo empleado,* VCS*, consiste en la formulación estequiométrica del equilibrio químico.*

6.1 Generalidades

El programa de cómputo `Ceqc` fue escrito en el lenguaje interpretado `Python`. Teóricamente, `Ceqc` está diseñado para trabajar en cualquier sistema operativo que soporte y tenga instalado un interprete `Python` junto con las librerías no estándar requeridas por `Ceqc`; Aunque `Ceqc` no ha sido probado en un sistema operativo diferente a `Windows XP`.

Las librerías no estándar requeridas por `Ceqc`[8] son: `Numeric`, `Scientific` y `scipy`. La versión empleada del interprete `Python` fue la 2.3.[9] Se recomienda utilizar la `Enthought Edition` que ya incluye las librerías necesarias.

6.2 Descripción del funcionamiento del programa `Ceqc`

Como ejemplo, para mostrar el funcionamiento de `Ceqc` se utiliza el sistema

$$\{(\mathrm{Zn}, \mathrm{CO}, \mathrm{H_2}, \mathrm{ZnO}, \mathrm{CO_2}, \mathrm{H_2O}, \mathrm{CH_4}),\ \ (\mathrm{Zn}, \mathrm{C}, \mathrm{H}, \mathrm{O})\}$$

a la temperatura de 1100 K y presión de 1.8 atm. Inicialmente hay 1.2 mol de ZnO y 1.0 mol de $\mathrm{CH_4}$.

El programa comienza con una pantalla de bienvenida que explica de manera general qué hace, seguido de esto pide las especies del sistema, temperatura, presión y moles iniciales de las especies, la información proporcionada por el usuario se representa con letra italizada:

---

[8] Todas disponibles desde `www.python.org`

[9] Aunque es compatible con versiones superiores.

```
1   Ceqc
2
3       Programa para el cálculo del equilibrio químico en un
4       sistema complejo.
5
6       El cálculo se fundamenta en el principio de minimización
7       de la energía libre de Gibss. El algoritmo se basa en
8       la formulación estequiométrica optimizada. Los modelos de
9       solución están definidos en en el archivo editable
10      'modelos.py'.
11
12                                  César Ulises Santiago Cortés
13
```

```
14  > Escriba las especies del sistema >>  ZnO CO CH4 H2 Zn CO2 H2O
15  > temperatura(K) >>  1100
16  > presión(atm) >>  1.8
17
18  > Moles iniciales de  Zn >>   0
19  > Moles iniciales de  CO2 >>   0
20  > Moles iniciales de  CO >>   0
21  > Moles iniciales de  H2 >>   0
22  > Moles iniciales de  ZnO >>   1.2
23  > Moles iniciales de  H2O >>   0
24  > Moles iniciales de  CH4 >>   1.0
25
```

seguido de esto aparece una pantalla que muestra la especie y la fase donde, dada la temperatura, se supone se encuentra.

```
26      ============================================================
27
28      > La fase de cada especie que tomaré en consideración es:
29
30      >   1.  Zn        Líquido   (LIQUIDO)
31      >   2.  CO2       Gas       (GAS)
32      >   3.  CO        Gas       (GAS)
33      >   4.  H2        Gas       (GAS)
34      >   5.  ZnO       Sólido    (ZINCITA)
35      >   6.  H2O       Gas       (VAPOR)
36      >   7.  CH4       Gas       (GAS)
37
```

Al final se pregunta si se desea cambiar la fase de alguna especie:

```
> Desea cambiar la fase de alguna de ellas >>  ↩
    > ¡Interpreté su respuesta como un NO!

============================================================
```

En este caso se ha dicho ↩, «*no*», no se desea cambiar. Si se hubiese contestado con un «*si*» habría aparecido una pantalla mostrando las opciones de cambio, pero éstas se explicaran más adelante en esta misma sección. A continuación aparece otra pantalla mostrando la composición de las fases del sistema:

```
> Fases de sistema:

>     liquido:
>       -----------------------------------------
>         Zn        Líquido   (LIQUIDO)

>     gas:
>       -----------------------------------------
>         H2        Gas       (GAS)
>         CH4       Gas       (GAS)
>         H2O       Gas       (VAPOR)
>         CO2       Gas       (GAS)
>         CO        Gas       (GAS)

>     solido_0:
>       -----------------------------------------
>         ZnO       Sólido    (ZINCITA)


> ¿Desea cambiar la composición de alguna fase de sistema? >> ↩
    > ¡Interpreté su respuesta como un NO!

============================================================
```

aquí también se pregunta si se desea cambiar la composición de alguna de las fases, por el momento se ha respondido con un ↩ para pasar a la siguiente pantalla donde se muestra un listado de los modelos de solución disponibles y definidos en el módulo `modelos.py`.

```
67      > Los modelos disponibles para las fases del sistema son:
68
69      >   1. g_ideal_gas
70      >   2. g_ideal_liquido
71      >   3. g_ideal_solucion_solida
72      >   4. g_ideal_solido
73      >   5. parametros_de_interaccion
74
75      > Por favor seleccione algún modelo para cada fase.
76          > Modelo para "liquido" >>  2
77          > Modelo para "gas" >>  1
78          > Modelo para "solido_0" >>  4
79
```

En este caso para la fase `liquido` se ha seleccionado el modelo `g_ideal_liquido`, para la fase `gas`, el modelo `g_ideal_gas` y para la fase `solido_0`, el modelo `g_ideal_solido`. Para la fase `solido_0` pudo haberse escogido el modelo `g_ideal_solucion_solida` pero al ser esta fase monofásica se realizarán menos cálculos si se escoge `g_ideal_solido`.

Por último, aparecen dos pantallas finales, una muestra el balance de abundancia de elementos antes y después del cálculo de equilibrio:

```
80  Abundancia de elementos
81      ============================================================
82          Elemento    Moles iniciales     Moles finales
83      ============================================================
84                 Zn      1.2                 1.2
85                  C      1                   1
86                  H      4                   4
87                  O      1.2                 1.2
88      ============================================================
89
```

la siguiente pantalla muestra el balance final del cálculo, el cual es el objetivo del programa. La pantalla presenta la composición de las fases en número de moles y en fracción mol para cada fase y especie. Al terminar este cálculo se mostrara en pantalla el símbolo `>>>`, éste indica que se ha pasado a la «shell» del interprete `Python`, y que el programa `Ceqc` ha terminado su ejecución. Sin embargo, la información generada por `Ceqc` aún continúa almacenada y a ella puede accederse a través de la `clase Ceqc` como se muestra a continuación.

```
90      Balance final
91      ============================================================
92              Especie     Moles Finales   Fracción mol
93      ============================================================
94      liquido
95              Zn          0.6221444       1
96      gas
97              H2          1.218418        0.5475976
98              CH4         0.3874877       0.1741499
99              H2O         0.00660645      0.002969158
100             CO2         0.003025729     0.001359863
101             CO          0.6094865       0.2739235
102     solido_0
103             ZnO         0.5778556       1
104     ============================================================
105
```

Para tener acceso a la lista de elementos presente en el sistema deberá escribirse después de `>>>`.

```
106 >>> Ceqc.v_elementos↩
107 ['Zn', 'C', 'H', 'O']
108
```

El resultado será todo lo que después de ↩ no tenga alguna tabulación, en este caso `['Zn', 'C', 'H', 'O']`. La lista de compuestos se obtiene:

```
109 >>> Ceqc.v_especies↩
110 ['Zn', 'CO', 'H2', 'ZnO', 'CO2', 'H2O', 'CH4']
111
```

La temperatura:

```
112 >>> Ceqc.T↩
113 1100.0
114
```

La energía estándar ideal, $g^{\circ}_{\text{especie}}$, en julios y en el mismo orden que `v_especies`:

```
115 >>> Ceqc.v_g0↩
116 [-67587.001603581914, -347043.05072867317, -162166.60000363566, -425702.9532
117 7306754, -656638.403241197, -472568.60247096658, -309381.94669528573]
118
```

La matriz fórmula:

```
119 >>> Ceqc.m_formula↩
120 array([[ 1.,  0.,  0.,  1.,  0.,  0.,  0.],
121        [ 0.,  1.,  0.,  0.,  1.,  0.,  1.],
122        [ 0.,  1.,  0.,  1.,  2.,  1.,  0.],
123        [ 0.,  0.,  2.,  0.,  0.,  2.,  4.]])
124
```

La matriz estequiométrica:

```
125 >>> Ceqc.m_nula↩
126 array([[ 1.,  1., -1.],
127        [-1.,  0., -1.],
128        [ 0., -1., -2.],
129        [-1., -1.,  1.],
130        [ 1.,  0.,  0.],
131        [ 0.,  1.,  0.],
132        [ 0.,  0.,  1.]])
133
```

O bien, el vector fórmula de `ZnO`:

```
134 >>> Ceqc.vector('ZnO')↩
135 {'H': 0, 'Zn': 1, 'C': 0, 'O': 1}
136
```

La lista completa de «miembros» de `Ceqc` puede obtenerse tecleando `dir(Ceqc)`:

```
137 >>> dir(Ceqc)↩
138 ['RT', 'T', 'VCS', '_Atributos__cadenaAlimentada', '_Atributos__cambios',
139 '_Atributos__entradas', '_Atributos__mol_inicial', '_Principal__iteraccion',
140 '__call__', '__doc__', '__init__', '__module__', '_atributos',
141 '_completaVectorFormula', '_creaMatrizFormula', '_especies',
142 '_especiesAlimentadas', '_especies_cadena', '_listaDeElementos',
143 '_molesDeElemento', '_moles_iniciales', '_presion', '_temperatura',
144 'archivoV_nPL','calculosDeN', 'completa_sistema', 'contempladas',
145 'dic_elementos', 'dic_especies','erroneas', 'gle', 'imprime_matriz_1',
146 'imprime_matriz_2', 'm', 'm_atomica', 'm_formula', 'm_nula', 'm_nulaT',
147 'maxIter', 'minDeltaG', 'mol_elemento', 'mol_iniciales', 'n', 'no_ceros',
148 'numElementos', 'numEspecies', 'numIteraciones', 'numRestricciones',
149 'ordenaV_n', 'ordenaV_nPL', 'p', 'presion', 'q', 'rango',
150 'reordenarArreglos','sistema', 'v_abundancia', 'v_deltaGj', 'v_delta_xi',
151 'v_elementos', 'v_especies', 'v_g', 'v_g0', 'v_n', 'v_nPL', 'vec_n0',
152 'vector']
153
```

No todos los miembros son accesibles al usuario final, en especial los que comienzan con dos guiones bajos. Los que comienzan con sólo un guión bajo no se recomienda usarlos. Otros como `calculosDeN` o `sistema` sirven sólo si se desea hacer alguna especie de control del comportamiento del programa, de ser así, es aconsejable consultar el **Apéndice A**.

La fase de cada especie se selecciona automáticamente. Esta fase es la que `Ceqc` considera como estándar, pero es posible que se equivoque o que se desee considerar otra fase no estándar en los cálculos, en tal caso debe responderse «*sí*» en la pregunta de la línea 38 y a continuación, aparecerá una pantalla preguntando cuál es el número de la especie que se desea cambiar, por ejemplo, si se teclea `1`, `Zn`, la pantalla quedará:

```
1   > Desea cambiar la fase de alguna de ellas >>  si↩
2       > Escriba el número de la especie >>  1↩
3
4       >       Fase           Intervalo de validez
5       >   ----------------------------------------
6       >   1.  SOLIDO         (298.0, 693.0)
7       >   2.  LIQUIDO        (693.0, 1180.0)
8       >   3.  GAS            (1180.0, 2000.0)
9
```

En ella se presenta las fases disponibles y el intervalo de temperatura, en kelvin, para las que están definidas. Después de la pantalla anterior se deberá escoger el número de alguna de las fases.

```
10      > Escoja una nueva fase:  2↩
11
```

La siguiente pregunta:

```
12  > Desea cambiar la fase de alguna otra >>
13
```

comienza nuevamente el ciclo.

De acuerdo a las fases elegidas para cada especies `Ceqc` determina a que fase del sistema debería pertenecer la especie. De igual forma que con la fase de cada especie, las fases del sistema pueden modificarse, esto se hace contestando «*sí*» a la pregunta de la línea 62 y entonces aparecerá una pantalla mostrando algunas opciones y se pedirá que elija una:

```
1       > ¿Desea cambiar la composición de alguna fase de sistema? >>  si↩
2       >   1. Cambiar especies a otra fase de sistema
3       >   2. Crear una nueva fase de sistema
4       >   3. Mostrar composición de fases de sistema
5       >   4. No cambiar fases de sistema
6
7       > Seleccione una opción >>
8
```

De elegirse la opción `1` se mostrarán las fases del sistema y se pedirá elegir una de ellas para almacenar las especies a cambiar:

```
9       > Fases de sistema disponibles:
10      >   1.  liquido
11      >   2.  gas
12      >   3.  solido_0
13
14      > Seleccione la fase de sistema donde estarán las especies:  1↩
15
```

Por ejemplo, si se desea considerar al `ZnO` como dentro de la fase `liquido`, debe seleccionarse la opción `1`, `liquido`. Al hacerlo, `Ceqc` responderá solicitándole al usuario la especie(s) que desea incluir en la fase `liquido`, en este caso `ZnO`.

```
16      > Seleccione la fase de sistema donde estarán las especies:  1↩
17      > Escriba las especies que se adicionarán a la fase de sistema
18                                                          "liquido": ZnO↩
19      > El sistema es:
20      ============================================================
21
22      > Fases de sistema:
23
24
25      >    liquido:
26      >   ----------------------------------
27      >       Zn        Líquido   (LIQUIDO)
28      >       ZnO       Sólido    (ZINCITA)
29
30      >    gas:
31      >   ----------------------------------
32      >        H2        Gas       (GAS)
33      >        CH4       Gas       (GAS)
34      >        H2O       Gas       (VAPOR)
35      >        CO2       Gas       (GAS)
36      >        CO        Gas       (GAS)
37
38
39      > ¿Desea cambiar la composición de alguna fase de sistema? >>
40
```

Finalmente, el programa termina preguntando si desea cambiar alguna otra fase y el proceso comienza nuevamente. Si en lugar de la opción `1` se hubiese elegido la `2` se le pediría dar el nombre de la nueva fase y las especies que contendría dicha fase.

```
1   > Seleccione una opción >>  2↩
2       > Escriba el nombre de la nueva fase de sistema: solido_2↩
3       > Escriba las especies que contendrá la fase de sistema "solido_2":
4                                                                    CO2↩
5       > El sistema es:
6       ============================================================
7
8       > Fases de sistema:
9
10
11      >     liquido:
12      >     ----------------------------------------
13      >         Zn        Líquido   (LIQUIDO)
14
15      >     gas:
16      >     ----------------------------------------
17      >         H2        Gas       (GAS)
18      >         CH4       Gas       (GAS)
19      >         H2O       Gas       (VAPOR)
20      >         CO        Gas       (GAS)
21
22      >     solido_0:
23      >     ----------------------------------------
24      >         ZnO       Sólido    (ZINCITA)
25
26      >     solido_2:
27      >     ----------------------------------------
28      >         CO2       Gas       (GAS)
29
30
31      > ¿Desea cambiar la composición de alguna fase de sistema? >>
32
```

En este ejemplo se ha creado la nueva fase llamada `solido_2`, compuesta por un único elemento, $CO_{2(g)}$. La opción `3` simplemente muestra la pantalla anterior. La opción `4` termina esta subrutina y continúa con el programa.

6.3 Descripción del programa `Ceqc`

El programa `Ceqc` puede ser discutido bajo cuatro categorías, éstas son:

1. Base de datos termodinámicos.
2. Entrada de datos de los reactantes.
3. Cálculos termodinámicos *estándar*.
4. Cálculos termodinámicos *no estándar*.

La Base de datos termodinámicos incluye la información termodinámica de «todas» las especies que pueden ser contempladas en los cálculos. La entrada de datos de los reactantes es toda la información necesarias que deberá ser proporcionada por el usuario del programa para comenzar su ejecución. Los cálculos termodinámicos *estándar* son aquellos que se realizan de forma *rutinaria o estándar* tanto para sistemas ideales como no ideales. El cálculo termodinámico *no estándar* es una extensión del cálculo estándar, esta categoría incluye todos aquellos cálculos que son incorporados al programa, por parte del usuario, en forma de módulos. Esta última categoría es necesaria cuando se desea hacer una adición, extensión, modificación o remplazo de algunos o todos los modelos de solución (química) proporcionados por `Ceqc`.

Cada una de estas categorías determinan el desarrollo operativo del programa `Ceqc`, y serán discutidas a detalle en las siguientes secciones.

6.4 Base de datos termodinámicos, BT

La información termodinámica incluida en la BT del programa de cómputo desarrollado, abarca los datos de 1881 especies puras. La BT se encuentra almacenada en el archivo editable `base_mia.py` y en el archivo compilado `base_mia.pyc`. Cada vez que el archivo editable (extensión `.py`) es modificado, de manera automática, se actualiza el archivo compilado (extensión `.pyc`). El objetivo de compilar el archivo de la BT es aumentar la velocidad de acceso a la misma.

Es importante señalar que `Python` distingue entre mayúsculas y minúsculas, por lo tanto, no es lo mismo escribir `Co` que `CO` o `co`, en el primer caso se está haciendo referencia al elemento cobalto, en el segundo al compuesto monóxido de carbono y en el tercero a ninguna especie. Lo datos de la BT están escritos siguiendo la nomenclatura habitual, por ejemplo, `Fe2O3` para Fe_2O_3, `Fe2(SO4)3` para $Fe_2(SO_4)_3$ y `(CaO)(Al2O3)2` para $(CaO)(Al_2O_3)_2$.

La organización de la BT está basada en la estructura de «diccionario» de `Python`.El nombre del diccionario es `Base`, y de él se desprenden cada una de la entradas o especies químicas, por ejemplo:

```
1   Base={
2       ⋮
3       "Fe":{...}
4       "LiBr":{...}
5       ⋮
6       "Cu":{...}
7       "FeO":{...}
8       "Ca2CO3":{...}
9       ⋮
10      }
11
```

Cada entrada puede ser a su vez un diccionario con datos termodinámicos. Para el Cu, por ejemplo, la entrada contiene los siguientes datos:

```
1   "Cu":{
2       "nombre":"Cobre",
3       "linea2":{"fases": 3},
4       "fase1":{"a": 5.94, "rango1": 298.0, "c": -0.332, "b": 0.905,
5           "e": 0.0, "d": 0.0, "H298": 0.0,"rango2": 1356.6,
6           "densidad": 8.92, "nombre": "SOLIDO", "T_trans": 1356.6,
7           "S298": 7.913, "H_trans": 0.0, "fase": 101},
8       "fase2":{"a": 7.5, "rango1": 1356.6, "c": 0.0, "b": 0.0,
9           "e": 0.0, "d": 0.0, "H298": 2.2245, "rango2": 2848.0,
10          "densidad": 8.92, "nombre": "LIQUIDO", "T_trans": 2848.0,
11          "S298": 8.6663, "H_trans": 3.17, "fase": 801},
12      "fase3":{"a": 5.84, "rango1": 2848.0, "c": 0.0, "b": 0.0,
13          "e": 0.0, "d": 0.0, "H298": 79.2, "rango2": 3400.0,
14          "densidad": 0.0, "nombre": "GAS", "T_trans": 0.0,
15          "S298": 37.954, "H_trans": 72.743, "fase": 901},
16      "texto":"BARIN ET AL 1977"}
17
```

En la línea 2 se encuentra almacenado el nombre de la especie, en este caso `Cobre`. La línea 3 indica el número de fases para las que se tiene información, `linea2:fases:3`. En la línea 4 , 8 y 12 se encuentra la entrada a la información para estas tres fases. El orden en que se almacena la información no es relevante. Así, desde la línea 4 y hasta la línea 7, se encuentran los parámetros `a`, `b`, `c` y `d`, estos son los coeficientes obtenidos por el método de mínimos cuadrados de la función de capacidad calorífica, expresada en $\text{cal} \cdot \text{K}^{-1}$:

$$C_P(T) = a + bT \times 10^{-3} + cT^{-2} \times 10^{5} + dT^{2} \times 10^{-6} \tag{6.1}$$

En la línea 4 y 5 se encuentran dos claves llamadas `rango1` y `rango2` éstas se refieren al rango de temperatura, en kelvin, para la que está definida y es válida la `fase1`. Otras claves de esta fase son la `H298` y la `S298` que representan la entalpía y entropía a 298.15 K, expresadas en kcal y cal·K^{-1}, de la especie para la `fase1`.

