VOLUMEN 28
Verano de la Ciencia XXIX
ISSN 2395-9797
www.jóvenesenlaciencia.ugto.mx

# Desarrollo de programas de Python de uso educativo para los PE's de Ingeniería Química Sustentable e Ingeniería Física

Barrientos-Alvarez Luis Carlos[1], García-Paredes Joel Alejandro[2], Ley-Soto Brandon Jesús[1], Valdivia-Sánchez Felipe Alejandro[1], Reyes-Aguilera José Antonio[1], Armas-Pérez Julio César[1]

[1]Universidad de Guanajuato, División de Ciencias e Ingenierías, Depto. de Ingenierías Química, Electrónica y Biomédica. León, Gto. México [lc.barrientosalvarez@ugto.mx],[b.leysoto@ugto.mx], [fa.valdiviasanchez@ugto.mx],[ja.reyes@ugto.mx],[jc.armas@ugto.mx][1]
[2]Universidad de Guanajuato, División de Ciencias e Ingenierías, Depto. de. Ingeniería Física. León, Gto. México [ja.garciaparedes@ugto.mx][2]

## Resumen

Se elaboraron códigos en lenguaje Python con fines educativos para abordar problemas de Física Computacional y Métodos Numéricos en el área de Física, y códigos de Procesos de Separación, Ingeniería de Calor y Termodinámica Química, que permitieron abordar problemas clásicos de la Ingeniería Química. Los códigos se elaboraron por medio de Google Colab, y fueron publicados en repositorios creados para cada tópico en la plataforma GitHub con licencia pública general GNU v3.0. Los resultados demostraron que el uso de Python permitió resolver eficazmente la parte matemática de los problemas de Ingeniería Física e Ingeniería Química, y que el uso de este lenguaje favoreció la adecuada presentación de gráficas y tablas relativas a estos problemas educativos.

**Palabras clave**: Python, lenguaje de programación; Ingeniería Química; Física; GitHub; Educación; Habilidades tecnológicas.

## Introducción

En el campo de la enseñanza de la Ingeniería Química y la Física, existen infinidad de métodos y técnicas que los estudiantes, al cursar estos programas educativos (PE's), deben de aprender para ejercer de manera apropiada su profesión. Muchas de estas técnicas y métodos involucran cálculos complejos, repetitivos o dibujo de diagramas.

Los estudiantes de estos PE's frecuentemente se enfrentan a los retos de aplicar estas técnicas y métodos a mano o, en el mejor de los casos, con ayuda de software de hoja de cálculo. Esto provoca que incremente la probabilidad de equivocarse y de consumir tiempo valioso en realizar estas tareas.

Una alternativa para optimizar tiempos de cálculo y reducir considerablemente los errores, es el uso de lenguajes de programación de alto nivel como lo son C, Fortran o C++, así como programas matemáticos como Matlab o Mathematica, en los cuales se pueden codificar para adaptarse a la resolución de múltiples problemas en alguna categoría específica. El problema de los lenguajes de programación de alto nivel es el proceso de aprendizaje y el desarrollo de los códigos que puede requerir bastante tiempo; en el caso de los softwares matemáticos, el costo de estos suele ser prohibitivo para los estudiantes. En los últimos años, la comunidad está empleando el lenguaje de programación denotado Python, el cual presenta muchas ventajas y ha demostrado ser eficiente, fácil de aprender, que puede ejecutarse en muchas plataformas diferentes y ser descargado de manera gratuita.

Una lista corta de las ventajas de usar el lenguaje de Python es:

- Es uno de los lenguajes de programación más utilizados actualmente y que sigue en crecimiento.
- Tiene una sintaxis básica que permite una lectura y compresión fácil de los programas.
- Permite escribir programas con menos líneas de código en comparación de otros lenguajes.
- Hay muchos recursos disponibles en internet para aprender Python.
- Python cuenta con una gran biblioteca estándar que contiene códigos para casi cualquier tarea.
- Tiene una comunidad muy activa alrededor del mundo que prestan su apoyo [1].

Por otro lado, Github[2] es una plataforma que ha crecido de manera exponencial y que ha servido de repositorio de un gran número de proyectos. Esta plataforma para desarrolladores fue creada en 2008, permite almacenar, compartir y

trabajar proyectos con otros usuarios empleando un software de control de versiones de Git, siendo su principal característica el trabajo colaborativo. Las ventajas de usar la plataforma GitHub son:

- Es la plataforma web más usada a nivel mundial entre los desarrolladores software.
- Muchos de los proyectos en la página son públicos y permite a las personas acceder al código, descargarlo y experimentar con él por lo que fomenta el aprendizaje a partir de este.
- Permite que la comunidad haga contribuciones en los proyectos.
- La página web es gratuita, si bien se puede obtener una cuenta de pago no es obligatoria para trabajar con él. La liga de la pagina es https://github.com/

En particular, en la División de Ciencias e Ingenierías de la Universidad de Guanajuato, se imparten diversas asignaturas para los PE's de Ingeniería Química Sustentable e Ingeniería Física donde es fundamental el desarrollo de códigos computacionales. Algunas de estas asignaturas son: Ingeniería de Calor, Procesos de Separación, Termodinámica Química, Métodos Numéricos o Física Computacional, donde los cálculos son complejos, largos y repetitivos. Por ello, se ha propuesto en este proyecto el desarrollo de software en lenguaje Python, capaz de resolver problemas específicos de estas materias y que, adicionalmente, sea de acceso libre para la comunidad estudiantil a través de la plataforma GitHub.