La clave `densidad` de cada fase, almacena la densidad promedio de la fase y está expresada en g·cm^{-3}. La clave `nombre` almacena el nombre de la fase, en ocasiones se hace referencia a la fase con un nombre genérico como: `sólido 2`, `líquido 1`, o `gas`.

La clave `H_trans` representa la entalpía de transformación «a» la fase actual. `T_trans` es la temperatura a la cuál la fase debería transformar a otra más estable. Es importante resaltar que para una fase determinada, la temperatura de trasformación no corresponde a la entalpía de transformación. Por ejemplo, en la entrada de la fase `SOLIDO` se encuentra `T_trans=1356.6` y `H_trans=0.0`, esto indica que la transformación `SOLIDO` → `LIQUIDO` sucede a 1356.6 K pero la `H_trans` anterior no corresponde a esta transformación, la `H_trans` de esta reacción se encuentra en la fase `LIQUIDO` y es igual a 3.17 kcal. Quizá ésta última explicación quede más clara si se revisan las **tablas 6.1** y **6.2** y se comparan con la información para el cobre en la BT.

Tabla 6.1 Información termodinámica del cobre contenida en la base de datos. Parte 1.

Fase	**Nombre**	**Intervalo de T**	ΔH_{Trans}(Kcal)
`fase1`	`SOLIDO`	298.0 – 1356.6	3.17 a 1356.6 K
`fase2`	`LIQUIDO`	1356.6 – 2848.0	72.743 a 2848.0 K
`fase3`	`GAS`	2848.0 – 3400.0	

Tabla 6.2 Información termodinámica del cobre contenida en la base de datos. Parte 2.

Fase	H_{298}(Kcal)	S_{298}(cal/K)	**Densidad**(g · cm^{-3})	a	b	c	d
`fase1`	0.000	7.926	8.920	4.925	2.051	0.316	0.000
`fase2`	1.923	8.147	8.925	7.850	0.000	0.000	0.000
`fase3`	38.224	38.224		5.840	0.000	0.000	0.000

La entrada `fase` indica el tipo de fase, un número de 100 a 800 representa un sólido, de 800 a 900 a un líquido, de 900 a 990 un gas y de 990 a 1000 una fase acuosa.

Finalmente, la clave `texto` en la línea 16 muestra la fuente de dónde se obtuvo la información del compuesto. La BT se compone de 1881 especies (entradas) similares a la descrita, si se desea actualizar o extender esta BT deberá respetarse el formato de diccionario con las claves escritas de forma correcta y empleando la unidades mencionadas. El siguiente cuadro esquematiza la forma general de la base.

```
1   Base={
        ⋮
2       "especie":{
3           "nombre":"cadena de caracteres"
4           "linea2":{
5               "fases":número
6               }
7           "fase1":{
8               "a":número
9               "b":número
10              "c":número
11              "d":número
12              "rango1":número
13              "rango2":número
14              "H298":número
15              "S298":número
16              "H_trans":número
17              "T_trans":número
18              "nombre":"cadena de caracteres"
19              "densidad":número
20              "fase":número
21              }
            ⋮
22          }
23      "texto":"cadena de caracteres"
        ⋮
24      }
25
```

6.5 Entrada de datos de los reactantes

Los datos alimentados por parte del usuario son parte fundamental de la ejecución de cualquier programa, ellos permiten especificar que lo que se desea y cómo se desea que se haga. `Ceqc`, al igual que muchos otros programas, solicita para comenzar su ejecución algunos datos «mínimos», estos son:

1. especies participantes,
2. temperatura,
3. presión,
4. moles iniciales,
5. selección de fase y
6. selección del modelo de la fase.

Cuando `Ceqc` solicita el nombre de las especies participantes estos deberán escribirse separados por espacios, pues es la forma en que `Ceqc` reconoce una especie como diferente. No todos los símbolos son válidos para `Ceqc`, cuando se escribe una fórmula química. `Ceqc` es capaz de reconocer los símbolos no válidos, descartarlos y obtener segmentos «sintácticamente» válidos de la fórmula mal escrita. De esta forma, son válidos los siguiente símbolos.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
1234567890[]()+-
```

`Ceqc` todavía hace más discriminaciones en cuanto a orden y sintaxis. Por ejemplo, si en la entrada de especies se escribe algo como: `c#CACO3`. `Ceqc` respondería lo siguiente:

```
1   > Escriba las especies del sistema >>  c#CACO3↩
2
3       > Existe un error en la escritura de la fórmula:  c#CACO3
4       > Detecté algunos segmentos sintácticamente válidos de la entrada estos
5           son:
6
7           1.   CCO3
8
9       > Si desea que alguno de estos segmentos lo considere para los cálculos,
10      > por favor escriba los números correspondientes (de lo contrario
11          presione "enter") >> ↩
12
```

Aquí, el símbolo # es un carácter inválido y es descartado inmediatamente, tanto `c` como `A` son caracteres válidos pero no corresponden a algún elemento ni pueden pertenecer a uno, también son descartados. De toda la fórmula escrita se recupera un segmento sintácticamente válido aunque quizá, como en este caso, semánticamente inválido, o bien no presente en la base de datos.

Cuando se teclea la fórmula química de un reactante, `Ceqc` es capaz de reconocer los elementos químicos presentes y sus cantidades estequiométricas en cada fórmula. Con esta información el programa determina los elementos presentes en el sistema, crea los vectores fórmula de cada especie con respecto al sistema, determina la matriz fórmula, la matriz estequiométrica, que a su vez ésta es la columna vertebral del algoritmo empleado y realiza el balance de masa por elementos. Esta característica de `Ceqc` facilita el empleo del programa al no solicitar al usuario información adicional que se encuentra implícita en la información ya dada.

La temperatura, presión y moles iniciales solicitadas por el programa deben escribirse en Kelvin, atmósferas y número de moles. En la entrada de cada cantidad puede escribirse una expresión en lugar de un cantidad absoluta, por ejemplo, si la temperatura del sistema es de 1200° F puede convertirse a Kelvin directamente escribiendo: `273.15 + 5./9.*(1200-32)`. Si la masa del compuesto NaCl es de 20g, también puede convertirse a moles escribiendo directamente: `20/(22.99+35.45)`.

La selección de fase y modelo de la fase del sistema no permite el ingreso de una expresión, pues las opciones son directas.

6.6 Cálculos termodinámicos estándar

Básicamente los cálculos termodinámicos estándar consisten en extraer la información termodinámica desde la base de datos y manipularla para obtener los diferentes parámetros termodinámicos de cada especie a una temperatura dada.

De la base de datos se extraen los parámetros de la ecuación de C_P, el valor de ΔH_{298}^{fase} y S_{298}^{fase} para una fase determinada. Los cálculos termodinámicos realizados en esta etapa son los siguiente, todos son función de la temperatura:

$$C_P^{fase} = a + bT \times 10^{-3} + cT^{-2} \times 10^5 + dT^2 \times 10^{-6} \tag{6.2}$$

$$\Delta H_T^{fase} = \Delta H_{298}^{fase} + \int_{298}^{T} C_P \, dT \tag{6.3}$$

$$\Delta S_T^{fase} = S_{298}^{fase} + \int_{298}^{T} \frac{C_P}{T} \, dT \tag{6.4}$$

$$\Delta G_T^{fase} = \Delta H_T^{fase} - T\Delta S_T^{fase} \tag{6.5}$$

6.7 Cálculos termodinámicos no estándar

Para resolver las ecuaciones que expresan las condiciones de equilibrio, conviene tener una expresión apropiada para el potencial químico de cada especie que las relacione con la composición, además de hacerlo con la temperatura y la presión. El potencial químico para una especie en solución ideal, por ejemplo,

$$g_i(T, P, x_i) = g_i^{\circ}(T, P) + RT \ln x_i \tag{6.6}$$

depende sólo de su propia composición. Habrá otras expresiones de potencial químico, que se consideren no ideales, en las que dependa su valor de una fracción molar de una sola especie, por ejemplo, del modelo de parámetros de interacción, soluciones regulares, etcétera. Este tipo de definiciones son las que se realizan en los cálculos termodinámicos no estándar. Para explicar en que consisten materialmente estos cálculos es necesario explicar el módulo `modelo.py` del programa `Ceqc`.

En este módulo deben escribirse las expresiones de potencial químico que definen los modelos de solución. En primer lugar, al principio del módulo se define una variable llamada `__all__` en ella deben escribirse todos los nombres de modelos definidos dentro del mismo módulo. Si llegase a definirse una expresión de potencial químico y su nombre no es colocado en la variable `__all__`, la definición sería invisible para el resto del programa fuera del módulo. La variable `__all__` en el módulo `modelo.py` en el momento de escribir esta línea es:

```
1   __all__ = ['g_ideal_gas', 'g_ideal_liquido', 'g_ideal_solucion_solida',
2              'g_ideal_solido', 'parametros_de_interaccion' ]
3
```

Pues solamente están definidas las expresiones para cinco modelos de solución.

Por defecto, este módulo importa tres funciones matemáticas para realizar algunos de los cálculos más comunes. Por ejemplo, en `modelo.py` está escrito:

```
4   from Numeric import log, array, dot
5   from p_interaccion import Actividades
6   from constantes import R
7
```

Esto importa de la librería `Numeric` la función `log` para logaritmos naturales, `array` para definir arreglos y `dot` para realizar multiplicaciones entre matrices. Entre otras, pueden importarse la siguientes funciones de `Numeric` :

1. `absolute`
2. `arccos`
3. `arccosh`
4. `arcsin`
5. `arcsinh`
6. `arctan`
7. `ceil`
8. `cos`
9. `cosh`
10. `e`
11. `exp`
12. `fabs`
13. `floor`
14. `hypot`
15. `log10`
16. `maximum`
17. `pi`
18. `power`
19. `sin`
20. `sinh`
21. `sqrt`
22. `tan`
23. `minimum`
24. `tanh`.

Después de este preámbulo comienzan la definiciones. Por ejemplo, el modelo de gas ideal es:

$$g_i = g_i^\circ + RT \ln Px_i \tag{6.7}$$

donde g_i° es calculada con la **ecuación 6.5** de la sección anterior. Para definir este modelo deben seguirse algunas reglas generales. Todos las definiciones comenzarán con la palabra clave `def`, seguida de un espacio se coloca el nombre del modelo, sin espacios. Los argumentos deben colocarse entre paréntesis después del nombre del modelo. Los «únicos» argumentos que pueden llevar las definiciones son:

```
(sistema, T, v_especies, presion, fase)
```

por el momento es toda la información del programa disponible. Debe seguirse siempre este orden. Todos estos parámetros son definidos por el programa. El primer parámetro `sistema`, es en realidad un diccionario con todos los parámetros del sistema y a ellos se puede acceder por medio de `nombre_del_método` en `sistema.fases.nombre_del_método`, donde `nombre_del_método` puede ser:

1. `especie_en_fase_especifica_multicomponente(fase,especie)`: regresa `True` si la `especie` se encuentra en la fase `fase` multicomponente y `False` en caso contrario.

2. `especie_en_fase_multicomponente(especie)`: regresa `True` si la especie `especie` se encuentra en una fase multicomponente y `False` en caso contrario.

3. `especies()`: regresa una lista con el nombre de las especies del sistema.

4. `especies_de_la_fase(fase)`: regresa una lista con el nombre de las especies en la fase `fase`.

5. `fase_de_la_especie(especie)`: regresa el nombre de la fase donde se encuentra la especie `especie`.

6. `fase_multicomponente(fase)`: regresa `True` si la fase `fase` es multicomponente y `False` en caso contrario.

7. `fases()`: regresa una lista con el nombre de las fases del sistema.

8. `moles_de_la_especie(especie, mol)`: regresa el número de moles de la especie `especie`.

9. `moles_de_la_fase(fase, mol)`: regresa el número de moles de la fase `fase`.

10. `moles_de_la_fase_donde_la_especie(especie, mol)`: regresa el número de moles de la fase `fase` donde se encuentra la especie `especie`.

11. `moles_totales(mol)`: regresa las moles totales del sistema.

12. `actualiza(especie,valor,nombre_propiedad)`: reasigna valor de la propiedad `nombre_propiedad` para la especie `especie`.

Puede accederse a otra información del sistema por medio de los métodos 8, 9 y 10. Por ejemplo, para acceder al valor de `g0` de una determinada especie deberá escribirse:

```
moles_de_la_especie(especie, 'g0')
```

además de g° espocible utilizar `g`, `vector` y `termo`. La opción `termo` es especial pues por medio de ella se accede a los métodos y constantes obtenidos en los cálculos termodinámicos estándar. Por ejemplo, para acceder al valor de la constantes `a` para la expresión de C_P de la especie `'A'` tiene que teclearse

```
moles_de_la_especie('A', 'termo').a
```

los métodos de `termo` son los definidos en el módulo `energia.py`.

En la definición del modelo, seguido del parámetro `sistema`, deberá encontrarse `T`, la temperatura, `v_especies`, vector de especies, `presion`, presión y `fase`, nombre de la fase. Finalmente, el modelo de gas ideal quedará, entonces:

```
8   def g_ideal_gas(sistema, T, v_especies, presion, fase):
9           mol_fase = sistema.fases.moles_de_la_fase(fase)
10          for especie in sistema.fases.especies_de_la_fase(fase):
11                  x_i = sistema.fases.moles_de_la_especie(especie)/mol_fase
12                  g0 = sistema.fases.moles_de_la_especie(especie, 'g0')
13                  g =  g0 + R*T*log(abs(presion*x_i))
14                  sistema.fases.actualiza(especie, g, 'g')
15
```

este modelo contempla todas las especies dentro de una fase. Por supuesto es posible utilizar todas las características del lenguaje `Python` en las definiciones de un modelo. En la línea 13 está realmente la definición de la **ecuación 6.7**. La sentencia en la línea 14 es importante utilizarla ya que es ésta la que actualiza todo el sistema y de esta actualización es de donde se realizan todos los cálculos y muestran los resultados.

7 Forma modular del programa Ceqc

Con la finalidad de facilitar la adición o borrado de partes del programa de equilibrio químico, éste se constituye de 23 módulos. Cada uno de estos módulos está relacionado con la entrada general, procesamiento interno de la entrada o cálculos de equilibrio químico. En la **figura 7.2** se esquematiza la interrelación de los módulos, a pesar de lo aparentemente intrincado de este diagrama, estos módulos pueden clasificarse dentro de una de cuatro categorías:

1. módulos de cálculos generales,
2. módulos de procesamiento de entrada,
3. módulos de cálculos termodinámicos,
4. módulos de equilibrio químico.

En principio, los módulos que se encuentran en recuadros sombreados, son módulos de la librería de `Python`, todos ellos entran en la categoría «módulos de cálculos generales», la clasificación de todos los demás modulos se describe en la **figura 7.1** y algunos detalles de los módulos individuales se dan en la **sección 7.1**.

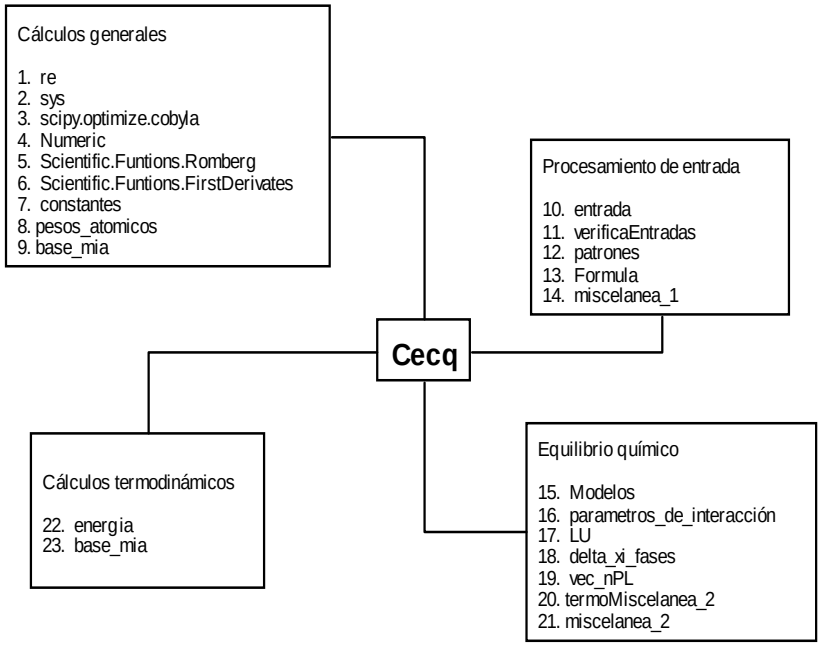

Figura 7.1 Clasificación de los módulos de `Ceqc`.

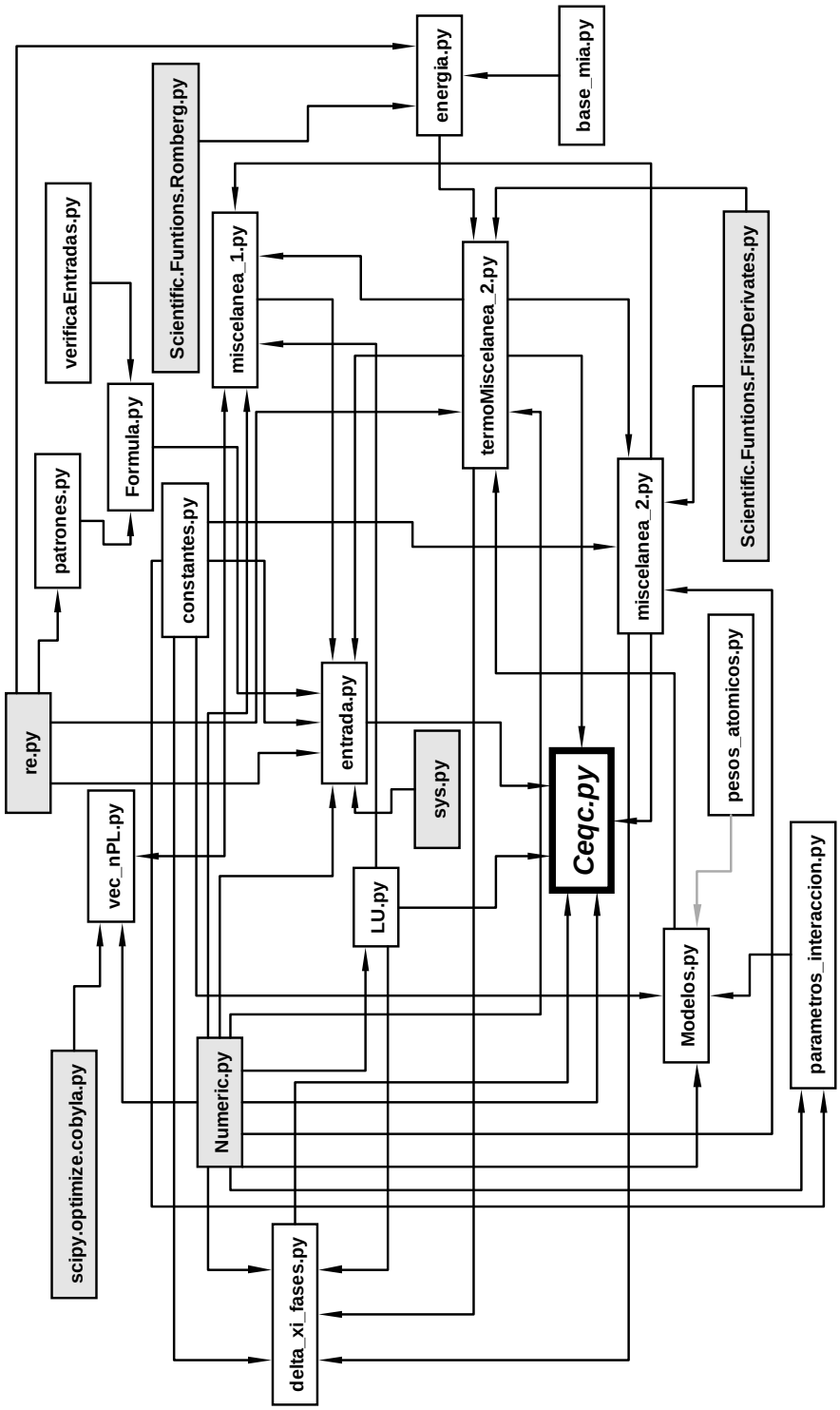

Figura 7.2 Relación entre los módulos de `Ceqc`.

7.1 Descripción de Módulos

El programa consiste de un módulo principal, 16 módulos auxiliares y una base de datos. En las siguientes secciones se da una descripción de la función de cada uno de ellos.

7.2 Módulo principal, `Ceqc`

Debido a que `Python` es un programa «orientado a objetos» es frecuente encontrar en el código fuente de `Ceqc` definiciones de `clase`, éstas son tipos de datos donde se agrupan acciones o subrutinas que poseen características o persiguen un objetivo común.

En el módulo principal llamado, precisamente, `Ceqc` se define la clase `Principal`, ésta es la encargada del comienzo y término del programa `Ceqc`.

7.3 Módulo `constantes`

En este módulo deben definirse las constantes físicas y químicas que se ocuparán en los cálculos, en este momento la única constante definida es la constante R de los gases ideales, con el valor de $8.314\ \mathrm{kJ} \cdot \mathrm{kg}^{-1} \cdot \mathrm{mol}^{-1} \cdot \mathrm{K}^{-1}$.

7.4 Módulo `energia`

Dentro de este módulo se define la «`clase`» `Energia`, ésta es la encargada de hacer los cálculos de energía libre de Gibbs estándar, entalpía estándar y entropía estándar de cada uno de las especies a la temperatura del sistema y considerando un comportamiento ideal.

Los cálculos que se realizan en este módulo permitirían, alternativamente, obtener las funciones termodinámicas de especies puras a una temperatura dada de forma independiente a cualquier sistema: si se ejecuta únicamente este módulo, se obtendrá la siguiente pantalla, en ella se pide teclear el nombre y la temperatura de una especie. Por ejemplo, al teclear Cu a 1200 K, se presentan los cálculos $\Delta H_{1200\ \mathrm{K}}$, $\Delta S_{1200\ \mathrm{K}}$ y $\Delta G_{1200\ \mathrm{K}}$, también algunos datos contenidos en la base de datos como $\Delta H_{298\ \mathrm{K}}$, $S_{298\ \mathrm{K}}$, nombre de la fase, parámetros a, b, c y d de la expresión de C_P, todos ellos para la fase que energía considera más estable a la temperatura dada.

```
Especie: Cu ↩
Temperatura: 1200 ↩

    especie         : Cu
    nombre          : Cobre
    temperatura     : 1200.0         K

    T_trans         : 1356.6         K
    H_trans         : 0              J
    densidad        : 8.92           g/cm^3
    entalpia        : 24625.2        J
    entropía        : 70.4093        J
    e. libre        : -59866         J
    DeltaH298       : 0              J
    S298            : 33.108         J
    clave_fase      : 101
    nombre_fase     : SOLIDO

    expresión Cp    : a + 1E-3*b*T + 1E5*.c*T**-2 + 1E-6*d*T**2
    a, b, c, d      : 5.94     , 0.905    , -0.332   , 0
================================================================
¿Desea cambiar la fase estándar?: Si↩

```

Después de presentar estos datos el programa pregunta si desea cambiar la fase estándar del Cu a 1200 K, al responderse «sí» a esta pregunta, aparece una lista de las fases disponibles del cobre y se pide elegir una de ellas:

```
    Cambia la fase estándar
    ------------------------------

    >       Fase           Intervalo de validez
    >   ----------------------------
    >   1.  SOLIDO         (298.0, 1357.0)
    >   2.  LIQUIDO        (1357.0, 2848.0)
    >   3.  GAS            (2848.0, 3400.0)

    > Escoja una nueva fase: 2

```

El Cu es sólido a 1200 K, como se muestra en la primera pantalla de este ejemplo, al escoger la fase líquido en esta segunda pantalla, se está pidiendo calcular las propiedades termodinámicas del cobre líquido a 1200 K, y éstas son:

```
35      especie         : Cu
36      nombre          : Cobre
37      temperatura     : 1200.0         K
38
39      T_trans         : 2848           K
40      H_trans         : 13.2633        J
41      densidad        : 8.92           g/cm^3
42      entalpia        : 37612          J
43      entropía        : 79.9718        J
44      e. libre        : -58354.1       J
45      DeltaH298       : 9307.28        J
46      S298            : 36.26          J
47      clave_fase      : 801
48      nombre_fase     : LIQUIDO
49
50      expresión Cp    : a + 1E-3*b*T + 1E5*.c*T**-2 + 1E-6*d*T**2
51      a, b, c, d      : 7.5      , 0        , 0        , 0
52  ================================================================
53
54  ¿Desea cambiar la fase estándar?: No↩
55
```

Finalmente se pregunta, otra vez, por el cambio de fase estándar, al responder «no» el ciclo comienza nuevamente.

7.5 Módulo `patrones`

En la entrada de datos, las fórmulas se escriben siguiendo ciertas reglas de escritura (nomenclatura química), este hecho es aprovechado por `Ceqc` para determinar el número y tipo de elementos presentes en una especie química. En el módulo `patrones` se definen patrones de búsqueda especiales[10] que identifican la sintaxis y tipo de caracteres presentes en la especie química. Por ejemplo, el patrón de búsqueda para una fórmula se definió en este trabajo como:

[10] Algunos lenguajes de programación como `Perl`, `Python` o `Matlab`, emplean lo que se denomina *expresiones regulares*, éstas son un tipo de código que permite la búsqueda de patrones dentro de una cadena de texto.

```
pFormula = re.compile('(([A-Z][a-z]{0,1}[/d]*)+|([(]([A-Z][a-z]{0,1}[/d]*)+
                       [)][/d]*)+)+([[][/d]*([+]|[-])[]])?')
```

Quizá, esta nomenclatura quede más clara en la notación Backus–Naur, en esta notación, las reglas de sintaxis son usadas para expresar la manera en la cual las cantidades son construidas a partir de unidades simples. La sintaxis del patrón anterior es:

⟨fórmula primaria⟩ ⟶ ⟨elemento⟩⟨posible dígito⟩
⟨fórmula secundaria⟩ ⟶ ⟨fórmula primaria⟩
`|`⟨fórmula primaria⟩⟨fórmula secundaria⟩
`|`⟨fórmula secundaria⟩`(`⟨fórmula secundaria⟩`)`⟨posible dígito⟩
`|(`⟨fórmula secundaria⟩`)`⟨posible dígito⟩⟨fórmula secundaria⟩
`|(`⟨fórmula secundaria⟩`)`⟨posible dígito⟩`(`⟨fórmula secundaria⟩`)`⟨posible dígito⟩
⟨fórmula iónica⟩ ⟶ ⟨fórmula secundaria⟩`[`⟨ion⟩`]`
⟨elemento⟩ ⟶ ⟨mayúscula⟩`|`⟨mayúscula⟩⟨minúscula⟩
⟨posible dígito⟩ ⟶ ⟨nada⟩`|`⟨cadena de dígitos⟩
⟨ion⟩ ⟶ ⟨signo⟩`|`⟨signo⟩⟨cadena de dígitos⟩
⟨cadena de dígitos⟩ ⟶ ⟨dígito decimal⟩`|`⟨dígito decimal⟩⟨cadena de dígitos⟩
⟨dígito decimal⟩ ⟶ `0|1|2|3|4|5|6|7|8|9`
⟨mayúscula⟩ ⟶ `A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z`
⟨minúscula⟩ ⟶ `a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z`
⟨signo⟩ ⟶ `+|-`
⟨nada⟩ ⟶

Por ejemplo, la fórmula H_2SO_4 puede descomponerse en unidades más simples de la siguiente manera:

`H2SO4` ⟶ ⟨fórmula secundaria⟩
`H2·SO4` ⟶ ⟨fórmula primaria⟩⟨fórmula secundaria⟩
`H·2·S·O4` ⟶ ⟨elemento⟩⟨posible dígito⟩⟨fórmula primaria⟩⟨fórmula primaria⟩
`H·2·S·O4` ⟶ ⟨mayúscula⟩⟨cadena de dígitos⟩⟨elemento⟩⟨posible dígito⟩⟨elemento⟩⟨posible dígito⟩
`H·2·S·O·4` ⟶ `H` ⟨dígito decimal⟩⟨mayúscula⟩⟨nada⟩⟨mayúscula⟩⟨cadena de dígitos⟩
`H·2·S·O·4` ⟶ `H 2 S O` ⟨dígito decimal⟩
`H·2·S·O·4` ⟶ `H 2 S O 4`

Esto es básicamente lo que se hace al emplear expresiones regulares. Puede observarse que el patrón también reconoce especies iónicas escritas de la forma ⟨fórmula secundaria⟩`[`⟨ion⟩`]`, es decir, `HSO4[+1]` es una expresión válida. Pero, aunque en la base de datos y en los patrones se dé la posibilidad de especies iónicas, el algoritmo de `Ceqc` tendría que modificarse ligeramente para contemplar este tipo de especies.

7.6 Módulo `Formula`

Utilizando el módulo `patrones`, este módulo es el encargado de determinar la sintaxis y validez de las fórmulas escritas en la entrada. Una vez que ha reconocido los componentes de la fórmula química para cada especie, determina el tipo y cantidad de elementos en el sistema, y de aquí el vector fórmula para cada especie. Al ejecutar únicamente este módulo se mostrará en pantalla:

```
1   >>>
2   Cu4SO4(OH)6(H2O)
3   {'H': 8, 'S': 1, 'O': 11, 'Cu': 4}
4
5   fórmula:
6
```

donde se muestra la descomposición de la fórmula $Cu_4SO_4(OH)_6(H_2O)$, Lagnita, y se invita a probar para otra fórmula.

7.7 Módulo `miscelanea_1` y `vec_nPL`

Este módulo, escribe e importa el módulo `vec_nPL`. Aquí se resuelve el problema de programación lineal:

$$\text{mín!} = \underline{\mathbf{n}}^{\mathrm{T}}\underline{\mathbf{g}}^{\circ} \qquad \text{sujeto a} \qquad \mathbf{A}\underline{\mathbf{n}} = \underline{\mathbf{b}},\ n_i \geq 0 \tag{7.1}$$

donde $\underline{\mathbf{g}}^{\circ}$ es el vector potencial estándar de las especies del sistema, $\underline{\mathbf{n}}^{\mathrm{T}}$ es el vector de abundancia de especies transpuesto, $\mathbf{A}$ es la matriz fórmula, $\underline{\mathbf{b}}$ es el vector de abundancia de elementos. Específicamente, de `vec_nPL` se obtiene $\underline{\mathbf{v}}_{\mathrm{nPL}}$ que se incorpora, indirectamente, al algoritmo VCS como las moles iniciales, $\underline{\mathbf{v}}_{n^{\circ}}$. El algoritmo VCS requiere que las especies se

encuentran ordenadas de acuerdo al procedimiento de programación lineal, PL, descrito en la **sección 5.4**, este ordenamiento también se lleva acabo aquí y de ser necesario se ejecuta la orden de recalcular la matriz estequiométrica.

7.8 Módulo `miscelanea_2`

En este módulo se calcula el parámetro de etapa de acuerdo al procedimiento descrito en la **sección 4.7**, así como el valor de $\underline{\mathbf{x}}_{n+1}$ en cada iteración, **sección 4.5**.

7.9 Módulo `entrada`

Este módulo solicita y administra los datos alimentados por el usuario. Es el encargado de ejecutar el módulo `Formula`, `energia`, crear la matriz formula, y ordenar la creación del diccionario `sistema`, empleado para definir los diferentes modelos de solución.

7.10 Módulo `LU`

Este módulo es esencialmente un módulo que sirve para resolver ecuaciones lineales mediante la descomposición LU. Sólo que aquí pueden resolverse varios «sistemas» de ecuaciones lineales al mismo tiempo en los que varía únicamente las constantes independientes, es decir, un sistema lineal tiene la forma:

$$\mathbf{A}\underline{\mathbf{v}} = \underline{\mathbf{c}} \tag{7.2}$$

donde $\mathbf{A}$ es la matriz de coeficiente constantes, $\underline{\mathbf{v}}$ es vector de variables y $\underline{\mathbf{c}}$ el vector de constantes independientes, pues en este módulo puede resolverse:

$$\mathbf{A}\underline{\mathbf{v}} = \mathbf{C} \tag{7.3}$$

donde $\mathbf{C}$ es una matriz de constantes independientes. Ver **secciones 1.9**, **1.10**, **2.4**, y **2.5**

7.11 Módulo `delta_xi_fases`

En este módulo se define una única `clase` llamada `VCS`, ésta es la encargada de computar el cálculo definido por las **ecuaciones 5.60** y **5.61**, para un sistema mono y multifásico.

Es decir, en este módulo se determina el vector avance de reacción y en consecuencia la dirección de búsqueda, característica de el algoritmo VCS.

7.12 Módulo `termoMiscelanea_2`

En este módulo se hace la definición de la función de energía libre para cada especie del sistema de acuerdo a los modelos de solución definidos en el módulo `modelos`. Además, se definen tres `clases`: `Clasifica_especies_en_fase`, `Fase` y `Fases`. La primera determina el tipo de fase para cada especie (la fase por defecto) de acuerdo a los datos encontrados en la bases de datos y administra los cambios de fase, para cada especie, solicitados por el usuario. La `clase Fase` es la definición mínima de la fase de sistema, por ejemplo, una fase de sistema compuesta de O_2 y H_2. Esta clase administra detalles como: si la fase es multicomponente, la cantidad de moles de la fase y el modelo de solución. La `clase Fases` es, en sí, la administradora del sistema, en ella están definidas la opciones «necesarias» para los «cálculos termodinámicos no estándar», **sección 6.7**.

7.13 Módulo `Modelos`

El contenido de este módulo se describe en la **sección 6.7**. Sólo resta decir que están definidos cuatro modelos de solución ideal y uno no ideal:

1. Gases ideales
2. Líquidos ideales
3. Sólidos ideales
4. Sólido ideal[11]
5. Parámetros interacción

7.14 Módulo `parametros_interaccion`

La definición del modelo de parámetros interacción dentro del módulo `modelos` es la siguiente:

$$g_i(\underline{\mathbf{n}}, T, P) = g_i^{\circ}(\underline{\mathbf{n}}, T, P) + RT \log a_i(\underline{\mathbf{n}}, T, P) \tag{7.4}$$

En esta ecuación, la actividad, a_i, es la función calculada mediante el modelo de parametros de interacción en la misma forma que se describe en la **section 3.13**. Este módulo lo compone

[11] Igual que el modelo de solución «sólidos ideales», excepto que éste contempla una sola especie.

una `clase` llamada actividad que realiza todos los cálculos. Para emplear esta `clase` es necesario especificar el nombre del solvente, la temperatura, y dos diccionario, ambos con los parámetros a y b del coeficiente de actividad henriano

$$RT\ln\gamma_i^\circ = a + bT.$$

y de la expresión para cada uno de los parámetros de interacción, $\epsilon_{i,j}$.

$$RT\epsilon_{i,j} = a + bT.$$

A manera de ejemplo, en el módulo se definen estos dos diccionarios para el sistema {(), (Fe, Al, C, Ca, Si)}, ver **tablas 3.2** y **3.3**.

Coeficientes de actividad henriano

```
ab_henriano_std = {
        'Al': {'a': -71097.93, 'b': 12.88606   },
        'C' : {'a': 17234.09,  'b': -14.35763  },
        'Ca': {'a': -39398.16, 'b': 84.8902    },
        'Si': {'a': -131496.21,'b': 15.33027   }
        }
```

Coeficientes de la expresión de parámetros de interacción

```
ab_parametro ={
        'AlAl' :{'a': 58577.63 ,'b':28.5988    },
        'AlC'  :{'a': 77067.07 ,'b':0.0        },
        'AlCa' :{'a': -117222.0,'b':0.0        },
        'AlSi' :{'a': 108999.6 ,'b':0.0        },
        'CC'   :{'a': 199310.2 ,'b':0.0        },
        'CCa'  :{'a': -246024.0,'b':0.0        },
        'CSi'  :{'a': 155796.9 ,'b':-3.55822   },
        'CaCa' :{'a': 0.0      ,'b':0.0        },
        'CaSi' :{'a': -167236.0,'b':0.0        },
        'SiSi' :{'a': 33470.55 ,'b':85.2144    }
        }
```

El número de «expresiones» de parámetros de interacción necesarias es igual a

$$f(n) = \begin{cases} {}^{n}C_2 & \text{para } n > 1 \\ 0 & \text{para n} = 1 \end{cases} \tag{7.5}$$

donde n es el número de especies participantes. Si se ejecuta el módulo de manera independiente, realizará el cálculo de este sistema con la composición:

```
x={
    'Fe': 0.95797,
    'Al': 0.00978,
    'C' : 0.01955,
    'Ca': 0.00489,
    'Si': 0.00782}
```

y a 1873 K. El resultado de las actividades químicas de este sistema será:

```
1   >>>
2      i        x_i    ln gama_i^0    ln gama_i       gama_i            a_i
3
4     Fe    0.95797        0.00000     -0.00383      0.99617         0.9543
5     Al    0.00978       -3.01580     -2.83453      0.05875    0.000574252
6      C    0.01955       -0.62019     -0.32775      0.72054      0.0140869
7     Ca    0.00489        7.68047      7.21019   1353.14833        6.61363
8     Si    0.00782       -6.60042     -6.30413      0.00183   1.43009e-005
9   >>>
10
```

igual que lo obtenido en la **sección 3.13.1**, **tabla 3.4**.

7.15 Módulo `verificaEntradas`

Es un módulo auxiliar del modulo `Formula`, de selección de elementos de una lista.

8 Ejemplos de uso

En esta sección se presentan tres ejemplos que muestran el uso y la funcionalidad del programa `Ceqc`*. En el primero, se determina la composición en equilibrio de un sistema relativamente simple (monofásico ideal); en el segundo, de un sistema multifásico ideal y en el tercero, de un sistema «complejo» (multifásico, donde una de las fase es no ideal). Los resultados obtenidos para el último sistema se comparan con los resultados generados por dos programas comerciales para el cálculo de equilibrio químico.*

8.1 Ejemplo de uso para una solución química ideal

8.1.1 Ejemplo 1

El SO_2 es un precursor en la formación de SO_3 empleado para la producción de H_2SO_4. Un ejemplo de la generación de SO_3 a partir de SO_2 es el siguiente: calcular la fracción de SO_2 que en el equilibrio se convierte en SO_3 al oxidarse con O_2 a 900 K y 1 atm. Inicialmente se encuentran presentes SO_2 y O_2 en una relación equimolar.

Este es uno de los problemas de solución ideal más sencillos matemáticamente. El sistema

$$\{(SO_3, SO_2, O_2),\ \ (S, O)\}$$

es un sistema monofásico de gases ideales. Al utilizar `Ceqc` con estos datos se tiene[12]:

[12] Se contempla una mol inicial de SO_2 y otra de O_2.