La licencia que se utilizó para este proyecto fue la licencia GNU GPL V.3 ya que es una licencia de software libre muy utilizada debido que garantiza a los usuarios finales poder ejecutar, estudiar, compartir y modificar el software además protege nuestro trabajo con derechos de autor sin ser tan restrictivo con el usuario final [3].

# Material y Métodos

Se desarrollaron códigos en Python en formato .ipynb de uso educativo en Google Colab para los siguientes temas:

- Procesos de Separación
- Ingeniería de Calor
- Termodinámica Química
- Métodos Numéricos y Física Computacional

Por cada tema se abrió un repositorio en GitHub con licencia pública general GNU v3.0, donde se abrieron subcarpetas o *Branches* para abordar diferentes subtemas. La liga principal de los códigos se encuentra en:

https://github.com/jcarmascodes/

A continuación, se describe cada uno de los subtemas que se desarrollaron para el proyecto:

## Procesos de Separación

En este ámbito se abordaron subtemas de operaciones unitarias como absorción, destilación, extracción, evaporación y humidificación. En cada subtema se abrió un archivo del tipo README.md, en el cual en lenguaje LaTeX se añadieron los conceptos básicos para la comprensión de cada operación unitaria, así como ecuaciones fundamentales para cada operación; de igual forma se añadió material audiovisual para apoyar en la mejor comprensión de cada tópico. Las gráficas se hicieron con ayuda de *pyplot* de la biblioteca *matplotlib.*

En las operaciones unitarias de separación más tradicionales, se utilizó el cálculo de etapas entre una línea de operación y una línea de equilibrio [4]. Para realizar en Python esta construcción de etapas debió de construirse una línea de equilibrio, cuyos datos se obtuvieron a través de información experimental o modelos termodinámicos. Para obtenerse una forma funcional que permita evaluar la línea de equilibrio Python posee la herramienta *polyfit* de la biblioteca *Numpy* que ajustó el conjunto de datos a un polinomio de un determinado grado, esto se observa en la Figura 1.

```
H_eq=[13,16,19,23,27,31,36,42,48,55,64,74]
T_eq=[50,55,60,65,70,75,80,85,90,95,100,105]

alfa_coefx=np.polyfit(T_eq,H_eq,10) #Se ajustan los puntos por medio de un ajuste polinómico
```

*Figura 1. Ajuste de datos a un polinomio de grado n.*

Es así como se construyeron todas las líneas de equilibrio de este trabajo.

La otra parte fundamental, fue el dibujo de las líneas de operación las cuales se plantean a partir de balances másicos; en este caso las líneas se dibujaron de manera particular, dependiendo de la operación de separación. Para el caso de absorción, esta se construyó con ayuda de la Ecuación 1:

$$y = \frac{L_S}{V_E}x + y_S + \frac{L_S}{V_E}x_E \tag{1}$$

La cual se expresó en Python usando valores de *x* propuestos por otra herramienta de *Linspace*, que es herramienta de Numpy, la ejemplificación de esto se observa en la Figura 2.

```
x = np.linspace(0, 0.025, 9)      # 0.0050(1% H=2),   0.00359(1% H=2.8),
mo = LN / VN1                      #Pendiente de la línea de operación
    Yo[i] = mo*x[i] + YA1          #Ecuación de la recta de operación (
```

*Figura 2. Definición de líneas de operación en Python.*

Al tener las 2 líneas planteadas, se procede al dibujo de etapas. La primera etapa comienza a dibujase desde la posición inferior (o superior) de la línea de operación [5].

El procedimiento para el dibujo de etapas consistió en sustituir una concentración inicial de "*y"* en la ecuación o relación funcional de equilibrio para obtener una concentración correspondiente en dicho punto en "*x"*; posteriormente se sustituyó este valor en la ecuación de la línea de operación para obtener un nuevo valor en "*y"*. Este procedimiento se realizó en un ciclo *while* hasta que se alcanzó la concentración final deseada Cada valor obtenido se guardó en un *array*. Para el cálculo de número de etapas, se adjuntó un contador de ciclos. Todas las operaciones por etapas se determinaron de esta misma forma; en el caso de destilación se añadieron 2 ciclos de este tipo, ya que la columna se dividió en 2 secciones (rectificación y agotamiento) [6]. El ciclo *while* se ilustra en la Figura 3.

```
while yp < YAN1 : #Este ciclo permite asignar
    xp = YP[i] / H # A partir de un valor inic
    XP.append(xp)
    yp = mo*XP[i] + YA1 #La fracción en líquid
    YP.append(yp)
    print ('%13.7f %15.7f ' % (XP[i],YP[i]))
    i += 1
print ("i =", i)

NE = i / 2
```

*Figura 3. Definición de un ciclo while para el conteo y dibujo de etapas*

En el caso de destilación, la forma funcional de la línea de operación en la zona de rectificación fue la ilustrada en la Ecuación 2:

$$y_{N+1} = \frac{R_D}{R_D + 1}x_n + \frac{x_D}{R_D + 1} \tag{2}$$

Otras rectas de operación como la de agotamiento en destilación, o las líneas de operación propias de la extracción se ajustaron de forma análoga a las líneas equilibrio y fueron evaluadas con la función de *polyval* de *Numpy* [7].

- *Cálculo del punto pinch en destilación por el Método McCabe-Thiele*

Se buscó determinar el punto pinch; para ello se calculó *q* en función del tipo de alimentación que entra a la columna. Para más información para el cálculo de este valor ir a la literatura de procesos de separación.

Una vez conocido, *q* se dibujó la línea de alimentación ilustrada en la Ecuación 3, asignando valores de *x* a través de la herramienta de *numpy* de *linspace.*

$$y = -\frac{q}{1-q}x + \frac{x_F}{1-q} \tag{3}$$

Una vez conocida la línea de alimentación, se calcula el punto pinch por medio de un proceso iterativo con un ciclo *while* para encontrar el punto donde el valor al evaluar el polinomio de datos de equilibrio con la herramienta de *numpy de polyval* y la línea de alimentación se igualan, esto a través de asignar como criterio de convergencia un valor de tolerancia [6].