```
1   > Escriba las especies del sistema >>  SO3 SO2 O2↩
2   > temperatura(K) >>  900↩
3   > presión(atm) >>  1↩
4
5   > Moles iniciales de  SO3 >>  0↩
6
7   > Moles iniciales de  SO2 >>  1↩
8
9   > Moles iniciales de  O2 >>  1↩
10
```

obteniéndose

```
11      ============================================================
12
13      > La fase de cada especie que tomaré en consideración es:
14
15      >   1.  SO3       Gas       (Gas)
16      >   2.  SO2       Gas       (Gas)
17      >   3.  O2        Gas       (Gas)
18
19      > Desea cambiar la fase de alguna de ellas >> ↩
20          > ¡Interpreté su respuesta como un NO!
21
```

```
22  ============================================================
23
24      > Fases de sistema:
25
26      >    gas:
27      >    ----------------------------------
28      >        SO3       Gas       (Gas)
29      >        SO2       Gas       (Gas)
30      >        O2        Gas       (Gas)
31
32      > ¿Desea cambiar la composición de alguna fase de sistema? >> ↩
33          > ¡Interpreté su respuesta como un NO!
34
```

```
35  ============================================================
36
37      > Los modelos disponibles para las fases del sistema son:
38
39      >   1. g_ideal_gas
40      >   2. g_ideal_liquido
41      >   3. g_ideal_solucion_solida
42      >   4. g_ideal_solido
43      >   5. parametros_de_interaccion
44
45      > Por favor seleccione algún modelo para cada fase.
46
47      > Modelo para "gas"  1↩
48
```

```
49
50
51      Abundancia de elementos
52      ============================================================
53          Elemento    Moles iniciales     Moles finales
54      ============================================================
55          S           1                   1
56          O           4                   4
57      ============================================================
58
59      Balance final
60      ============================================================
61      Especie     Moles Finales   Fracción mol
62      ============================================================
63      gas
64          SO3     0.8078775       0.5061695
65          SO2     0.1921225       0.1203729
66          O2      0.5960612       0.3734576
67      ============================================================
68
```

Los resultados obtenidos se resumen en la siguiente tabla.

que son el número de moles en equilibrio del sistema.

Tabla 8.1 Composición en equilibrio a 900 K del sistema compuesto por SO_3, SO_2, O_2.

Fase	Especie	Moles iniciales	Moles finales	Fracción mol, x
Gas				
	SO_3	0.0000	0.8078775	0.5061695
	SO_2	1.0000	0.1921225	0.1203729
	O_2	1.0000	0.5960612	0.3734576

8.1.2 Ejemplo 2

En un proceso de recuperación de Zn a 1380 K y 1atm de presión se efectúa la siguiente reacción

$$ZnO_{(s)} + CO_{(g)} \rightleftarrows Zn_{(g)} + CO_{2(g)}$$

Inicialmente se tienen 4 mol de ZnO, 10 mol de CO y 0.1 mol, determinar la composición en equilibrio de gas y la cantidad de Zn en equilibrio.

Este es un sistema un poco más complicado que el anterior, pues en el ya intervienen dos fases: una sólida y otra gaseosa. Al utilizar `Ceqc` de la misma manera que en el ejemplo anterior y con sólo estos datos se obtienen los siguientes resultados.

Tabla 8.2 Composición en equilibrio a 1380 K del sistema compuesto por $ZnO_{(s)}$, $CO_{(g)}$, $Zn_{(g)}$ y $CO_{2(g)}$.

Fase	Especie	Moles iniciales	Moles finales	Fracción mol, x
Gas				
	Zn	0.0000	3.298918	0.2462078
	CO_2	0.1000	3.398918	0.2536711
	CO	10.000	6.701082	0.5001212
Sólido				
	ZnO	4.0000	0.701082	1.0000000

8.2 Ejemplo de uso para una solución química no ideal

Un cálculo de equilibrio químico permite calcular las composiciones en equilibrio que simulan la precipitación de inclusiones en acero líquido. `Ceqc` es capaz de realizar tal cálculo, por ejemplo, considérese un acero líquido a 1873 K con la composición mostrada en la **tabla 8.3**. Además considérese una atmósfera compuesta por CO, CO_2, O_2 en las cantidades mostradas en la **tabla 8.3**.

Tabla 8.3 Composición inicial del acero líquido y de la atmósfera a 1873 K.

Especie	**Cantidad**/ mol
Fe	98.000
Al	1.000
C	2.000
Ca	0.500
Si	0.800

Acero líquido.

Especie	**Cantidad**/ mol
CO	1.000
CO_2	0.500
O_2	0.400

Atmósfera.

El acero líquido en contacto con la atmósfera formará inclusiones de diferente composición que se consideran cada una como una fase independiente. Las inclusiones formadas serán de dos tipos: $(CaO)(Al_2O_3)_2$ y $(CaO)_2(SiO_2)$, cuyas cantidades iniciales son cero.

Aunque es posible la formación de otros tipos de inclusiones como: CaO, FeO, Al_2O_3, $(CaO)_3(Al2O_3)$, $(CaO)(Al2O_3)$, $(CaO)(Al2O_3)_2$, $CaO(Al2O_3)_6$, $(CaO)_3(SiO_2)_2$, $(CaO)(SiO_2)$, $(CaO)_2SiO_2$, $(CaO)_3SiO_2$, SiO, $(Al_2O_3)_3(SiO_2)_2$, entre otras, todas ellas son menos estables termodinámicamente que $(CaO)(Al_2O_3)_2$ y $(CaO)_2(SiO_2)$. Esto es, al calcular la composición en equilibrio del sistema bajo las condiciones dadas e incluyendo estas especies, la cantidad de moles formadas de todas ellas, excepto para las dos últimas, tiende a ser cero. Por lo que, no se consideran en este ejemplo.

En la fase líquida se emplea el modelo de parámetros de interacción, en las sólidos el modelo de solución sólida ideal y en la fase gaseosa el modelo de gases ideales. Así, el sistema, será:

$$\{(C,\ Si,\ Ca,\ Al,\ Fe,\ CO_2,\ CO,\ O_2,\ (CaO)(Al_2O_3)_2,\ (CaO)_2(SiO_2)),\ (Al, C, O, Si, Fe, Ca)\}$$

Al ejecutar `Ceqc` con estos datos, el estado del sistema que considera inicialmente se muestra acontinuación. Se observa que el Ca se encuentra en estado gaseoso y el C forma una fase independiente como sólido.

```
1   > Fases de sistema:
2
3       >    liquido:
4       >    ---------------------------------
5       >        Si        Líquido   (Líquido)
6       >        Al        Líquido   (Líquido)
7       >        Fe        Líquido   (Líquido)
8
9       >    gas:
10      >    ---------------------------------
11      >        Ca        Gas       (Gas)
12      >        CO2       Gas       (Gas)
13      >        CO        Gas       (Gas)
14      >        O2        Gas       (Gas)
15
16      >    solido_1:
17      >    ---------------------------------
18      >        C         Sólido       (Grafito)
19
20      >    solido_0:
21      >    ---------------------------------
22      >        (CaO)(Al2O3)2Sólido    (Sólido)
23
24      >    solido_2:
25      >    ---------------------------------
26      >        (CaO)2(SiO2)Sólido     (Sólido)
27
```

Esta información debe modificarse mediante las opciones de cambio de fase y especie de `Ceqc` para obtener el estado del sistema como se muestra a continuación.

```
28  > Fases de sistema:
29
30      >    liquido:
31      >    ----------------------------------
32      >        C          Sólido    (Grafito)
33      >        Si         Líquido   (Líquido)
34      >        Al         Líquido   (Líquido)
35      >        Fe         Líquido   (Líquido)
36      >        Ca         Líquido   (Líquido)
37
38      >    gas:
39      >    ----------------------------------
40      >        Ca         Gas       (Gas)
41      >        CO2        Gas       (Gas)
42      >        CO         Gas       (Gas)
43      >        O2         Gas       (Gas)
44
45      >    solido_0:
46      >    ----------------------------------
47      >        (CaO)(Al2O3)2Sólido    (Sólido)
48
49      >    solido_2:
50      >    ----------------------------------
51      >        (CaO)2(SiO2)Sólido     (Sólido)
52
```

La parte a resaltar de este cálculo es que el sistema está compuesto por cuatro fases: dos sólidos, un gas y un líquido, y la fase líquida emplea un modelo no ideal para determinar el potencial químico en cada iteración, es decir se ejecuta el módulo `parametros_de_interaccion` con la composición transitoria en cada iteración.

Finalmente, después de modificar el sistema `Ceqc` realiza los cálculos y obtiene la composición del sistema en equilibrio, como se muestra en la **tabla 8.4**, en esta tabla, también, se muestran los resultados obtenidos con otros dos sistema de cómputo: `Fact`[19] y `Chemsage`[10].

No se observa una diferencia significativa entre los resultados obtenidos con el programa `Ceqc`, desarrollado en este trabajo, y los dos programas comerciales, siendo el sistema en comparación relativamente complejo.

Las posibles diferencias entre los resultados obtenidos con diferentes programas de cómputo pueden deberse a

1. El algoritmo empleado en el calculo principal
2. La información en la base de datos y
3. La precisión empleada en los cálculos y en la base de datos.

Tabla 8.4 Resultados del cálculo de equilibrio del sistema, C, Si, Ca, Al, Fe, CO_2, CO, O_2, $(CaO)(Al_2O_3)2$, $(CaO)_2(SiO_2)$ a 1873 K y 1 atm, empleando tres sistemas de cómputo: `Ceqc`, `Chemsage` y `Fact`.

Fase	**Especie**	`Ceqc`	`Chemsage`	`Fact`
Líquido				
	Si	0.674666	0.674729	0.6748200
	Al	0.002674	0.001926	0.0014049
	Fe	98.000000	98.000000	98.000000
	Ca	3.878995×10^{-8}	1.634540×10^{-9}	2.489800×10^{-9}
	C	2.946998	2.948700	2.949400
Gas				
	CO_2	0.000341	0.0011047	0.001165
	CO	0.552662	0.5501860	0.549430
	O_2	4.81185×10^{-14}	4.18336×10^{-13}	4.2629×10^{-13}
Sólido 0				
	$(CaO)(Al_2O_3)_2$	0.249331	0.249519	0.249650
Sólido 2				
	$(CaO)_2(SiO_2)$	0.125334	0.125241	0.125180

9 Comentarios de la ejecución de Ceqc

En esta sección se describen algunas hechos a resaltar encontrados durante la ejecución del programa de cómputo `Ceqc`.

9.1 Fase de una sola especie

`Ceqc` tiene dificultades para controlar el balance de masa cuando, en un sistema multifásico, la cantidad de moles en una fase de una sola especie se consume durante el cálculo.

9.2 Número de especies

`Ceqc` no puede realizar cálculo alguno si el número de especies es menor al número de elementos que la componen. Esto es debido a que no puede crearse la matriz estequiométrica.

Consideraciones finales

El programa `Ceqc`, al igual que la mayoría de los programas de cómputo, está sujeto a revisión y, de ser necesario, modificación con el fin de mejorar su desempeño. Por el momento, algunas de las características que `Ceqc` no posee, y muy personalmente, me gustaría que poseyera es, una interfaz gráfica de usuario (ventanas), GUI, que hiciera posible la selección y modificación de especies, fases y modelos termodinámicos de `Ceqc` de una manera no lineal, como lo hace actualmente.

El programa `Ceqc` ha sido diseñado para aceptar la inclusión de otros modelos termodinámicos tales como:

- Modelo de soluciones regulares para aleaciones líquidas,
- Modelos para escorias,
- Modelos de sub redes para sales fundidas.

Incorporar estos modelos (junto con la correspondiente base de datos) a la programación de `Ceqc` contribuiría a hacer de él un programa mejor. En cuanto al desempeño de su ejecución, aún quedan detalles que mejorar como los mostrados en al final de la sección anterior. **sección 9**. Sin embargo, considero que el desempeño de `Ceqc` es aceptable.

Conclusiones

Las conclusiones obtenidas considerando las capacidades del programa de cómputo `Ceqc` son las siguientes:

1. El programa es capaz de realizar el cálculo de la composición en equilibrio de un sistema monofásico ideal y no ideal.
2. Puede realizar cálculos de la composición en equilibrio de un sistema multifásico y multicomponente ideal y no ideal.
3. El programa permite calcular, para especies puras, las propiedades termodinámicas, tales como entalpía, entropía y energía libre de Gibbs a una temperatura dada.
4. El programa `Ceqc` simplifica la interacción con el usuario, en comparación con los programas comerciales, al requerir únicamente el nombre de las especies participantes, número de moles iniciales, temperatura y presión.
5. La estructura del programa `Ceqc` permite adicionar modelos de solución no contemplados inicialmente por el programador.
6. Es posible adaptar el programa de equilibrio químico `Ceqc` a programas que simulen procesos más complejos, como los de cinética química.

Del desempeño del programa de cómputo `Ceqc` se concluye que:

7. El programa `Ceqc` no puede controlar el balance de masa cuando, en un sistema multifásico, la cantidad de moles en una fase de una sola especie se consume durante el cálculo. Ésta es una limitación que deberá resolverse en una versión posterior del programa.
8. La exactitud de los resultados del programa `Ceqc` ha sido comprobada mediante la comparación con los resultados obtenidos con los programas comerciales `Fact` y `Chemsage`. En dicha comparación se empleó un sistema multifásico no ideal y no se observó una diferencia significativa entre los resultados de `Ceqc` y los dos programas comerciales.

Del método de minimización de energía empleado en el desarrollo de `Ceqc` se concluye que:

9. El método VCS, empleado en este trabajo, es un algoritmo estequiométrico que requiere de menos recursos computacionales en comparación con los algoritmos no estequiométricos. Esto es debido a que el número de variables siempre es igual al número de especies, N,

menos el número de elementos, M, para un algoritmo estequiométrico y es $M + N + \pi$ para un algoritmo no estequiométrico.

10. No se requiere manejo de excepciones (singularidades numéricas) para el algoritmo VCS contrario a lo que sucede con otros algoritmos encontrados en la literatura.

De la aplicación del programa `Ceqc` se concluye que:

11. El programa `Ceqc` puede servir como herramienta de investigación en el estudio de sistemas químicos pues posee una base de datos, de casi 1900 especies, totalmente extensible y modificable. Además de ser capaz de manejar modelos de solución definidos por el investigador. Pueden añadirse fácilmente otros modelos para considerar sistemas como escorias, aleaciones líquidas, sales fundidas, etcétera.
12. El programa `Ceqc` puede servir como material didáctico para la enseñanza de las funciones termodinámicas y del cálculo de equilibrio químico y teoría de soluciones.

Referencias

1. Asher, D., et. al., Numerical Python 2.3.5, Universidad de California, USA (2001). Disponible en www.python.org/doc/.

2. Avriel, M., Nonlinear Programming (Analysis and Methods), Dover publications, Nueva York (2003).

3. Chapra, S. C. y Canale, R. P., Métodos númericos para ingenieros, 3 ed., MacGraw–Hill, México (1999).

4. de Capitani, C. y Brown, T. H., The computation of chemical equilibrium in complex systems containing non–ideal solutions, **51**, Geochimica et Cosmochimica Acta, 2639 (1987).

5. Denbigh, K. G., The principles of chemical equilibrium, 4 ed., Cambridge University Press, Gran Bretaña (1981).

6. Edgar, T. F., Himmelblau, D. M. y Lasdon, L. S., Optimization of chemical processes, 2 ed., McGraw–Hill, Nueva York (2001).

7. Eriksson, G., Calculation of phase equilibria in multicomponent alloy systems using a special adapted version of the program «`Solgasmix`», CALPHAD, **18**, 15 (1984).

8. Eriksson, G., Thermodynamic studies of high temperature equilibria, Acta Chemica Scandinavica, **25**, 2651 (1971).

9. Eriksson, G., Thermodynamic studies of high temperature equilibria, Chemica Scripta, **8**, 100 (1975).

10. Eriksson, G. y Hack, K., `ChemSage`: A computer program for the calculation of complex equlibria, Metallurgical Transactions B, **21B**, 1013 (1990).

11. Gordon, S. y McBride, B. J., Computer program for calculation of chemical equilibrium compositions, rocket performance, incident and reflected shocks, and Chapman–Jouguet

detonations, NASA, USA (1971).

12. Kincaid, D. y Cheney, W., Análisis numérico. Las matemáticas del cálculo científico, Addison Wesley Iberoamericana, EUA (1994).

13. Lupis, C. H. P., Chemical Thermodynamics, 1 ed., North Holland, Nueva York (1983).

14. Lutz, M. y Asher, D., Learning Python, 2 ed., O'Reilly, EUA (2003).

15. Lutz, M., Programming Python, 2 ed., O'Reilly, EUA (2001).

16. Mathews, J. H. y Fink, K. D., Métodos numéricos con Matlab, 3 ed., Pentice Hall, Madrid (2000).

17. Myers, A. K., Computacional of multiple reaction equilibria, Chemical Engineering Education, 112 (1991).

18. Nakos, G. y Joyner D., Álgebra lineal con aplicaciones, International Thomson editores, México (2001).

19. Pelton, A. D., et. al., F*A*C*T Thermochemical database for calculations in materials chemistry at high temperatures, High Temperature Science, **26**, 231 (1990).

20. Smith, W. R. y Missen, R. V., Análisis del equilibrio en reacciones químicas, Noriega editores, México (1987).

21. Rao, Y. K., Stoichiometry and thermodynamics of metallurgical processes, Cambridge University Press, USA (1985).

22. Romero, J. A., Termodinámica metalúrgica y de materiales, IPN, México (2006).

23. van Rossum, G., Python Library Reference 2.3.5, PythonLabs, (2005). Disponible en www.python.org/doc/.

24. van Rossum, G., Python Reference Manual 2.3.5, PythonLabs, (2005). Disponible en

www.python.org/doc/.

25. van Rossum, G., Python Tutorial 2.3.5, PythonLabs, (2005). Disponible en www.python.org/doc/.

26. van Zeggeren, F. y Storey, S. H., The computation of chemical equilibria, 1 ed., Cambridge University Press, (1970).

27. Zeleznik, F. J. y Gordon, S., Calculation of complex chemical equilibria, Industrial and Engineering Chemistry, **60**, 27 (1968).

Apéndices

A Código fuente de `Ceqc`

A.1 Módulo `energia`

```
1       # -*- coding: cp1252 -*-
2       '''En este módulo se dan las definiciones para la obtención de \
3           los parámetros termodinámicos'''

4       __all__ = ['Energia', 'muestra_especies_base']

5       import re
6       #Éste es una base de datos definida por el diccionario 'Base'
7       from base_mia import Base
8       from Scientific.Functions.Romberg import romberg  #integral

9       class Energia:
10          def __init__(self, temperatura, especie, imprime = 0, joules = 1 ):
11              self.j = joules
12              self.T = float(temperatura)
13              self.especie = especie
14              self.base = Base[especie]
15              f = (lambda x: x and 1)
16              # 1 caloría = 4.184 Joule =3.184 + 1.0
17              self.factor = f(joules)*3.184 + 1.0
18              self.un = self.__unidad()
19              self.imprime = imprime
20              self.numeroDeCps = self.base['linea2']['fases']
21              self._calcula_ghs()

22          def __unidad(self):
23              if self.j:return 'J'
24              else: return 'cal'

25          def atributo(self, atributo):
26              try:
27                  return self.__dict__[atributo]
28              except KeyError:
29                  raise AttributeError, atributo

30          def _calcula_ghs(self,estandar=1):
31              '''Calcula la energía libre de Gibbs y devuelve la energia de gibbs,
32              entalpia , entropia, el tipo de fase y el nombre de fase'''
33              if estandar:
34                  self.claves_fases = []
```

```
35                 self.l_intervalos = []
36                 for i in range(self.numeroDeCps):
37                     self.l_intervalos += [(self.base['fase'+str(i+1)]['rango1'],\
38                         self.base['fase'+str(i+1)]['rango2'])]
39                     self.claves_fases += [self.base['fase'+str(i+1)]['fase']]
40                     self.fase = self.ubicacion_de_fase(self.T, self.l_intervalos)
41             else:
42                 pass
43             [self.a, self.b, self.c, self.d] =\
44                 [i for i in self.parametros(self.fase)]
45             self.entalpia = self.DeltaH(self.T, self.fase)*self.factor
46             self.entropia = self.DeltaS(self.T, self.fase)*self.factor
47             #ya no se requiere conversión
48             self.gibbs = self.DeltaG(self.entalpia, self.entropia, self.T)
49             self.datos_fase = self.base['fase'+str(self.fase)]
50             self.clave_fase = self.datos_fase['fase']
51             self.fase_nombre = self.datos_fase['nombre']
52             if self.imprime: self.imprime_resultados()

53         def ubicacion_de_fase(self, temperatura, l_intervalos):
54             fase = 0
55             if l_intervalos[0][0] > temperatura:
56                 print '\n\t> Advertencia:\n\t>\tLos parámetros termodinámicos de',\
57                     self.especie, 'se extrapolarán.'
58                 print '\t>\tLa temperatura mínima de definición para esta \
59                 especie es',l_intervalos[0][0],'K\n'
60                 return fase + 1

61             for intervalo in l_intervalos:
62                 fase += 1
63                 if intervalo[0] <= temperatura <= intervalo[1]:
64                     self.fase = fase
65                     return fase

66             print '\n\t> Advertencia:\n\t>\tLos parámetros termodinámicos de', \
67                 self.especie, 'se extrapolarán.'
68             print '\t>\tLa temperatura máxima de definición para esta\
69                 especie es', l_intervalos[-1][1],'K\n'
70             return fase

71         def cambia_fase(self):
72             print '\n\t>\t%-4s%-15s%-20s' % ('', 'Fase', 'Intervalo de validez')
73             print '\t>\t'+'-'*41
74             for i in range(self.numeroDeCps):
75                 print '\t>\t%-4s%-15s%-20s' % (str(i+1)+'.', \
76                     self.base['fase'+str(i+1)]['nombre'], self.l_intervalos[i])
77                 print ''
```

```
78              while 1:
79                  seleccion = raw_input('\t> Escoja una nueva fase: ')
80                  try:
81                      seleccion = int(seleccion)
82                      break
83                  except ValueError:
84                          pass

85              print ''
86              if seleccion in range( 1,self.numeroDeCps+1):
87                  self.fase = seleccion
88                  self._calcula_ghs(0)
89              else:
90                  print 'Seleccion no valida...'

91          def parametros(self,fase):
92              '''Retorna los parámetros para la ecuación de Cp'''
93              for parametro in ['a','b','c','d']:
94                  yield self.base['fase' + str(fase)][parametro]

95          def DeltaH298(self,fase):
96              '''Retorna la entalpia a 298 de la fase dada
97                  en T_transición se toma la fase inferior'''
98              if fase == 0:
99                  return self.base['fase1']['H298']
100             else:
101                 return self.base['fase' + str(fase)]['H298']

102         def S298(self,fase):
103             '''Retorna la entalpia a 298 de la fase dada
104                 en T_transición se toma la fase inferior'''
105             if fase == 0:
106                 return self.base['fase1']['S298']
107             else:
108                 return self.base['fase' + str(fase)]['S298']

109         def Cp(self,T):
110             return self.a + 1.E-3*self.b*T + 1.E5*self.c*T**-2 + 1.E-6*self.d*T**2.

111         def CpS(self,T):
112             return self.a*T**-1 + 1.E-3*self.b + 1.E5*self.c*T**-3 + 1.E-6*self.d*T

113         def DeltaH(self,temperatura,fase):
114             return  self.DeltaH298(fase)*1.E3 + \
115                 romberg(self.Cp, (298.0,temperatura))

116         def DeltaS(self,temperatura,fase):
```

```
117             return  self.S298(fase) + romberg(self.CpS, (298.0,temperatura))

118         def DeltaG(self,H,S,T):
119             return H - T*S

120         def imprime_resultados(self):
121             print '\t%15s%-3.30s'     %('especie : ', self.especie)
122             print '\t%15s%-15.30s'    %('nombre  : ', self.base['nombre'])
123             print '\t%15s%-15.10sK\n' %('temperatura : ',self.T)
124             print '\t%15s%-15.6gK'    %('T_trans : ',\
125                 self.base['fase'+str(self.fase)]['T_trans'])
126             print '\t%15s%-15.6g%-3s' %('entalpia : ', self.entalpia, self.un)
127             print '\t%15s%-15.6g%-3s' %('entropía : ', self.entropia, self.un)
128             print '\t%15s%-15.6g%-3s' %('e. libre : ', self.gibbs, self.un)
129             print '\t%15s%-15.6g%-3s' %('DeltaH298 : ',\
130                 self.DeltaH298(self.fase)*1E3*self.factor, self.un)
131             print '\t%15s%-15.6g%-3s' %('S298 : ', \
132                 self.S298(self.fase)*self.factor, self.un)
133             print '\t%15s%-15.10s'    %('clave_fase : ', self.clave_fase)
134             print '\t%15s%-15.30s'    %('nombre_fase : ', self.fase_nombre)
135             print ''
136             print '\t%15s%-15.50s'%\
137                 ('expresión Cp : ','a + 1E-3*b*T + 1E5*.c*T**-2 + 1E-6*d*T**2')
138             print '\t%15s%-9.9g, %-9.9g, %-9.9g, %-9.9g'   %\
139             ('a, b, c, d : ',self.a, self.b, self.c, self.d)

140     def muestra_especies_base():
141         '''Muestra un listado de las especies en la base de datos'''
142         cont = 1
143         listado = Base.keys()
144         listado.sort()
145         for i in listado:
146             if cont%6 != 0:print '%-20s' % str(i),
147             else: print i
148             cont += 1

149     if __name__ == '__main__':
150         def muestra(temperatura, especie, imprime = 1, joules = 1):
151             E = Energia(temperatura, especie, imprime, joules)
152             while 1:
153                 print '=='*40
154                 respuesta1 = raw_input('\n¿Desea cambiar la fase estándar?: ')
155                     if respuesta1.lower() in ('s', 'si'):
156                         print '\n\n\tCambia la fase estándar'
157                         print '\t'+'--'*30
158                         E.cambia_fase()
159                     elif respuesta1.lower() in ('n', 'no'):
```

```
160                         print '\tAdios...'
161                         break
162                     elif respuesta1.lower() not in ('s', 'si', 'n', 'no'):
163                         print '\t\tNo entiendo su respuesta.\
164                             Inténtelo nuevamente...'
165                         continue
166                     elif not respuesta0:
167                         print '\t¡Interpreté su respuesta como un NO!'
168                         print '\tAdios...'
169                     else:
170                         print '\t¡Interpreté su respuesta como un NO!'
171                         print '\tAdios...'
```

A.2 Módulo `constantes`

```
1       # -*- coding: cp1252 -*-
2       '''Módulo constantes

3       Se defininen algunas constantes físicas
4       '''

5       R = 8.314       #kPa m^3/(kg-mol K)
6                       #kJ/(kg mol K)

7       VMGI = 22.415   #Volumen molar de un gas ideal
8                       #cm^3/mol

9       NA = 6.023e23   #Número de Avogadro
10                      #moléculas/mol
11                      #átomos/(g-átomo)

12      CB = 1.38e-23   #Constante de Boltzmann
13                      #J/(molécula mol)
```

A.3 Módulo `formula`

```
1       # -*- coding: cp1252 -*-

2       __all__ = ['ValidaFormula','VectorFormula']
3       import re
4       from patrones import  *
5       from verificaEntradas import checaRepetidos
```

```
6       class ValidaFormula:
7           def __init__(self,formula):
8               self.formula = formula
9               self.formulaValida = \
10                  self.__sonIguales(self.formula,self.__validacion(self.formula))

11          def __checaSimbolos(self,formula, opcion):
12              '''Extrae de forma ordenada todos los carácteres o símbolos
13                  válidos de 'formula'. Regresa la cadena extraida'''
14              cadena = ''
15              if opcion == 0:
16                  valido = pCaracteres.finditer(formula)
17              else:
18                  valido = pElementos.finditer(formula)
19              for i in valido:
20                  cadena += i.group()
21              return cadena

22          def __generaValidos(self,formula):
23              ''' Extrae los segmentos válidos (sintáticamente)  \
24                  de 'formula' de acuerdo al 'pFormula'.'''
25              lista, cadena, x = [], '', 0
26              #se repite el ciclo hasta que sean iguales,  más una vez (+1)
27              while x <= 1:
28                  iterador = pFormula.finditer(formula)
29                  cadena = ''
30                  for i in iterador:
31                      aux = i.group()
32                      #si no está ya en 'lista' se incluye
33                      if aux not in lista:
34                          lista.append(aux)
35                      cadena += aux
36                  formula = cadena
37                  #una vez que sean iguales
38                  if formula == cadena:
39                      x += 1
40              return lista

41          def __validacion(self,formula):
42              return self.__generaValidos(self.__checaSimbolos(formula,1))

43          def __muestraOpciones(self,lista):
44              '''Los segmentos válidos de una fórmula mal escrita se \
45                  muestran para ser seleccionados'''
46              for i in range(len(lista)):
47                  print '\t\t'+str(i+1)+'.  ', lista[i]
48              print '\n\t> Si desea que alguno de estos segmentos lo considere \
```

```
49                  para los cálculos,'
50              print '\t> por favor escriba los números correspondientes (de lo\
51                  contrario presione "enter") >> ',
52              elecciones = map(int,list(re.findall('\d+',raw_input())))
53              enRango = []
54              for i in elecciones:
55                  if i in range(1,len(lista)+1):
56                      enRango.append(i)
57              elecciones = checaRepetidos(enRango)
58              for i in range(len(elecciones)):
59                  elecciones[i] = elecciones[i] - 1
60              return elecciones

61          def __sonIguales(self, formula, lista):
62              '''Si 'formula' y los elementos de 'lista' son diferentes
63              despliege opciones. Cuando una fórmula es escrita se extraen
64              los segmentos sintáticamente válidos mediante el método
65              '__generaValidos', si exite un único segmento válido y éste
66              es igual a la fórmula escrita, entonces la fórmula se ha
67              escrito correctamente. Si existe una diferencia entre ese
68              único segmento o hay más de un segmento, la fórmula se ha
69              escrito erróneamente'''
70              try:
71                  if len(formula) == len(lista[0]):
72                      pass
73                  else:
74                      print '\n\t> Existe un error en la escritura de la fórmula:\
75                          ', formula
76                      print '\t> Detecté algunos segmentos sintáticamente válidos \
77                          de la entrada estos son:\n '
78                      indices = self.__muestraOpciones(lista)
79                      #libera memoria y previene que guarde algun valor antrior
80                      lista2 = None
81                      lista2 = []
82                      if len(indices):
83                          for i in indices:
84                              lista2.append(lista[i])
85                          lista = lista2
86                      else:
87                          lista = []
88                  return lista, None
89              except IndexError:
90                  formulaErronea = formula
91                  return [], formulaErronea

92      class VectorFormula:
93          '''Convierte y devuelve el vector fórmula  de un compuesto y
```

```
94          pone los asteriscos a las fórmulas que les haga falta.
95          Nota: en la nueva versión de la base de datos no es necesario
96          poner asteriscos ni la notación fortran
97          '''

98          def __init__(self, formula):
99              self.formula = formula
100             self.formulaReducida = ''
101             #self.formulaFortran = ''
102             self.dicVector = {}
103             self.__vector()
104             #self.__poneAsteriscos()

105         def __vector(self):
106             lista = []
107             for i in pFormulaVector.finditer(self.formula):
108                 lista.append(i.group())
109             self.__todo(lista)

110         def __base(self, i, fuera=1):
111             elemento = re.match('[A-Z][a-z]*', i).group()
112             if re.search('\d+',i):
113                 cantidad = int(re.search('\d+',i).group())*fuera
114             else:
115                 cantidad = 1*fuera
116             if elemento in self.dicVector.keys():
117                 #print 'Éste ya está', elemento
118                 self.dicVector[elemento] += cantidad
119             else:
120                 self.dicVector[elemento] = cantidad

121         def __todo(self,lista):
122             for i in lista:
123                 if re.match('[A-Z][a-z]*[\d]*', i):
124                     self.__base(i)
125                     self.formulaReducida += i
126                 elif re.match('[(][(][A-Z][a-z]*[\d]*)+[)][\d]*', i):
127                     dentroFuera = re.finditer('([A-Z][a-z]*[\d]*)+|[\d]+',i)
128                     lista2 = []
129                     for j in dentroFuera:
130                         lista2.append(j.group())
131                     if len(lista2) == 2:#si no hay algún número
132                         dentro = lista2[0]
133                         fuera = int(lista2[1])
134                     elif len(lista2) == 1:#si no hay algún número
135                         dentro = lista2[0]
136                         fuera = 1
```

```
137                     #si afuera de los paréntesis hay un uno o nungún número
138                     if fuera == 1:
139                         self.formulaReducida += dentro
140                     else:
141                         self.formulaReducida += i
142                     dentroFuera2 = re.finditer('[A-Z][a-z]*[\d]*', dentro)
143                     for j in dentroFuera2:
144                         self.__base(j.group(),fuera)

145     if __name__ == '__main__':
146         formula = '(CaO)3(SiO2)'#'Fe2SO4[3+]'
147         F = ValidaFormula(formula)
148         Vec = VectorFormula(F.formulaValida[0][0])
149         print F.formulaValida[0][0]
150         print Vec.dicVector
```

A.4 Módulo LU

```
1       # -*- coding: cp1252 -*-
2       ## solucion por LU
3       from Numeric import zeros, shape, array, absolute, \
4           argmax, dot, transpose, sum, arange, \
5           concatenate,identity

6       def LU(M, q=None):
7           '''Factorización LU de una matriz M(n,m).

8           LLeva acabo la fase de factorización de la eliminación gaussiana
9           con pivoteo de filas escaladas. Aunque realiza intercambio de co-
10          lumnas, si es necesario, no realiza pivoteo total.

11          Devuelve la matriz LU, los vectores de permutación, p y q, de filas
12          y columnas, las dimensiones de la matriz, (n, m) y el número de filas
13          linealmente dependientes.'''

14          a = M.copy()
15          n, m = shape(a)
16          p = arange(n)
17          s = zeros(n, 'd')
18          if q==None:q=range(m)
19          for i in range(n):
20              s[i] = max(absolute(a[i]))
21          if not sum(s):
22              print 'Matriz nula'
23              return (None,)*6
```

```
24          for k in range(n):
25              h, q, dependencias = encuentraIndice(a, p, q, s, k, n, m)
26              if h == None: break
27              intercambiar(p,k,h)
28              for i in range(k+1, n):
29                  z = a[p[i],q[k]]/a[p[k],q[k]]
30                  a[p[i],q[k]] = z
31                  for j in range(k+1, n):
32                      a[p[i],q[j]] -= z*a[p[k],q[j]]
33          return a, p, q,  n, m, dependencias

34      def encuentraIndice(a, p, q, s, k, n, m):
35          '''Selecciona j >= k tal que  |a(p[j],k)|/s[p[j]] >= |a(p[i],k)|/s[p[i]]
36          para range(k, n).

37          Si el índice j no puede encontrarse dentro de la columna k,  inicial,
38          lo busca  en k+1, k+2, ..., m. De no encontrarse dicho índice declara
39          dipendencia lineal de las filas restantes, a partir de la fila k.

40          a       matriz de dimensiones (n,m).
41          p       vector de índices de permutación de filas.
42          q       vector de índices de permutación de columnas.
43          s       vector de las normas uniformes para cada fila de a(n,m),
44                  max(|a(i,j|) para 0 <= j <= m.
45          k       índices de la columna y fila inicial de búsqueda.
46          n-k     número de filas dependientes
47          '''
48          col = k
49          suma = 0.0
50          while 1:
51              as = zeros(n-k, 'd')
52              if col != m:
53                  for l in range(n-k):
54                      i = l + k
55                      as[l] = abs(a[p[i],q[col]])/s[p[i]]
56                  suma = sum(as)
57                  if suma:
58                      if col > k: intercambiar(q, k, col)
59                      return argmax(as) + k, q, 0
60                  else:
61                      col += 1
62              else:
63                  print '"Dependencia lineal de', n-k, 'fila(s)"'
64                  return None, q, n-k

65      def subMatriz(m_original, indices_fila, indices_col):
66          '''Genera una sub-matriz a partir de una matriz original.
```

```
67          Se proporcionan las listas de índices de filas y columnas,
68          de la matriz original, que contendrá la sub-matriz.'''