- Evaporación

Se diseñó un evaporador de triple efecto para aumentar la concentración de una mezcla acuosa. Como es bien sabido, el proceso de cálculo de evaporador de efecto múltiple es iterativo; para lo que se plantearon ecuaciones de balance a través de suposiciones preliminares.

En base a los valores de caída de temperatura, se calculó la temperatura de ebullición dentro del evaporador, así como la temperatura de saturación dentro de la chaqueta. Lo que permitió plantear un balance de energía para determinar el valor de las corrientes de vapor y líquido en cada segmento del evaporador. Dado que es un evaporador de triple efecto se planteó un balance de energía como un sistema de ecuaciones de 6 incógnitas; para el cuál se usó la biblioteca de *numpy* en la función *linalg.solve* para obtener el valor de los flujos. El uso de esta herramienta se ilustra en la Figura 4.

```
# Definir la matriz de coeficientes A
A = np.array([
     #S         L1      L2  V1  V2  V3
    [lambda1, -L1cp*T1, 0, -H1, 0, 0], #Se propone un sistema de ecuaciones de balance de energía
    [0, -1, 0, -1, 0, 0],
    [0, 1, -1, 0, -1, 0],
    [0, 0, 1, 0, 0, -1],
    [0, L1cp*T1, -L2cp*T2, lambda2, -H2, 0],
    [0, 0, L2cp*T2, 0, lambda3, -H3]
])

# Definir el vector de términos independientes B
B = np.array([-F*Fcp*T0, -F, 0, L3, 0, L3cp*T3])

# Resolver el sistema de ecuaciones
X = np.linalg.solve(A, B)
```

*Figura 4. Uso de calculadoras de sistemas de ecuaciones lineales en Python*

Los valores de entalpía de vapor de agua se introdujeron a mano por el usuario a través de la herramienta *input*. El uso de esta herramienta se ejemplifica en la Figura 5.

```
H1=float(input("Inserte la entalpía del vapor sobrecalentado a T1 y Psat2 (kJ/kg): "))
lambda1=float(input("Inserte el calor latente de vaporización a Psat1 (kJ/kg):"))
```

*Figura 5. Inserción de valores numéricos en Python.*

En base a lo obtenido, se calcula el área de cada evaporador; las cuales deben ser aproximadamente iguales.

Dado que es un proceso iterativo, se introdujo este procedimiento de cálculo dentro de un ciclo *while* el cual concluye cuando el usuario indica que el resultado comple con la exactitud adecuada.

En caso de no tener una exactitud adecuada, se propusieron nuevas caídas de temperatura para forzar a los evaporadores a una determinada área con ayuda de la Ecuación 4 [7].

$$A = \frac{\frac{q_1}{U_1} + \frac{q_2}{U_2} + \frac{q_3}{U_3}}{\Delta T} \tag{4}$$

- Humidificación:

Las operaciones de humidificación requieren de balances de materia y de energía para poder establecer sus líneas de operación (la cual se ajustó a 3 puntos por medio de *polyfit)*. Una vez determinados estos balances puede procederse al cálculo del número de unidades de transferencia y la altura de estas como se muestra en la Ecuación 5 [8, 9].

$$N = \frac{ln[(y_b - y_b)/(y_a - y_a)]}{ln[(y_b - y_a)/(y_b - y_a)]} \quad (5)$$

Dado que esta tiene una relación funcional compleja, fue más sencillo definirla como una función y llamarla en el código cuando fue necesario. La definición y llamado de funciones se observa en la Figura 6.

```
def NOy(yb,yT,yb_eq,yT_eq):
  return (yb-yT)/(((yb-yb_eq)-(yT-yT_eq))/log(((yb-yb_eq))/((yT-yT_eq))))

N_Sup=NOy(H[0],H[1],H_eq[0],H_eq[1]) #Se calcula el no. de unidades de transferencia
```

*Figura 6. Definición de funciones en procesos de separación.*

## Ingeniería de Calor

En el repositorio de Ingeniería de Calor se encuentran los códigos tomados del curso de los temas de diseño de intercambiadores de calor de tubos y coraza, condensadores, rehervidores tipo termosifón. En cada carpeta se encuentra un archivo README.md que contiene los conceptos básicos y las ecuaciones utilizadas para el diseño de estos equipos, además se encuentran códigos de ejemplo que contiene las fórmulas utilizadas.

Aunque estos equipos sean distintos, en todos los equipos se buscó el coeficiente de transferencia global de energía que requiere el equipo para un proceso usando la Ecuación 6. En todos se calculó el calor transferido (Ecuación 7) entre las sustancias y con la Ecuación 8 de diseño de calor se pudo obtener el área de transferencia total de calor del equipo. [10,11]

$$U = \frac{1}{\frac{D_o}{D_i h_i} + \frac{D_o}{D_i h d_i} + \frac{x D_o}{k_w D_w} + \frac{1}{h d_o} + \frac{1}{h_o}} \quad (6)$$

```
U0 = 1 / (Do1 + Do2 + xDo + hdo + hoinv)
print('+------------------------------+')
print('+   Uo = %.3f' %U0,'BTU/ft\u00b2 h°F   +')
print('+------------------------------+')
```

Donde:

$\frac{1}{h_o}$: Resistencia al flujo de calor que existe en el fluido desde la parte laminar hasta la parte turbulenta en la coraza.

$\frac{1}{h_{do}}$: Resistencia al flujo de calor entre la capa de los depósitos en el exterior de los tubos.

$\frac{D_o}{h_i Di}$: Resistencia al flujo de calor en el fluido desde la parte laminar hasta la parte turbulenta en el interior de los tubos.

$\frac{xD_o}{k_w D_w}$: Resistencia que ofrece el material del tubo del intercambiador.

$\frac{D_o}{D_i h_{di}}$: Resistencia al flujo de calor por la capa de suciedad o incrustación en el interior del tubo.

$$Q = WC_P\Delta T \quad (7)$$

```
# Calculando el calor transferido, es decir el calor perdido #
Q = MC*CpC*(T1-T2)
```

*Figura 8. Calor transferido en un intercambiador de calor.*

$$A = \frac{Q}{U\Delta T_{ml}} \quad (8)$$

```
AT=Q/(UD*DTMLc) #Área total de transferencia  (ft^2)
print(f'Area total={AT} ft\u00b2')
```

*Figura 9. Área total de transferencia de calor de un intercambiador de calor.*

## Métodos Numéricos

En el repositorio de Métodos Numéricos se encuentran los principales códigos tomados del curso que se imparte en la DCI, UG. El objetivo de la UDA (Unidad De Aprendizaje) es que el estudiante conozca diferentes métodos que se pueden utilizar para resolver problemas en los que no se conocen soluciones analíticas. Se debe tener en claro que las soluciones en los métodos numéricos generalmente son aproximaciones, por lo que tienen un margen de error; lo que se busca es hacer el error más pequeño posible.

Se hizo una recopilación de códigos en Python para esta UDA, que pertenecen a algunos estudiantes de cursos anteriores, los cuales abarcaron los temas principales del curso. Un ejemplo fue el algoritmo de raíces de polinomios, como el método de la secante (Ecuación 9 y 10), primero se eligió la función a evaluar [12].

$$f_2(x) = \sqrt{x} - \cos x \qquad (9)$$

```
def funcion2(x):
    return (sqrt(x) - cos(x))
```

*Figura 10. Función escrita en código Python.*

El método de la secante está dado por:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \qquad (10)$$

```
sol= root_scalar(funcion2,method='secant', x0 = 0, x1 = 1)

print ("\n\nRESULTADOS MÉTODO SECANTE CON ROOTS")
print("La raíz es: {} \n".format(sol.root))
print("Datos extra: \n {}".format(sol))
```

*Figura 11. Algoritmo del método de la secante usando librería Roots.*

El código completo se encuentra en:

https://github.com/jcarmascodes/Metodos-numericos/blob/main/Raices-de-Polinomios/Metodos-abiertos.ipynb

Otro ejemplo es la interpolación de Lagrange. La interpolación se utiliza para construir una función que englobe todos los datos experimentales que se obtienen de una medición. Entonces el algoritmo de la interpolación de Lagrange se definió con la ayuda de las Ecuaciones 11 y 12 como:

donde:

$$f_n(x) = \sum_{i=1}^{n} L_i(x) f(x_i) \qquad (11)$$

$$L_i(x) = \prod_{j_{j\neq i}=0}^{n} \frac{x - x_j}{x_i - x_j} \qquad (12)$$

La gráfica de este código se encuentra en el apartado de Resultados. El código completo puede revisarse en

https://github.com/jcarmascodes/Metodos-numericos/blob/main/Ajuste-de-curvas/Interpolacion/Lagrange.ipynb

## Física computacional

Física computacional no es un curso que se imparte de manera obligatoria en la DCI. Sin embargo, para algunos estudiantes es atractivos tomar la UDA como optativa. En este curso los contenidos son muy específicos y los códigos que se encuentran en este repositorio se pueden definir como una aplicación de Métodos Numéricos a la Física. Es importante mencionar que esta UDA es adecuada para los estudiantes de la Licenciatura en Ingeniería Física a partir de la sexta

inscripción ya algunos algoritmos requieren conocimientos elevados correspondientes a temas q de Mecánica cuántica y Mecánica estadística.

Un ejemplo también abordado es el código de Proyectiles [13], donde se puede apreciar la diferencia entre un objeto lanzado con resistencia de aire a comparación de un tipo parabólico ideal, donde no se considera la resistencia del aire. La gráfica de este código se encuentra en el apartado de Resultados. El código completo puede revisarse en https://github.com/jcarmascodes/Fisica-computacional/blob/main/Mecanica-clasica/Proyectiles.ipynb

Otro ejemplo que es muy interesante es el Modelo Ising 2D de Mecánica Estadística [14, 15], en este modelo se realiza una simulación que conduce a la transición de fase de un sistema físico interactuante. Esto es importante porque no se estudia en Licenciatura, pero abre las puertas a un estudiante que desea cursar la maestría en Física. Se puede revisar el código completo en https://github.com/jcarmascodes/Fisica-computacional/blob/main/Mecanica-estadistica/MonteCarloIsing2D.ipynb

## Termodinámica química

En esta área se concentró en los subtemas de las ecuaciones de estado (ecuación de Van der Waals, Redlich-Kwong, Soave, Peng-Robinson [16] y el cálculo teórico de los coeficientes de actividad para diagramas de equilibrio líquido-vapor (ecuación de NRTL y ecuación de Wilson). En los apartados correspondientes a cada ecuación de estado se resuelve esta y se realizael cálculo teórico de los coeficientes de actividad, se presentan los archivos de *jupyter notebook*, ubicados en cada carpeta, con los resultados correspondientes y se realiza la explicación del procedimiento para realizar los diferentes cálculos, en la redacción de esta sección se emplea LaTeX para la escritura de las ecuaciones.