69          filas = len(indices_fila)
70          columnas = len(indices_col)
71          m_nueva = zeros((filas, columnas) , 'd')
72          print m_nueva
73          for i in range(filas):
74          for j in range(columnas):
75              m_nueva[i,j] = m_original[indices_fila[i],indices_col[j]]
76          return m_nueva

77      def intercambiar(arreglo, indice_1, indice_2):
78          '''Intercambia dos elementos de un arreglo.

79          Se proporciona, además del arreglo, los índices 1 y 2 de
80          los elementos a ser intercambiados
81          '''

82          flota = arreglo[indice_1]
83          arreglo[indice_1] = arreglo[indice_2]
84          arreglo[indice_2] = flota
85          return arreglo

86      def sustitucionAtras(A, p, q, n, m, dependencias):
87          '''Resuelve  Ax = b en términos de L y U

88          Se utilizan los componentes de la matriz A que se origina
89          del proceso gaussiano, función LU.

90          A        es la matriz aumentada [A:b] de dimensiones (n, m),
91                  por lo que no debe proporcionarse una matriz b extra.
92          b        es una matriz de dimensiones (n, m-n).
93          p        vector de índices de permutación de filas.
94          q        vector de índices de permutación de columnas.
95          dependencias    número de filas linealmente dependientes.'''

96          a = A.copy()
97          l = m - n + dependencias
98          z = zeros((l, n), 'd')
99          x = zeros((l, n), 'd')
100         n = n - dependencias
101         l = m - n
102         for k in range(l):
103         for i in range(n):
104             z[k,i] =a[p[i],q[k+n]] - suma1(0, i, a[p[i]], z[k])
105         for i in range(n-1, -1, -1):
```

```
106             x[k,i] = (z[k,i] - suma2(i+1, n, a[p[i]], x[k]))/a[p[i],q[i]]
107         return x

108     def suma1(inf, sup, v_1, v_2):
109         '''Sumatoria 1 del algoritmo de la función sustitucionAtras
110         '''
111         suma = 0.0
112         if sup > inf:
113             for j in range(inf, sup):
114                 suma += v_1[j]*v_2[j]
115         elif sup == inf:
116                 suma += v_1[0]*v_2[0]
117         return suma

118     def suma2(inf, sup, v_1, v_2):
119         '''Sumatoria 2 del algoritmo de la función sustitucionAtras
120         '''
121         suma = 0.0
122         if sup > inf:
123             for j in range(inf, sup):
124                 suma += v_1[j]*v_2[j]
125         elif sup == inf:
126                 suma += v_1[-1]*v_2[-1]
127         return suma

128     def espacioNulo(A, q=None):
129         '''Computa la base 'racional' del espacio nulo de la matriz A
130         '''
131         if q==None:
132             n, m = shape(A)
133             q = range(m)
134         a, p, q, n, m,  dependencias = LU(A, q)
135         #print n, m
136         rango = n - dependencias
137         gle = m - rango
138         X = sustitucionAtras(a, p, q, n, m, dependencias)
139         if not a:
140             print 'Adios...'
141             return None
142         x_T = zeros((rango, gle), 'd')
143         X_T = transpose(X)
144         for i in range(rango):
145             x_T[i] = X_T[i]
146         #print x_T
147         m_nula = concatenate((-x_T,  identity(gle, 'd')))
148         return m_nula, rango, gle, p, q, m, n
```

A.5 Módulo `modelos`

```
1       # -*- coding: cp1252 -*-

2       __all__ = ['g_ideal_gas', 'g_ideal_liquido', 'g_ideal_solucion_solida',\
3          'g_ideal_solido', 'parametros_de_interaccion' ]

4       from Numeric import log, array, zeros, dot # *
5       from p_interaccion import Actividades
6       from constantes import R

7       def g_ideal_gas(sistema, T, v_especies, presion, fase):
8           mol_fase = sistema.fases.moles_de_la_fase(fase)
9           for especie in sistema.fases.especies_de_la_fase(fase):
10              x_i = sistema.fases.moles_de_la_especie(especie)/mol_fase
11              g0 = sistema.fases.moles_de_la_especie(especie, 'g0')
12              g =  g0 + R*T*log(abs(presion*x_i))
13              sistema.fases.actualiza(especie, g, 'g')

14      def g_ideal_liquido(sistema, T, v_especies, presion, fase):
15          mol_fase = sistema.fases.moles_de_la_fase(fase)
16          for especie in sistema.fases.especies_de_la_fase(fase):
17              x_i = sistema.fases.moles_de_la_especie(especie)/mol_fase
18              g0 = sistema.fases.moles_de_la_especie(especie, 'g0')
19              g =  g0 + R*T*log(abs(x_i))
20              sistema.fases.actualiza(especie, g, 'g')

21      def g_ideal_solido(sistema, T, v_especies, presion, fase):
22              especie = sistema.fases.especies_de_la_fase(fase)[0]
23              g = sistema.fases.moles_de_la_especie(especie, 'g0')
24              sistema.fases.actualiza(especie, g, 'g')

25      def g_ideal_solucion_solida(sistema, T, v_especies, presion, fase):
26          mol_fase = sistema.fases.moles_de_la_fase(fase)
27          for especie in sistema.fases.especies_de_la_fase(fase):
28              x_i = sistema.fases.moles_de_la_especie(especie)/mol_fase
29              g0 = sistema.fases.moles_de_la_especie(especie, 'g0')
30              g =  g0 + R*T*log(abs(x_i))
31              sistema.fases.actualiza(especie, g, 'g')

32      def parametros_de_interaccion(sistema, T, v_especies, presion, fase):
33          mol_fase = sistema.fases.moles_de_la_fase(fase)
34          x_i={}
35          especies = sistema.fases.especies_de_la_fase(fase)
36          for especie in especies:
37              x_i[especie] = sistema.fases.moles_de_la_especie(especie)/mol_fase
38          A = Actividades('Fe', T, x_i)
```

```
39          for especie in especies:
40              g0 = sistema.fases.moles_de_la_especie(especie, 'g0')
41              g =  g0 + R*T*log(abs(A.actividad[especie]))
42              sistema.fases.actualiza(especie, g, 'g')
```

A.6 Módulo patrones

```
1       # -*- coding: cp1252 -*-
2       import re

3       __all__ = ['pCaracteres', 'pFormula', 'pElementos', 'pFormulaVector',\}
4           'pElemento_elemento']

5       '''Módulo 'patron'. Aquí se definen  los patrones de búsqueda para
6       las expresiones de fórmulas proporcionadas por el usuario'''

7       #patrón de caracteres válidos
8       pCaracteres = re.compile('([A-Z]|[a-z]|\d|[(]|[)]|[[]|[]]|\+|\-)+')

9       #patrón estándar de una fórmula química, puede ser iónica, [+]
10      pFormula = re.compile(\
11          '(([A-Z][a-z]{0,1}[\d]*)+|([(][A-Z][a-z]{0,1}[\d]*)+\
12          [)][\d]*)+)+([[][\d]*([+]|[-])[]])?')

13      #patrón estándar de una fórmula química, puede ser iónica,
14      #[+] para la clase Vector
15      pFormulaVector = re.compile(
16          '[(]([A-Z][a-z]*[\d]*)+[)][\d]*|\
17          |[A-Z][a-z]*[\d]*|[[][\d]*([+]|[-])[]]')

18      #elemento seguido de elemento, patrón para poneasteriscos
19      pElemento_elemento = re.compile('[A-Z][a-z]?(?=[A-Z][a-z]?)')

20      #elementos válidos en una formula química,
21      #se incluye el indicador iónico de la forma: [numero signo]
22      #deben colocarse los elementos monocaracteres al final de la lista
23      pElementos = re.compile(\
24          r'''(Li|Na|Rb|Cs|Fr|Be|Mg|Ca|Sr|Ba|Ra|\
25          |Al|Ga|In|Tl|Si|Ge|Sn|Pb|As|Sb|Bi|\
26          |Se|Te|Po|Cl|Br|At|He|Ne|Ar|Kr|Xe|\
27          |Rn|Cu|Ag|Au|Zn|Cd|Hg|Sc|La|Ac|Ti|\
28          |Zr|Hf|Ku|Nb|Ta|Ha|Cr|Mo|Mn|Tc|Re|\
29          |Fe|Co|Ni|Ru|Rh|Pd|Os|Ir|Pt|H|K|B|\
30          |C|N|P|O|S|V|W|F|I|Y|[(]|[)]|\d+|\
31          |\[\d*\+\]|\[\d*-\])+''')
```

A.7 Módulo `termoMiscelanea_2`

```
1       # -*- coding: cp1252 -*-
2       '''Módulo de miscelánea VCS y Termodinámica'''

3       __all__ = ['vec_g', 'vec_deltaGj','Fase','Fases']
4       import re
5       #from Numeric import log, dot, exp
6       from Numeric import array, zeros, log, dot, exp
7       import modelos as modelos
8       from energia import Energia
9       from Scientific.Functions.Derivatives import isDerivVar

10      def vec_g(sistema, temperatura, v_especies, presion):
11          'Calcula la energía libre de cada especie en el sistema'
12          g = []
13          for fase in sistema.fases.fases():
14              sistema.fases[fase].modelo(sistema, temperatura,\
15                  v_especies, presion, fase)
16          for especie in v_especies:
17              g += [sistema.fases.moles_de_la_especie(especie, 'g')]
18              return g

19      def vec_deltaGj(gle, rango, v_g, m_nulaT):
20          v_deltaGj = []
21          for j in range(gle):
22              i = j + rango
23              v_deltaGj += [
24                  v_g[i]
25                  + dot(m_nulaT[j][:rango],v_g[:rango])
26                  ]
27          return array(v_deltaGj, 'd')

28      class Clasifica_especies_en_fase:
29          def __init__(self, v_especies, T):
30              self.T = T
31              self.cont = 0
32              self.v_especies = v_especies
33              self.__estableceFases()
34              self.cambiar_fase_de_la_especie()
35              self.muestra_sistema_fases()
36              self.modifica_fases()
37              self.selecciona_modelo()

38          def __estableceFases(self):
39              '''Establece las especies en una fase según su clave de fase.
```

```
40              Genera el diccionario fases donde se almacena esta información.
41              Se utiliza la clase 'Energia' para determinar: las propiedades
42              termodinamicas estándar de cada especie, y algunos otros datos
43              de la fase estándar de cada especie'''
44              self.fases = Fases()
45              for especie in self.v_especies:
46                  aux = Energia(self.T, especie)
47                  clave = aux.atributo('clave_fase')
48                  estado = self.__indica_estado(clave)
49                  if not self.fases.has_key(estado):
50                          self.fases[estado] = Fase()
51                  self.fases[estado][especie] = {'mol':0.0, 'termo':aux}
52                  del aux
53                  del clave

54          def __indica_estado(self, clave, muestra=0):
55              if (100 < clave < 800):
56                  if muestra:
57                      return 'Sólido'
58                  else:
59                      self.cont += 1
60                      return 'solido_'+str(self.cont-1)
61              elif (800 < clave < 900):
62                  if muestra: return 'Líquido'
63                  else:return 'liquido'
64              elif (900 < clave < 990):
65                  if muestra: return 'Gas'
66                  else: return 'gas'
67              elif (990 < clave < 1000):
68                  if muestra: return 'Acuoso'
69                  else: return 'acuoso'
70              else : raise 'NoContempladoError',\
71                  'Se ha empleado "else": __indica_estado'

72          def cambiar_fase_de_la_especie(self):
73              print '\n\t' + '='*60 + '\n'
74              print '\t> La fase de cada especie que tomaré en consideración es:\n'
75              cont = 1
76              dic = {'liquido':'Líquido','gas':'Gas', 'acuoso':'Acuoso'}
77              for especie in self.v_especies:
78                  fase = self.fases.fase_de_la_especie(especie)
79                  print '\t>\t'+str(cont)+'. ',
80                  print '%-10s%-10s%-10s' % (str(especie),
81                      str(dic.get(fase, 'Sólido')),
82                      '('+str(self.fases[fase][especie]['termo'].fase_nombre)+')')
83                  cont += 1
84              r1 = raw_input('\n\t> Desea cambiar la fase de alguna de ellas >> ')
```

```
85              while 1:
86                  if r1.lower() in ('s', 'si'):
87                      r2 = raw_input('\t> Escriba el número de la especie » ')
88                      try:
89                          r2 = int(r2) - 1
90                          especie = self.v_especies[r2]
91                          fase = self.fases.fase_de_la_especie(especie)
92                          self.fases[fase][especie]
93                          self.fases[fase][especie]['termo'].cambia_fase()
94                          clave = self.fases[fase][especie]['termo'].atributo(\
95                              'clave_fase')
96                          estado = self.__indica_estado(clave)
97                          if not self.fases.has_key(estado):
98                                  self.crear_fase_s(estado, [especie])
99                          else:
100                                 self.cambiar_de_fase_s(estado, [especie])
101                         r1 = raw_input('\n\t>'+\
102                             ' Desea cambiar la fase de alguna otra >> ')
103                     except ValueError:
104                         print '\t\t> ' + r2 + ' no es una elección válida'
105                         print '\t\t> Por favor escoja un número válido'
106                 elif r1.lower() in ('n', 'no'):
107                     break
108                 else:
109                     print '\t\t> ¡Interpreté su respuesta como un NO!\n'
110                     break

111         def cambiar_de_fase_s(self, fase_nueva, especies):
112             '''Cambia la fase del sistema donde se encuentran una o
113                 varias especies.

114             Si 'elimina_anterior' es verdadero elimina la especie
115             de la fase donde se encontraba, la especie, anteriormente'''
116             for especie in especies:
117                 if especie in self.v_especies:
118                     fase_anterior = self.fases.fase_de_la_especie(especie)
119                     if fase_anterior == fase_nueva:
120                         pass
121                     else:
122                         aux = self.fases[fase_anterior][especie]
123                         self.__elimina_especie_de_fase_s(especie)
124                         self.fases[fase_nueva][especie] = aux
125                         del aux
126                 else:
127                     print '\tLa especie ', especie, 'no está registrada'

128         def crear_fase_s(self, fase_nueva, especies):
```

```
129             '''Crea una nueva fase del sistema con 'especies' como componenetes'''
130             if not especies == []:
131                 self.fases[fase_nueva] = Fase()
132                 self.cambiar_de_fase_s(fase_nueva, especies)
133             else:
134                 print '\tNo puedo contemplar una fase sin especies.'

135         def __elimina_especie_de_fase_s(self, especie, elimina_anterior = True):
136             try:
137                 if elimina_anterior:
138                     fase_anterior = self.fases.fase_de_la_especie(especie)
139                     del self.fases[fase_anterior][especie]
140                     if self.fases[fase_anterior] == {}:
141                         del self.fases[fase_anterior]
142             except KeyError:
143                 pass

144         def selecciona_modelo(self):
145             '''Aquí se selecciona el modelo ha ocupar en cada fase

146             Se genera el "fases.modelo" que almacena el modelo de la fase
147             '''
148             model = modelos.__all__
149             numModelos = len(model)
150             print '\n\t' + '='*60 + '\n'
151             print '\t> Los modelos disponibles para las fases del sistema son:\n'
152             for i in range(numModelos):
153                 print '\t>\t'+str(i+1) + '. ' + str(model[i])
154             print '\n\t> Por favor seleccione algún modelo para cada fase.'
155             for i in self.fases.fases():
156                 clave = True
157                 while clave:
158                     try:
159                         r1 = int(raw_input('\n\t> Modelo para "'\
160                             +str(i)+'" >> ')) - 1
161                         if r1 < 0:raise ValueError
162                         self.fases[i].modelo = modelos.__dict__[model[r1]]
163                         clave = False
164                     except (ValueError, IndexError):
165                         print '\t\t> '+\
166                             'Seleccione el número de algún modelo disponible.'
167                         clave = True

168         def modifica_fases(self):
169             respuesta0 = raw_input(
170                 '\t> ¿Desea cambiar la composición de alguna fase de sistema? » ')
171             if respuesta0.lower() in ('n', 'no'):
```

```
172                 pass
173             elif respuesta0.lower() in ('s','si'):
174                 dic1 = {1: 'Cambiar especies a otra fase de sistema',
175                     2: 'Crear una nueva fase de sistema',
176                     3: 'Mostrar composicion de fases de sistema',
177                     4: 'No cambiar fases de sistema'}
178                 for i in dic1:
179                     print '\t>\t' + str(i) + '. ' + dic1[i]
180                 try:
181                     respuesta1 = int(re.findall('\d+',\
182                         raw_input('\n\t> Seleccione una opción » '))[0])
183                     if respuesta1 == 1:
184                         dic2 = {}
185                         cont = 1
186                         print '\n\t> Fases de sistema disponibles:'
187                         for i in self.fases.fases():
188                             print '\t> \t' + str(cont) + '. ', i
189                             dic2[cont] = i
190                             cont += 1
191                         while 1:
192                             try:
193                                 fase_nueva = int(
194                                     re.findall('\d+',raw_input(
195                                     '\n\t>
196                                     Seleccione la fase de sistema
197                                     donde estarán las especies: '))[0])
198                             except IndexError:
199                                 print '\n\t> ¡Selección inválida!'
200                                 continue
201                             especies = raw_input('\t>
202                                 Escriba las especies que se adicionarán a
203                                 la fase de sistema "'\
204                                  + str(dic2[fase_nueva]) + '": ').split()
205                             self.cambiar_de_fase_s(dic2[fase_nueva],especies)
206                             print '\t> El sistema es: '
207                             self.muestra_sistema_fases()
208                             self.modifica_fases()
209                             break
210                     elif respuesta1 == 2:
211                         fase_nueva = raw_input('\t>
212                             Escriba el nombre de la nueva fase de sistema: ')
213                         especies = raw_input('\t> Escriba las especies
214                             que contendrá la fase de sistema "'\
215                             + str(fase_nueva) + '": ').split()
216                         self.crear_fase_s(fase_nueva, especies)
217                         print '\t> El sistema es: '
218                         self.muestra_sistema_fases()
219                         self.modifica_fases()
```

```
220                     elif respuesta1 == 3:
221                         self.muestra_sistema_fases()
222                         self.modifica_fases()
223                     elif respuesta1 == 4:
224                         pass
225                     else:
226                         pass
227                 except IndexError:
228                     print '\n\t> ¡Selección inválida!'
229             elif not respuesta0:
230                 print '\t\t> ¡Interpreté su respuesta como un NO!\n'
231             else:
232                 print '\t\t> ¡Interpreté su respuesta como un NO!\n'

233         def muestra_sistema_fases(self):
234             print '\t'+ '='*60
235             print '\n\t> Fases de sistema:\n'
236             for fase in self.fases.fases():
237                 print '\n\t>\t ' + str(fase) + ':'
238                 print '\t>\t ' + '-'*43
239                 for especie in self.fases[fase].especies():
240                     clave = self.fases[fase][especie][\
241                         'termo'].atributo('clave_fase')
242                     print '\t>\t\t %-10s%-10s%-10s' % (str(especie),
243                         self.__indica_estado(clave, 1),
244                         '('+str(self.fases[fase][especie][\
245                             'termo'].fase_nombre)+')')
246             print '\n'

247         def datos_termo_std_sistema(self, propiedad):
248             lista = []
249             for especie in self.v_especies:
250                 lista += [self.datos_termo_std(especie, propiedad)]
251             return lista

252         def datos_termo_std(self, especie, propiedad):
253             fase = self.fases.fase_de_la_especie(especie)
254             return self.fases[fase][especie]['termo'].atributo(propiedad)

255     class Fase(dict):
256         def moles(self, entrada='mol'):
257             return sum([x[entrada] for x in self.values()])

258         def especies(self):
259             'Regresa el numero de especies de la fase'
260             return self.keys()
```

```
261         def es_multicomponente(self):
262             if len(self) == 1: return 0  #False
263             else: return 1  #True

264         def modelo(self):
265             self.modelo = None

266     class Fases(dict):
267         def moles_totales(self, entrada = 'mol'):
268             '''Regresa las moles totales del sistema'''
269             suma = 0.0
270             for i in self.keys():
271                 suma += sum([x[entrada] for x in self.get(i).values()])
272             return suma

273         def actualiza(self, especies, propiedad, entrada = 'mol'):
274             '''Actualiza la clase 'Fases' con nuevas moles'''
275             if type(especies) != type([]):
276                 especies = [especies]
277             if type(propiedad) == type(array([0])):
278                 pass
279             elif type(propiedad) != type([]):
280                 propiedad = [propiedad]
281             cont = 0
282             for i in especies:
283                 for j in self.keys():
284                     if self[j].has_key(i):
285                         self[j][i][entrada] =  propiedad[cont]
286                         break
287                 cont += 1

288         def especies(self):
289             '''Regresa el nombre de las especies del sistema'''
290             nombre = []
291             for i in self.keys():
292                 nombre += self.get(i).keys()
293             return nombre

294         def fases(self):
295             '''Regresa una lista de las fases presentes'''
296             return self.keys()

297         def moles_de_la_especie(self, especie, entrada='mol'):
298             '''Regresa la cantidad de moles de la 'especie' '''
299             for fase in self.keys():
300                 try:return self[fase][especie][entrada]
301                 except KeyError: pass
```

```
302             print 'No existe especie: ', especie

303         def moles_de_la_fase(self, fase, entrada='mol'):
304             '''Regresa la cantidad de moles de la 'fase' '''
305             try: return sum([x[entrada] for x in self[fase].values()])
306             except KeyError: print 'Fase no existente'

307         def moles_de_la_fase_donde_la_especie(self, especie, entrada='mol'):
308             return self.moles_de_la_fase(self.fase_de_la_especie(especie), entrada)

309         def fase_de_la_especie(self, especie):
310             '''Regresa el nombre de la fases donde se encuentra la 'especie' '''
311             for j in self.keys():
312                 if self[j].has_key(especie):
313                     return j
314             print 'No existe especie: ', especie

315         def especies_de_la_fase(self,fase):
316             '''Regresa las especies que se encuentran en la 'fase' '''
317             return self[fase].keys()

318         def fase_multicomponente(self, fase):
319             '''Comprueba si la 'fase' es multicomponente'''
320             if fase != None:
321                 try:
322                     if len(self[fase]) == 1: return 0  #False
323                     else: return 1  #True
324                 except KeyError: print 'Fase no existente'

325         def especie_en_fase_multicomponente(self, especie):
326             '''Comprueba si la 'especie' se encuentra en
327             una fase multicomponente'''
328             try: return self.fase_multicomponente(self.fase_de_la_especie(especie))
329             except TypeError: pass

330         def especie_en_fase_especifica_multicomponente(self, fase, especie):
331             if (self.fase_multicomponenete(fase) and
332                 (especie in self.especies_de_la_fase(fase))):
333                 return True
334             else:
335                 return False
```

A.8 Módulo `VerificaEntradas`

```
1       # -*- coding: cp1252 -*-
```

```
2       def eliminaElementos (lista, indices):
3           '''Eliminar las especies repetidas en 'b'.'''
4           fijo=len(indices)
5           for i in range(fijo):
6               del lista[indices[i]]
7               #si no es la última
8               if i != (fijo-1):
9                   #corrección de desplazamiento por eliminación
10                  indices[i+1] -= i+1
11          return list(lista)

12      def checaRepetidos(lista):
13          '''Checa la repetición de elementos en un 'lista' y los elimina.'''
14          repetidos,indices=[],[]
15          for i in range(len(lista)):
16              #  si el elemento del índice  se sabe repetido se salta
17              if i in indices:
18                  continue
19              for j in range(i+1,len(lista)):
20                  #  si el elemento del índice  se sabe repetido se salta
21                  if j in indices:
22                      continue
23                  if lista[i] == lista[j]:
24                      indices.append(j)
25                  else:
26                      pass
27          #ordena 'indices' de menor a mayor
28          indices.sort()
29          # imprime el mensaje para advertir que borrará las especies repetidas
30          if indices:
31              for i in indices:
32                  if not lista[i] in repetidos:repetidos.append(lista[i])
33              #eliminar las especies repetidas en 'lista'
34              b = eliminaElementos(lista,indices)
35          return lista

36      def pruebaEntradas(lista):
37          '''Depura y valida los elementos de lista'''
38          lista = checaRepetidos(lista)
39          return lista
```

A.9 Módulo `delta_xi_fases`

```
1       # -*- coding: cp1252 -*-
2       from Numeric import array, dot, ones, transpose
3       from miscelanea_2 import *
```

```
4       from termoMiscelanea_2 import vec_g
5       from LU import espacioNulo
6       from constantes import  R

7       __all__ = ['VCS']

8       #proporcionar moles de cada fase como vector

9       class VCS:
10          '''Calcula el vector delta_xi con el algoritmo VCS

11          Si multifase > 0 realiza el cálculo para un sistema multifásico
12          si multifase = 0 realiza el cálculo para un sistema monofásico
13          '''
14          def __init__(self, sistema, v_especies, numElementos, gle, multifase=1):
15              self.multifase = multifase
16              self.sistema = sistema
17              self.T = sistema.T
18              self.RT = R* self.T
19              self.v_especies = v_especies
20              self.numElementos = numElementos
21              self.numEspecies = len(v_especies)
22              self.gle = gle

23          def __sum_v2(self, v_1, v_2):
24              suma = 0.0

25              #multifase
26              if self.multifase:
27                  for i in range(self.numElementos):
28                      suma += v_1[i]**2.*self.multicomponente[i] / v_2[i]

29              #monofase
30              else:
31                  for i in range(self.numElementos):
32                      suma += (v_1[i]**2. / v_2[i])
33              return suma

34          def  __sumaDoble(self, m_nulaT_j):
35              suma = 0.0
36              for fase in self.sistema.fases.fases():
37                  for k in range(self.numEspecies):
38                      if self.v_especies[k] in /
39                                  self.sistema.fases.especies_de_la_fase(fase):
40                          if self.sistema.fases.fase_multicomponente(fase):
41                              suma += m_nulaT_j[k]**2./\
42                                      self.sistema.fases.moles_de_la_fase(fase)
```

```
43              return suma

44          def __multiples(self):
45              self.multicomponente = []
46              for i in self.v_especies:
47                  self.multicomponente += [
48                          self.sistema.fases.especie_en_fase_multicomponente(i)]

49          def __nu_jk_EnFaseMultiple(self, nu_j, multicomponente):
50              '''Dado el vector estquiométrico nu_j, de la matriz nula
51              transpuesta, y un vector 'multiples' que indique si la especie k se
52              encuentra en una fase multicomponente; nu_kj_EnFaseMultiple regresa
53              'True' siempre que cuando al menos una especie para la cual
54              nu_jk != 0 se encuentre en una fase multicomponente. 'False' en
55              caso contrario.
56              '''
57              for k in range(self.numEspecies):
58                  if nu_j[k] and multicomponente[k]:
59                      return True
60              return False

61          def vec_delta_xi(self, v_deltaGj, m_nulaT, v_n):
62              k = self.numElementos
63              v_delta_xi = []
64              #multifase
65              if self.multifase:
66                  self.__multiples()
67                  for j in range(self.gle):
68                      if self.__nu_jk_EnFaseMultiple(m_nulaT[j],
69                                                     self.multicomponente):
70                          E = (1.*self.multicomponente[j+k]/v_n[j+k]
71                               + self.__sum_v2(m_nulaT[j], v_n)
72                               - self.__sumaDoble(m_nulaT[j])
73                               )
74                      else:
75                          E = 1.
76                      v_delta_xi += [(-1.0/E) * v_deltaGj[j]/self.RT]

77                  return array(v_delta_xi, 'd')

78              #monofase
79              else:
80                  suma_n = sum(v_n)
81                  for j in range(self.gle):
82                      E = (1.0/v_n[j+k]
83                            + self.__sum_v2(m_nulaT[j], v_n)
84                            - sum(m_nulaT[j])**2. / suma_n
```

```
85                          )
86                      v_delta_xi.append( (-1.0/E) * v_deltaGj[j]/self.RT)
87                  return array(v_delta_xi, 'd')

88          def imprime(self):
89              print self.multifase
```

A.10 Módulo `entrada`

```
1       # -*- coding: cp1252 -*-
2       from __future__ import division
3       import re
4       import sys
5       from constantes import R
6       from Numeric import  array, transpose
7       from Formula import ValidaFormula, VectorFormula
8       from miscelanea_1 import *
9       from termoMiscelanea_2 import Clasifica_especies_en_fase

10      class Atributos(MolesIniciales, Ordenador):
11          def __init__(self):
12              self.dic_especies = {}
13              self._especies_cadena()
14              self._temperatura()
15              self._presion()
16              self.RT = R * self.T
17              self.v_especies = self._especies()
18              self._atributos()
19              self._listaDeElementos()
20              self._completaVectorFormula()
21              self.numElementos = len(self.v_elementos)
22              self.numEspecies = len(self.v_especies)
23              self._creaMatrizFormula()
24              self.m_atomica = transpose(self.m_formula)
25              self._moles_iniciales()
26              self._molesDeElemento()
27              self.sistema = Clasifica_especies_en_fase(self.v_especies, self.T)
28              self.v_g0 = self.sistema.datos_termo_std_sistema('gibbs')
29              self.q = range(self.numEspecies)
30              self.p = range(self.numElementos)
31              self.rango = self.numElementos  ##este valor es provicional
32              self.completa_sistema('vector')

33          def _especies_cadena(self,opcion = 1):
34              self.__cadenaAlimentada = raw_input('\n> Escriba las especies del \
35                                                  sistema >> ')
```

```
36              if opcion == 1: self.__entradas()

37          def __entradas(self):
38              '''Genera un diccionario depurado de compuestos alimentados'''
39              self._especiesAlimentadas = self.__cadenaAlimentada.split()
40              self.erroneas = []
41              for especie in self._especiesAlimentadas:
42                  formula = ValidaFormula(especie)
43                  if formula.formulaValida[1] == None:
44                      for i in formula.formulaValida[0]:
45                          self.dic_especies[i] = {}
46                  else:
47                      self.erroneas.append(formula.formulaValida[1])
48              if self.erroneas:
49                  print '\n\t> Existen errores de escritura en las fórmulas:\n'
50                  for i in self.erroneas:
51                      print '\t\t> ', i
52                  print '\n\t> No pude obtener algún segmento válido de ellas.'
53                  print '\t> y no las consideraré en los cálculos.'
54                  print '\t> ¡Por favor verifíquelas!\n'

55          def _temperatura(self):
56              self.T = raw_input('> temperatura(K) >> ')
57              if self.T == '':
58                  print '\n\t> Es necesario definir una temperatura.'
59                  print '\t> Por favor hágalo.'
60                  print '\t>',
61                  self._temperatura()
62              else:
63                  try:
64                      self.T = float(eval(self.T))
65                  except ValueError:
66                      print '\n\t> No puedo interpretar su respuesta.'
67                      print '\t> Por favor escriba otra',
68                      self._temperatura()

69          def _presion(self):
70              self.presion = raw_input('> presión(atm) » ')
71              if self.presion == '':
72                  print '\n\t> Es necesario establecer la presion del sistema.'
73                  print '\t> Por favor hágalo.'
74                  print '\t>',
75                  self._presion()
76              else:
77                  try:
78                      self.presion = float(eval(self.presion))
79                  except ValueError:
```

```
80                      print '\n\t> No puedo interpretar su respuesta.'
81                      print '\t> Por favor, escriba otra',
82                      self._presion()
83          def _especies(self):
84              return self.dic_especies.keys()

85          def _atributos(self):
86              '''Crea un diccionario con los atributos de los compuestos.
87              El vector fórmula no esta aquí completo aún'''
88              for i in self.v_especies:
89                  VF = VectorFormula(i)
90                  self.dic_especies[i] = {'entrada': i,
91                              'vector' : VF.dicVector}
92          def _listaDeElementos(self):
93              '''Crea una lista todos los elementos presentes en los compuestos.'''
94              self.v_elementos = []
95              nombreCompuestos = self.v_especies
96              for compuesto in nombreCompuestos:
97                  for elemento in self.dic_especies[compuesto]['vector']:
98                      if not elemento in self.v_elementos:
99                          self.v_elementos.append(elemento)
100                     else:pass

101         def _completaVectorFormula(self):
102             '''El objetivo es completar el vector fórmula con lo elementos
103             aún no contemplados
104             '''
105             for compuesto in self.v_especies:
106                 for elemento in self.v_elementos:
107                     if not elemento in self.dic_especies[compuesto]['vector']:
108                         self.dic_especies[compuesto]['vector'][elemento] = 0
109         def _creaMatrizFormula(self):
110             self.m_formula = [[] for x in self.v_elementos]
111             h = 0
112             for j in self.v_elementos:
113                 for i in self.v_especies:
114                     self.m_formula[h].append(self.dic_especies[i]['vector'][j])
115                 h += 1
116             self.m_formula = array(self.m_formula)

117         def _moles_iniciales(self):
118             '''El objetivo de este método es almacenar las moles iniciales de
119             todas las especies
120             '''
121             for i in self.v_especies:
122                 self.__mol_inicial(i)
```

```
123         def __mol_inicial(self, especie):
124             '''El objetivo de este método es almacenar las moles iniciales de
125             una especie
126             '''
127             print '\n> Moles iniciales de ', especie, '>> ',
128             mol = raw_input()
129             if mol == '':
130                 print '\n\t> ¡Cantidad de moles igual a "Cero"!'
131                 mol = 0.0
132                 self.dic_especies[especie]['molesIniciales'] = mol
133             else:
134                 try:
135                     mol = float(eval(mol))
136                     self.dic_especies[especie]['molesIniciales'] = mol
137                 except ValueError:
138                     print '\n\t> No puedo interpretar su respuesta.'
139                     print '\t> Por favor escriba otra cantidad.'
140                     print '\t>',
141                     self.__mol_inicial(especie)

142         def _molesDeElemento(self):
143             self.dic_elementos = {}
144             self.v_abundancia = []
145             for i in self.v_elementos:
146                 self.dic_elementos[i] = {'mol': 0.0, 'mol_fin': 0.0, }
147             for j in self.v_elementos:
148                 for i in self.v_especies:
149                     self.dic_elementos[j]['mol'] += \
150                     (self.dic_especies[i]['molesIniciales'])*float(
151                                             self.dic_especies[i]['vector'][j])
152             for i in self.v_elementos:
153                 self.v_abundancia += [self.dic_elementos[i]['mol']]
154             self.v_abundancia = array(self.v_abundancia)

155         def contempladas(self):
156             '''Muestra las especies que se han conteplado para los cálculos'''
157             if not self.dic_especies.keys() == []:
158                 print '\n\t> Las especies contempladas:\n'
159                 for i in self.dic_especies.keys():
160                     print '\t',i,
161             else:
162                 print '\n\t> ¡Ninguna especie se ha contemplado!'
163             print '\n'

164         def imprime_matriz_1(self):
165             print '\nEspecies:\t', self.v_especies
166             print 'Elementos:\t', self.v_elementos
```

```
167             print 'Matriz Fórmula:'
168             print self.m_formula

169         def imprime_matriz_2(self):
170             print '\nMatriz Fórmula:\n'
171             sys.stdout.write('\t')
172             for i in self.v_especies:
173                 print '\t',i,
174             print '\n'
175             for j in self.v_elementos:
176                 print '\t',j,'\t',
177                 for i in self.v_especies:
178                     print self.dic_especies[i]['vector'][j],'\t',
179                 print '\n'

180         def mol_iniciales(self, especie):
181             '''El objetivo es mostrar las moles iniciales almacenadas
182             '''
183             return self.dic_especies[especie]['molesIniciales']

184         def mol_elemento(self, elemento):
185             return self.dic_elementos[elemento]['mol']

186         def vector(self, especie):
187             return self.dic_especies[especie]['vector']

188         def __cambios(self):
189             '''Modifica los datos de entrada
190             '''
191             respuesta0 = raw_input('¿Desea cambiar algún dato? » ')
192             if not respuesta0:
193                 print '¡Escriba algo!'

194             if respuesta0.lower() in ('s','si'):
195                 dic = {1: 'Agregar especies',
196                        2: 'Descartar especies',
197                        3: 'Cambiar temperatura',
198                        4: 'Cambiar presión',
199                        5: 'Cambiar número de moles',
200                        6: 'No cambiar datos'}
201                 for i in dic:
202                     print str(i) + '. ' + dic[i]
203                 try:
204                     respuesta1 = int(re.findall('\d+',raw_input(
205                                                 '\nSeleccione una opción: '))[0])
206                     if respuesta1 == 1:
207                         self._especies_cadena()
```

```
208                     self.__inicializa()
209                 elif respuesta1 == 2:
210                     self.contempladas()
211                     print 'Escriba las especies que desea eliminar'
212                     self._especies_cadena(0)
213                     self._especiesAlimentadas = \
214                                             self.__cadenaAlimentada.split()
215                     for i in self._especiesAlimentadas:
216                         try:
217                             del self.dic_especies[i]
218                         except KeyError:
219                             print 'La especie ' + i + ' no es válida'
220                     self.__inicializa()

221                 elif respuesta1 == 3:
222                     self._temperatura()
223                     self.RT = self.T*R
224                 elif respuesta1 == 4:
225                     self._presion()
226                 elif respuesta1 == 5:
227                     respuesta2 = raw_input('Escriba la especie: ')
228                     self.__mol_inicial(respuesta2)
229                     self._molesDeElemento()
230                 elif respuesta1== 6:
231                     pass
232                 else:
233                     print '\n\t> ¡No seleccionó una opcion válida!'
234             except IndexError:
235                 print '\n\t> ¡Selección inválida!'