Para cada ecuación de estado abordada, se realizaron los códigos considerando un procedimiento similar, pero tiene la versatilidad para trabajar con una sustancia es pura o una mezcla [16]. A continuación, se describe el funcionamiento del código para cada ecuación de estado, en la primera parte se realizó el cálculo de parámetros en donde utilizó las propiedades de los compuestos puros (depende de cada ecuación de estado). Después, estos parámetros se usaron para resolver la ecuación en su forma de polinomio en términos del volumen molar (V) [16].

```
coeff1_S=1                                          #TERMINO CÚBICO
coeff2_S=-1*((R*T)/P)                               #TERMINO CUADRATICO
coeff3_S=(1/P)*((a_S*alpha_S)-(b_S*R*T)-(P*(b_S**2)))   #TERMINO LINEAL
coeff4_S=-1*((a_S*alpha_S*b_S)/(P))                 #TERMINO INDEPENDIENTE

POL_S=[coeff1_S,coeff2_S,coeff3_S,coeff4_S]
```

$$\overline{V^3} - \left(\frac{RT}{P}\right)\overline{V^2} + \frac{1}{P}\left(a\,\alpha - bRT - Pb^2\right)\overline{V} - \left(\frac{a\alpha b}{P}\right) = 0$$

*Figura Figura 12. La implementación en código python de la ecuación de estado de Soave [15] en forma de polinomio en donde cada color representa cada término de este del mismo. En donde T, es la temperatura; P, presión del sistema; y R, es la constante de los gases ideales*

Después, siguiendo el procedimiento para calcular la raíz del polinomio, en donde para esta finalidad se utilizó la librería *numpy* para utilizar la función *roots*. Con esto se puede determinar el volumen molar, para posteriormente, dependiendo del caso, se fresuelven las ecuaciones de estado, se emplea una función polinomial cuya raíz real se considera el volumen molar de un gas, en el caso de más raíces reales, lo que indica la presencia de dos fases (líquido y gas) [16],fue necesario implementar un código que realice la distinción de los datos resultantes el programa, esto selogra utilizando la función *isreal* de la misma librería. Finalmente, se emplean los datos así obtenidos para obtener la densidad y el factor de compresibilidad del fluido.

```
EPOL_S=np.roots(POL_S)
RR_S=sum(np.isreal(EPOL_S))
if RR_S == 1:
    VM_S=EPOL_S[np.isreal(EPOL_S)].real[0]
    print("VOLUMEN MOLAR DEL FLUIDO (L/mol) = %5.6f"%(VM_S))
elif RR_S == 3:
    VM_S=max(EPOL_S)
    print("VOLUMEN MOLAR DEL FLUIDO (L/mol) = %5.6f"%(VM_S))
```

*Figura 13. La implementación en código python de la obtención de la raíz del polinomio utilizando la librería numpy*

Se generaros los programas para obtener coeficientes de actividad a partir de diagramas de equilibrio líquido-vapor, también calculados, pudiendo determinar su fracción mol y diferentes parámetros para la sustancia pura y mezclas de esta, l [16] [17].

Se describe el funcionamiento del código, y en parte inicial se incluye un panel de ingreso de datos de las sustancias puras (volumen molar, presión de vapor en estado puro [16]), y de los parámetros de interacción binario entre diferentes ecuaciones. Estos datos permitten obtener los parámetros de interacción entre los compuestos que forman la mezcla, con ello se logra también determinar los diagramas de equilibrio líquido-vapor P-xy (cuando la temperatura de la mezcla se mantiene constante [18]) y el diagrama T-xy (cuando la presión de la mezcla se mantiene constante [18]).

El código genera diferentes fracciones mol para cada sustancia en estado puro que esta presente en las mezclas mediante arreglos donde se registra y guardan las diferentes fracciones mol para cada sustancia. En el caso de la generación del diagrama de P-xy, se utilizó un proceso ciclado esto es para utilizar las fracciones mol para compuesto presente en la mezcla binaria que se guardaron en variables, las cuales se van iterando hasta obtener los valores correspondientes de los coeficientes de actividad empleando del tipo de ecuación (NRTL o Wilson [17]) para cada compuesto presente en la mezcla.

```
for i in range(n):
    GAMMA1.append(np.exp((X2[i]**2)*((T_21*((G_21/(X1[i]+(X2[i]*G_21)))**2))+((T_12*G_12)/((X2[i]+(X1[i]*G_12))**2)))))
    GAMMA2.append(np.exp((X1[i]**2)*((T_12*((G_12/(X2[i]+(X1[i]*G_12)))**2))+((T_21*G_21)/((X1[i]+(X2[i]*G_21))**2)))))
```

$$\ln\gamma_1 = x_2^2\left[\tau_{21}\left(\frac{G_{21}}{x_1 + x_2G_{21}}\right)^2 + \frac{\tau_{12}G_{12}}{(x_2 + x_1G_{12})^2}\right] \qquad \ln\gamma_2 = x_1^2\left[\tau_{12}\left(\frac{G_{12}}{x_2 + x_1G_{12}}\right)^2 + \frac{\tau_{21}G_{21}}{(x_1 + x_2G_{21})^2}\right]$$

*Figura 14. La implementación en código python del cálculo de los correspondientes coeficientes de actividad utilizando la ecuación de NRTL [16] mediante un proceso repetitivo dependiendo de la cantidad de pares de fracciones mol se hayan guardado en las variables*

Luego de realizar el cálculo de los coeficientes de actividad se obtuvo la presión de vapor de la mezcla utilizando la ecuación de Raoult para mezclas no ideales, posteriormente se calcularon las fracciones mol en la fase gaseosa. Una vez calculadas todas presiones totales para la mezcla binaria se grafica la fracción mol de la fase líquida de alguno de los componentes junto con la presión total para graficar la línea de puntos burbuja y de punto de rocío [18], se grafica cada uno de los puntos entre las fracciones mol de la fase gaseosa del mismo componente junto con la presión total del sistema; utilizando la librería de *matplotlib* con la función *pyplot*.

Para generar el diagrama de equilibrio de fase líquido-vapor tipo T-xy, en donde la variable de presión del sistema se mantiene constante, se utiliza el mismo procedimiento descrito para el diagrama P-xy; en el caso donde la temperatura es variable, se realiza un cálculo iterativo hasta que la presión total de vapor iguale a la presión del sistema, mediante un proceso de convergencia usando el método de Newton-Raphson que fue implementado en el código.[17]. El código comienza los cálculos a partir de una temperatura supuesta ($T_{sup}$) y luego repite el proceso con una nueva temperatura supuesta utilizando una diferencia de temperatura, por último, fue necesario calcular el error entre la presión calculada y la presión del sistema en donde si se alcanza un mínimo que sea mayor o igual a la tolerancia en la exactitud del cálculo el programa se cierra y continua con las siguientes fracciones mol. [18]

```
ERROR=np.log(PT/P_OBJ)
E_N=np.log(PT_N/P_OBJ)
T_MOD=(T*T_N*(E_N-ERROR))/(T_N*E_N-T*ERROR)
T=T_MOD
```

$$E = E' = \ln\left(\frac{p_{calc}}{p}\right)$$

$$T_{Nueva} = \frac{T_{sup}T'_{sup}(E' - E)}{T'_{sup}E' - T_{sup}E}$$

*Figura 15. La implementación en código python del cálculo de la nueva temperatura en donde utilizando el cálculo del error y de las temperaturas supuestas [17]*

# Resultados

## Procesos de Separación

Como se observa en las Figuras 16 a 21, los códigos de Python fueron capaces de calcular correctamente el número de etapas de equilibrio al hacer diagramas claros de McCabe-Thiele; así como converger en los resultados adecuados para el cálculo del área de evaporadores.

En la mayoría de los ejemplos mostrados, el cálculo fue automático y sólo en algunos de ellos requirió asistencia del usuario para la inserción de datos termodinámicos como datos de entalpía.

El link del repositorio es: https://github.com/jcarmascodes/Procesos-de-Separacion/tree/main

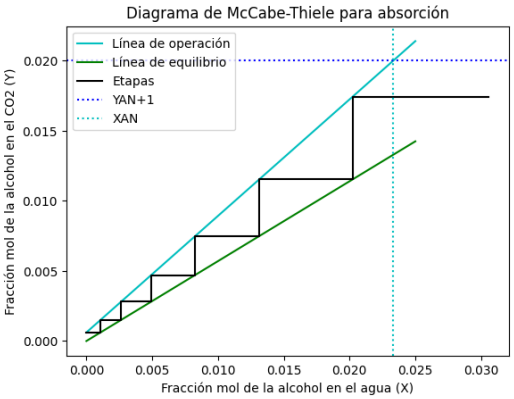


*Figura 16. Diagrama McCabe-Thiele para absorción.*

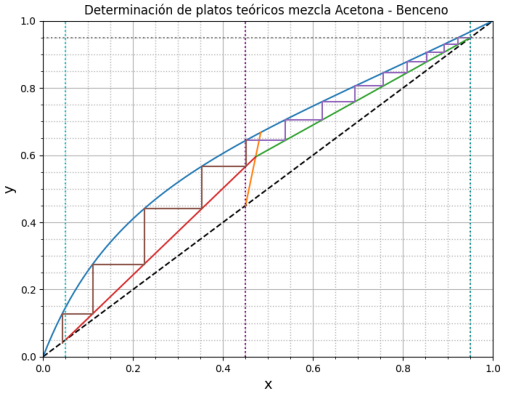


*Figura 17. Diagrama McCabe-Thiele para destilación con 2 alimentaciones.*

```
Tabla de composiciones con colores:
+----+-----------+-----------+-----------+
|    |   A1 (m2) |   A2 (m2) |   A3 (m2) |
+====+===========+===========+===========+
|  0 |   103.268 |   103.161 |   103.288 |
+----+-----------+-----------+-----------+

Si la similitud de las áreas es lo suficientemente parecida teclee 0, de lo contrario oprima 1
Inserte 0 (para finalizar) o 1 (para continuar la iteración)0
```