236         elif respuesta0.lower() in ('n', 'no'):
237             pass
238         else:
239             print '\t\t>Debe responder si o no'

240     def completa_sistema(self, atributo):
241         lista = []
242         for especie in self.v_especies:
243             lista += [self.dic_especies[especie][atributo]]
244         self.sistema.fases.actualiza(self.v_especies, lista, atributo)

245 if __name__ == '__main__':
246     A = Atributos()
```

A.11 Módulo `miscelanea_1`

```
1       # -*- coding: cp1252 -*-
2       from Numeric import array, dot, ones, transpose
3       from miscelanea_2 import *
4       from termoMiscelanea_2 import vec_g
5       from LU import espacioNulo

6       __all__ = ['MolesIniciales', 'Ordenador']

7       '''Deberán definirse

8       m_formula       matriz fórmula
9       m_atomica       matriz atómica
10      v_abundancia    vector de abundancia de especies
11      v_g0        vector de potencial químico estándar
12              de cada una de las especies
13      v_especies      vector con el nombre de las especies
14              participantes
15      v_elementos     vector con el nombre de los elementos
16              presentes
17      numEspecies     escalar con el número de especies
18      numElmentos     escalar con el número de elementos
19      temperatura     temperatura
20      presion     presón
21      R           8.314
22      RT          R*temperatura
23      p
24      q
25      '''

26      class MolesIniciales:

27          def archivoV_nPL(self):
28              '''Este método, escribe e importa el módulo vec_nPL.

29              En vec_n0 se resuelve el problema de programación lineal:

30                  mín! = n^T*g^0  sujeto a An=b, n>=0

31              donde g^0 es el vector potencial estándar de las especies del sis-
32              tema,  n^T es el vector de abundancia de especies transpuesto, A es
33              la matriz fórmula, b es el vector de abundancia de lementos.

34              Específicamente, de vec_nPL se obtiene v_nPL que se incorpora, indi-
35              rectamente,  al algoritmo VCS como las moles iniciales, v_n0.
```

```
36              Deben estar definidos:
37                  numElementos, numEspecies, numRestricciones, m_formula,
38                  v_abundancia, v_g0.
39              '''

40              #f = open( './__init__.py', 'w')
41              #f.close()
42              f = open( './vec_nPL.py', 'w')
43              f.write('from scipy.optimize.cobyla import fmin_cobyla\n')
44              f.write('from Numeric import ones, dot\n')
45              f.write('__all__=["n_ini"]\n\n')
46              f.write('n = ones(' + str(self.numEspecies) + ', "d")\n')
47              f.write('v_g0 = ')
48              f.write(str(self.v_g0) + '\n\n')
49              f.write('def energia(n, v_g0=v_g0):\n')
50              f.write('\treturn dot(n, v_g0)\n\n')
51              for  i in range(self.numElementos):
52                  f.write('r'+ str(i)+ ' = lambda n: ' )
53                  for j in range(self.numEspecies):
54                      if j != 0: f.write( '+' )
55                      f.write( str(self.m_formula[i,j]) + '*n[' + str( j ) +  ']')
56                  f.write( '-' + str(self.v_abundancia[i])+ '\n')
57                  f.write('r'+ str(self.numElementos + i ) + ' = lambda n:  -r' +\
58                                                          str(i) + '(n)\n'  )
59              for i in range(self.numEspecies):
60                  numSiguiente = 2*self.numElementos + i
61                  f.write( 'r'+ str(numSiguiente)+ ' = lambda n: n[' + str(i) + \
62                                                                  ']\n' )
63              f.write('\nv_nPL = fmin_cobyla( energia, n, [' )
64              self.numRestricciones = 2*self.numElementos +  self.numEspecies
65              for i in range( self.numRestricciones ):
66                  f.write( 'r' + str(i) )
67                  if (i != self.numRestricciones - 1):  f.write(',')
68              f.write('] )\n')
69              f.close()

70              from vec_nPL import v_nPL
71              self.v_nPL = array(v_nPL, 'd')

72          def ordenaV_nPL(self):
73              '''Arregla v_npl de acuerdo al procedimientos de PL.

74              El procedimiento de programación lineal, PL, computado en el método
75              __archivoV_nPL genera v_nPL. v_nPL es de uno de los dos tipos
76              siguientes:

77              1. C valores positivos ( C = rango(A) ) siendo el resto cero.
```

```
78              2. Menos de C valores positivos siendo el resto cero.

79              En el primer caso v_n0 se establece como los valores C positivos
80              de v_nPL.
81              En el segundo v_n0 se establece con tomando tomando el mayor valor
82              de v_nPL
83              C veces. En ambos casos los valores después de C se establecen
84              iguales a 1.0e-15.

85              __vec_n0_Modificado genera una v_nPL modificado y un vector con
86              los índices de permuteción de las especies.

87              Deben estar definidos:
88                  rango, numEspecies, v_nPL
89              '''
90              self.v_nPL = infimo(self.v_nPL)
91              self.v_n = self.v_nPL.copy()

92              self.sistema.fases.actualiza(self.v_especies, self.v_n)
93              #correIndices(self.q, self.v_nPL)
94              self.q = AI(self.q, self.v_n, self.rango)
95              # De clase Ordenador
96              self.reordenarArreglos()

97          def vec_n0(self):
98              '''Genera el vector moles iniciales

99              Requiere el vector generado por __ordenaV_PL junto con el vector de
100             permutación, el vector avance de reacción, la matriz N del espacio
101             nulo, y el parámetro de etapa.
102             '''

103             self.v_n = vec_nNuevo(self.v_nPL, self.v_xi, self.m_nula, \
104                             self.rango, self.sistema,\
105                             self.v_especies, self.RT, self.presion)

106     class Ordenador:
107         '''Establece los arreglos en el nuevo orden p q

108         Los arreglos ordenados:
109         m_formula, v_g0, v_especies, v_elementos, v_abundancia,
110         v_nPL, v_n, v_g
111         '''
112         def ordenaV_n(self):
113             '''Checa si el orden de las especies es correcto
```

```
114             En cada iteración mantiene las especies, con mayor número de
115             moles, al principio de la lista.
116             '''

117     ##        qq =  self.q[:]
118             self.q = AI(self.q, self.v_n, self.rango)
119             if self.q != range(self.numEspecies):
120                  self.reordenarArreglos()
121                  # Recalcula m_nula de ser necesario
122                  (self.m_nula, self.rango, self.gle,\
123                       self.p, self.q, self.m, self.n) = \
124                       espacioNulo(self.m_formula, self.q)
125                  self.calculosDeN += 1
126                  self.m_nulaT = transpose(self.m_nula)
127                  self.reordenarArreglos()

128         def reordenarArreglos(self):
129             if self.q != range(self.numEspecies):
130                 self.v_especies, self.v_g0, self.v_nPL, self.v_n = \
131                                                         arreglaVectores(\
132                     self.q, self.numEspecies,[self.v_especies, self.v_g0,\
133                                                 self.v_nPL, self.v_n])

134                 self.m_formula = arreglaMatriz(self.m_formula, self.p, \
135                                       self.q, self.numElementos,\
136                                       self.numEspecies)

137                 try: self.v_g = arreglaVector(self.q,self.numEspecies,self.v_g)
138                 except AttributeError: pass

139                 self.q = range(self.numEspecies)    # Reinicia la permutación q

140             if self.p != range(self.numElementos):
141                 self.v_elementos, self.v_abundancia = \
142                           arreglaVectores(self.p, self.numElementos,\
143                                  [self.v_elementos, self.v_abundancia])
144                 self.p = range(self.numElementos)   # Reinicia la permutación p
```

A.12 Módulo `miscelanea_2`

```
1       '''Módulo de miscelánea VCS '''

2       ##from scipy.optimize.cobyla import fmin_cobyla
3       ##from scipy.optimize.lbfgsb import fmin_l_bfgs_b
```

```
4       ##from scipy.optimize import fminbound
5       from Numeric import dot, exp, sqrt
6       from Numeric import zeros, array
7       from Scientific.Functions.FirstDerivatives import DerivVar
8       from termoMiscelanea_2 import vec_g
9       from constantes import R

10      __all__ = ['Omega', 'AI', 'correIndices', 'infimo', \
11             'vec_nNuevo', 'arreglaMatriz', 'arreglaVector',\
12             'arreglaVectores', 'creaDiccionario', 'ordena_de_acuerdo_a']

13      class Omega:
14          '''Genera el parámetro de etapa
15          '''
16          def __init__(self, fun, x, delta_x , sistema, v_especies, RT, presion):
17              self.fun = fun
18              self.x = x
19              self.delta_x = delta_x
20              self.sistema = sistema
21              self.v_especies = v_especies
22              self.T = RT/R
23              self.presion = presion
24              self.__principal()

25          def __fun_1(self, w):
26              der_x = []

27              for i in range(len(self.x)):
28                  der_x += [DerivVar(self.x[i] + w * self.delta_x[i],i)]
29              self.sistema.fases.actualiza(self.v_especies, der_x, entrada='mol')

30              par_fun = self.fun(self.sistema,self.T,self.v_especies,self.presion)
31              print '======',par_fun
32              sum_par_fun = 0.0

33              cont = 0
34              for i in par_fun[1]:
35                  sum_par_fun += i * self.delta_x[cont]
36                  cont +=1

37              return sum_par_fun

38          def __fun_2(self, w):
39              var_w = DerivVar(w)
40              der_x = []

41              for i in range(len(self.x)):
```

```
42                 der_x += [self.x[i] + var_w * self.delta_x[i]]

43             self.sistema.fases.actualiza(self.v_especies,der_x,entrada='mol')
44             par_fun = self.fun(self.sistema, self.T, self.v_especies, self.presion)
45             self.sistema.fases.actualiza(self.v_especies, self.x, entrada = 'mol')

46             g_i=dot(par_fun,der_x)

47             return g_i[1][0]#par_fun[0]#[1][0]

48         def __principal(self):
49             fw0 = self.__fun_2(0.0)
50             fw1 = self.__fun_2(1.0)
51             if fw1 > 0: self.w = fw0/(fw0 - fw1)
52             else: self.w = 1.0

53     def vec_nNuevo(v_n, v_delta_xi, m_nula, rango, sistema, v_especies, RT,\
54                    presion, no_ceros, v_deltaGj):

55         v_delta_n = dot(m_nula, v_delta_xi)
56         parametroEtapa = Omega(vec_g, v_n, v_delta_n, sistema,\
57                                v_especies, RT, presion)
58         v_n += parametroEtapa.w * v_delta_n
59         cont = 0

60         for i in v_n:
61             if (i < 0) and sistema.fases.especie_en_fase_multicomponente(\
62                                                             v_especies[cont]):
63                 v_n[cont] = 1.e-25
64             cont += 1

65         return v_n, no_ceros

66     def AI(indices, arreglo_patron, rango):
67         '''Corre los índices de un arreglo atendiendo el rango.
68         '''
69         arreglo = list(arreglo_patron)
70         arreglo.sort()
71         arreglo.reverse()
72         maxR = arreglo[rango]
73         j = 0
74         for i in range(len(arreglo_patron)):
75             if arreglo_patron[i] > maxR:
76                 c = indices[i]
77                 del indices[i]
78                 indices[j:j] = [c]
79                 j += 1
```

```
80          return indices

81      def correIndices(indices, arreglo_patron, condicion=1.e-25):
82          '''Corre los índices de un arreglo atendiendo la condición.
83          '''
84          j = 0
85          for i in range(len(arreglo_patron)):
86              if arreglo_patron[i] > condicion:

87                  c = indices[i]
88                  del indices[i]
89                  indices[j:j] = [c]
90                  j += 1
91          return indices

92      def infimo(arreglo, condicion = 1.e-25, infimo = 1.e-25):
93          '''Fija 'infimo' cuando no se cumple la condición
94          '''
95          for i in range(len(arreglo)):
96              if arreglo[i] > condicion: pass
97              else: arreglo[i] = infimo
98          return arreglo

99      def arreglaMatriz(matriz, p, q, n, m):
100         m_ = zeros((n,m), 'd')
101         for i in range(n):
102             m_[i] = matriz[p[i]]
103         matriz = m_.copy()
104         for i in range(n):
105             for j in range(m):
106                 m_[i,j] = matriz[i,q[j]]
107         return m_

108     def arreglaVector(q, m, vector):
109         v = []
110         for i in range(m):
111             v.append(vector[q[i]])
112         if type(vector) == type([]):
113             return v
114         elif type(vector) == type(array([])):
115             return array(v, 'd')
116         else:
117             print 'error no contemplado en arregla vector'

118     def arreglaVectores(q, m, vectores):
119         for j in range(len(vectores)):
120             vectores[j] = arreglaVector(q, m, vectores[j])
```

```
121         return vectores

122     def creaDiccionario(claves, valores):
123         dic = {}
124         cont = 0
125         for clave in claves:
126             dic[clave] = valores[cont]
127             cont += 1
128         return dic

129     def ordena_de_acuerdo_a(patron, lista):
130         '''Ordena "lista" siguiendo el orden de los elementos de "patron"

131         Para una correcta ordenación se asume que:
132             1. Tanto "patron" como "lista" no tienen elementos repetidos"
133             2. Todos los elementos de "lista" se encuentran en "patron"
134             3. En consecuencia, "len(lista)" es menor o igual a "len(patron)"
135         '''
136         lista_ordenada = []
137         for i in patron:
138             if i in lista:
139                 lista_ordenada += [i]
140         return lista_ordenada
```

A.13 Módulo `parametros_interaccion`

```
1       # -*- coding: cp1252 -*-

2       #parámetros de interacción
3       from Numeric import exp
4       from constantes import R

5       ##########################################################################
6       #sistema
7       T = 1873.0

8       elementos = ['Fe','Al','C','Ca','Si']

9       x={
10          'Fe': 0.80,
11          'Al': 0.04,
12          'C' : 0.08,
13          'Ca': 0.02,
14          'Si': 0.06
```

```
15          }

16      ab_henriano_std = {
17          'Al': {'a': -71097.93, 'b': 12.88606   },
18          'C' : {'a': 17234.09,  'b': -14.35763  },
19          'Ca': {'a': -39398.16, 'b': 84.8902    },
20          'Si': {'a': -131496.21,'b': 15.33027   }
21          }

22      ##parametros = ['AlAl','AlC','AlCa','AlSi','CC',
23                      'CCa','CSi','CaCa','CaSi','SiSi']
24      ab_parametro ={
25          'AlAl' :{'a': 58577.63 ,'b':28.5988    },
26          'AlC'  :{'a': 77067.07 ,'b':0.0    },
27          'AlCa' :{'a': -117222.0,'b':0.0    },
28          'AlSi' :{'a': 108999.6 ,'b':0.0    },
29          'CC'   :{'a': 199310.2 ,'b':0.0    },
30          'CCa'  :{'a': -246024.0,'b':0.0    },
31          'CSi'  :{'a': 155796.9 ,'b':-3.55822   },
32          'CaCa' :{'a': 0.0      ,'b':0.0    },
33          'CaSi' :{'a': -167236.0,'b':0.0    },
34          'SiSi' :{'a': 33470.55 ,'b':85.2144    }
35          }

36      ##########################################################################

37      class Actividades:
38          '''Calcula la actividad química de una fase

39          multicomponente de acuerdo al modelo de parámetros de interacción.
40          La clase pedirá la indiceción del componenete solvente, los demás
41          componenetes se consideran solventes. Se requieren los diccionarios:
42              1. x, composición (componente:fracción_mol)
43              2. ab_henriano_std
44              3. ab_parametro
45          donde x es el diccionario (componenete:fracción_mol)
46          ab_henriano_std es el dicionario de los coeficientes, a, b, para
47              calcular los coeficientes henrriano estándar de cada componente, X,
48              RT*ln gama^o_X = a + b*T
49          ab_parametro es el dicionario de los coeficientes, a, b, para calcular
50              los parámetros de interacción 'épsilon' de cada 2 componetes, X, Y,
51              RT*epsilon_XY = a +b*T
52          '''

53          def __init__(self, solvente, temperatura, x,\
54                       ab_henriano_std=ab_henriano_std, ab_parametro=ab_parametro):
55              self.solvente = solvente
```

```
56              self.T = temperatura
57              self.x = x
58              self.x_soluto = x.copy()
59              del self.x_soluto[solvente]
60              self.ab_henriano_std = ab_henriano_std
61              self.ab_parametro = ab_parametro
62              self.calcula_actividades()

63          def log_henriano_std(self, a, b):
64              '''Calcula el coeficiente henriano estándar,

65              dados los parámetros 'a', 'b' y la temperatura, 'T',
66              apartir de la expresión:
67              RT*ln gama^o = a + b*T'''

68              return (a + b*self.T)/(R*self.T)

69          def epsilon_ab(self, a, b=0):
70              '''Calcula el parámetro de interacción 'épsilon'

71              de un par de elementos, dados los parámetros
72              'a', 'b' y la temperatura 'T' e
73              '''
74              return (a + b*self.T)/(R*self.T)

75          def log_henriano_solvente(self):
76              '''Calcula el logaritmo del coeficiente henrriano del solvente

77              Realiza las combinaciones de elementos para los parámetros
78              de interacción devuelve un diccionario con las combinaciones y el
79              número de repeticiones
80              '''
81              combinacion_elementos = {}
82              epsilon = {}
83              suma = 0.0
84              for i in self.x_soluto:
85                  for j in self.x_soluto:
86                      #asegura que los elementos se encuentren siempre en
87                      #el mismo orden
88                      if i < j: h = i+j
89                      else    : h = j+i

90                      epsilon[h] = self.epsilon_ab(self.ab_parametro[h]['a'],\
91                                                   self.ab_parametro[h]['b'])
92                      suma += epsilon[h]*self.x_soluto[i]*self.x_soluto[j]

93                      try:    combinacion_elementos[h] += 1
```

```
94                      except KeyError: combinacion_elementos[h] = 1

95              self.log_gama_solvente = -0.5*suma
96              self.epsilon_sis = epsilon
97              self.combinacion_elementos = combinacion_elementos

98          def log_henriano_solutos(self):
99              self.log_gama_solutos = {}
100             for i in self.x_soluto:
101                 self.log_gama_solutos[i] = \
102                     self.log_henriano_std(self.ab_henriano_std[i]['a'], \
103                                           self.ab_henriano_std[i]['b'])\
104                     + self.log_gama_solvente
105                 for j in self.x_soluto:
106                     if i < j: h = i+j
107                     else    : h = j+i
108                     self.log_gama_solutos[i] += self.epsilon_sis[h]*x[j]

109         def calcula_actividades(self):
110             self.log_henriano_solvente()
111             self.log_henriano_solutos()
112             self.actividad = {}
113             self.actividad[self.solvente] = self.x[self.solvente]*exp(\
114                                                     self.log_gama_solvente)
115             for i in self.x_soluto:
116                 self.actividad[i] = self.x_soluto[i]*exp(\
117                                                 self.log_gama_solutos[i])

118         def maximoElemento(self, dic):
119             '''Halla la clave con el máximo valor del diccionario dic'''
120             max = 0
121             for i,j in dic.iteritems():
122                 if j > max:
123                      max = j
124                      clave = i
125             return clave

126         def combinaciones(self,n,r=2):
127             '''Halla el número de parámetros de interación necesarios
128             para el sistema

129             Halla el número de combinaciones de n objetos tomados de r en r
130             '''
131             a,b = 1, 1
132             c = n-r
133             if n in [1,0]: return 0
134             if r < c:pass
```

```
135             else: r = c
136             for i in range(r):
137                 a *= n-i
138                 b *= i+1
139             return a/b

140     if __name__ == '__main__':
141         F = Actividades('Fe', T, x, ab_henriano_std, ab_parametro)

142         for i in F.actividad:
143             print 'a_'+ str(i)+ '\t>>\t', F.actividad[i]
```