*Figura 18. Imagen de los resultados de código de Evaporador de Efecto Múltiple*

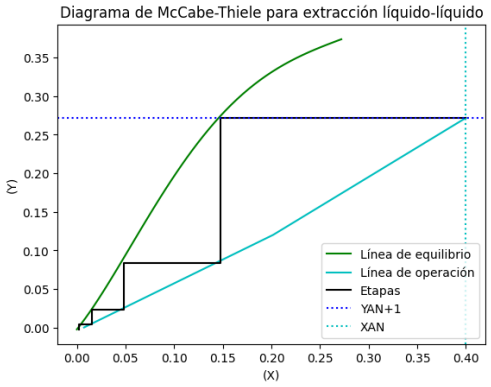


*Figura 19. Diagrama McCabe-Thiele para extracción líquido-líquido*

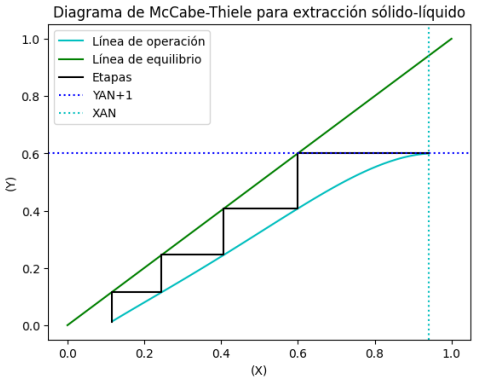


*Figura 20. Diagrama McCabe-Thiele para extracción sólido-líquido*

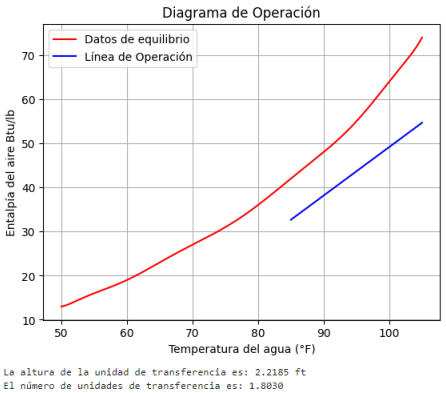


*Figura 21. Diagrama de Operación de una torre de enfriamiento*

## Ingeniería de Calor

En la Tabla 1 se muestra los resultados del área de transferencia de calor, el calor transferido, temperatura logarítmica media, el coeficiente global de transferencia de calor. que necesita el intercambiador de calor de tubos y coroza usando el método de Kern. El resultado del cálculo varía dependiendo del valor escogido global de transferencia de calor por el usuario, factor de corrección de temperatura (F).

El enlace del repositorio es: https://github.com/jcarmascodes/Ingenieria_de_calor

| DESCRIPCIÓN | VALORES | UDS |
|---|---|---|
| Calor transferido (Q) | 6534000.00 | BTU/h |
| Flujo másico en coraza (MC) | 60000 | Lb/h |
| Flujo másico en tubos (MT) | 130680.000 | Lb/h |
| Temp logaritmica media (Tlm) | 156.609 | °F |
| P | 0.333 | --- |
| R | 1.800 | --- |
| F | 0.990 | --- |
| Coef. global de transf de calor (U0) | 34.6. | BTU/ft²h°F |
| Área total de transf de calor (A0) | 1217.369. | ft² |

***Tabla 1.*** *Resultados obtenidos para el intercambiador de tubos y corazas.*

## Métodos Numéricos

A continuación, se muestran los resultados de los códigos utilizados:

```
sol= root_scalar(funcion2,method='secant', x0 = 0, x1 = 1)

print ("\n\nRESULTADOS MÉTODO SECANTE CON ROOTS")
print("La raíz es: {} \n".format(sol.root))
print("Datos extra: \n {}".format(sol))
```

*Figura 22. Código utilizado*.

```
RESULTADOS MÉTODO SECANTE CON ROOTS
La raíz es: 0.6417143708728826

Datos extra:
       converged: True
            flag: converged
 function_calls: 7
      iterations: 6
            root: 0.6417143708728826
```

*Figura 23. Salida del valor buscado*.

Ahora, para el ejemplo de interpolación de Lagrange de 3° orden se tiene el ajuste en los datos tomados en el Lago Platte

| f | i |
|---|---|
| z(m) | T(°C) |
| 0 | 22.8 |
| 2.3 | 22.8 |
| 4.9 | 22.8 |
| 9.1 | 20.6 |
| 13.7 | 13.9 |
| 18.3 | 11.7 |
| 22.9 | 11.1 |
| 27.2 | 11.1 |

*Figura 24. Valores del Lago Platte.*

```
from scipy.interpolate import lagrange
T=[22.8,22.8,22.8,20.6,13.9,11.7,11.1,11.1] #Temperatura en °C
z=[0,2.3,4.9,9.1,13.7,18.3,22.9,27.2] #profundidad en m
f=lagrange(z,T)
```

*Figura 25. Código de la interpolación de Lagrange.*

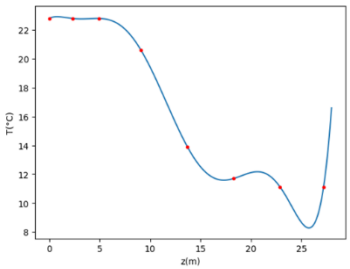


*Figura 26 .Gráfica de resultados.*

## Física computacional

En cuanto a los algoritmos de Física computacional, en la Figura se muestran los resultados del movimiento de proyectiles y del modelo Ising 2D.

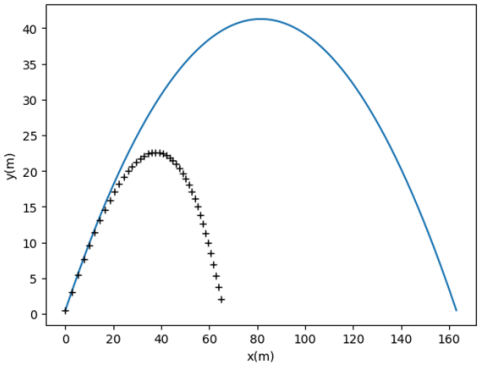


*Figura 27. Gráfica del movimiento de proyectiles con resistencia de aire (+) y un tiro parabólico ideal (azul).*

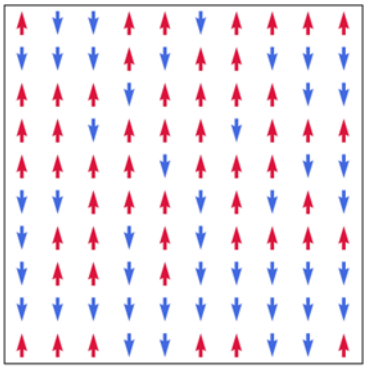

*Figura 28 . Gráfica del modelo Ising 2D con espín hacia arriba y hacia abajo.*

## Termodinámica química

En la Tabla 2, se muestra los resultados que se obtuvieron del programa en el cálculo del volumen molar de una mezcla de gases (metano, etano y propano) utilizando diferentes ecuaciones de estado. Donde también se calcularon los factores de compresibilidad, densidad del fluido, así como la cantidad de moles y masa de la mezcla presentes dentro de un volumen determinado. En la misma tabla se observa la variación de los valores obtenidos del programa, ya sea debido a las diferentes consideraciones matemáticas que hace cada una de las ecuaciones de estado.

En la Figura 29, se muestran los diagramas de equilibrio líquido-vapor de presión-composición (P-xy) y de temperatura-composición (T-xy) realizados con la ecuación de NRTL en donde la mezcla binaria es de acetona-metanol tanto los diagramas. En la Tabla 3, se muestran del extracto de los resultados empleados para obtener el diagrama presión-composición generados con la ecuación de NRTL para la mezcla binaria de acetona-metanol.

El enlace al repositorio es: https://github.com/jcarmascodes/Termodinamica_quimica/tree/main

***Tabla 2.*** *Comparación de los resultados obtenidos entre las ecuaciones de estado para una mezcla de metano-etano-propano a 310 K y 18.91 atm*

| ECUACIÓN DE ESTADO | VOLUMEN MOLAR (L/mol) | DENSIDAD (kg/m^3) | FACTOR DE COMPRESIBILIDAD | CANTIDAD DE MOLES DE SUSTANCIA (mol) | MASA DE SUSTANCIA (g) |
|---|---|---|---|---|---|
| Gas ideal | 1.347774 | 26.725545 | 1.000000 | 1.000000 | 36.020000 |
| Van der Waals | 1.119695 | 32.169480 | 0.830773 | 4.644123 | 167.281294 |
| Redlich-Kwong | 1.076658 | 33.455374 | 0.798842 | 4.829760 | 173.967945 |
| Soave | 1.069920 | 33.666068 | 0.793842 | 4.860176 | 175.063552 |
| Peng-Robinson | 1.048165 | 34.364816 | 0.777701 | 4.961051 | 178.697041 |

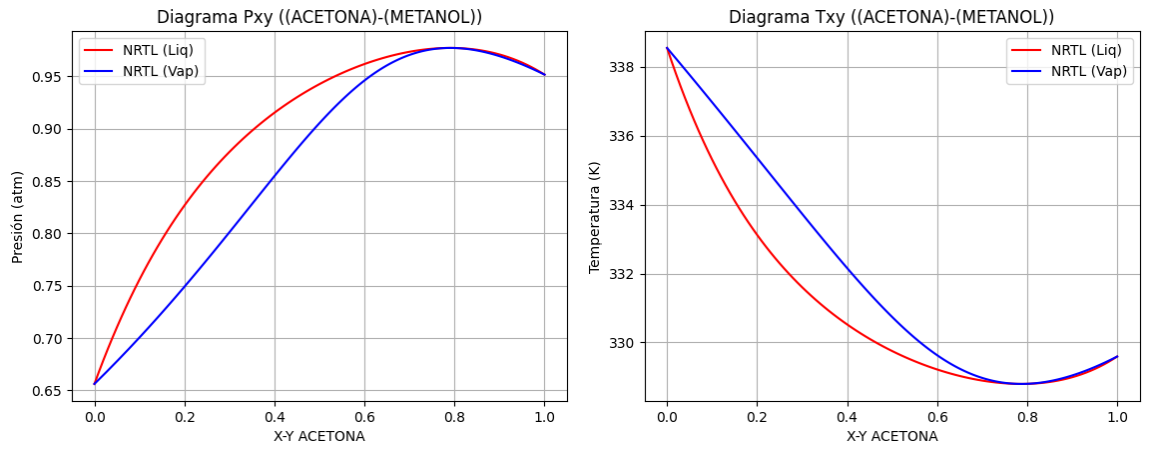


*Figura 29. Diagramas de equilibrio realizados con la ecuación de NRTL para obtener coeficientes de actividad para mezcla acetona-metanol a 328.15 K (P-xy) y 1 atm (T-xy)*

***Tabla 3.*** *Resultados del diagrama de presión composición para ecuación de NRTL para una mezcla binaria de acetona-metanol a 328.15 K*

| X1 | X2 | C.A. (γ1) | C.A. (γ2) | P.VAP. (P1,atm) | P.VAP. (P2,atm) | P.SIST. (PT,atm) | T.SIST. (K) | Y1 | Y2 |
|---|---|---|---|---|---|---|---|---|---|
| 0.0000 | 1.0000 | 1.9314 | 1.0000 | 0.0000 | 0.6560 | 0.6560 | 328.1500 | 0.0000 | 1.0000 |
| 0.1000 | 0.9000 | 1.6940 | 1.0069 | 0.1612 | 0.5944 | 0.7556 | 328.1500 | 0.2133 | 0.7867 |
| 0.2000 | 0.8000 | 1.5095 | 1.0276 | 0.2873 | 0.5392 | 0.8265 | 328.1500 | 0.3476 | 0.6524 |
| 0.3000 | 0.7000 | 1.3659 | 1.0624 | 0.3900 | 0.4878 | 0.8778 | 328.1500 | 0.4443 | 0.5557 |
| 0.4000 | 0.6000 | 1.2544 | 1.1122 | 0.4775 | 0.4377 | 0.9152 | 328.1500 | 0.5217 | 0.4783 |
| 0.5000 | 0.5000 | 1.1685 | 1.1786 | 0.5560 | 0.3866 | 0.9426 | 328.1500 | 0.5899 | 0.4101 |
| 0.6000 | 0.4000 | 1.1037 | 1.2637 | 0.6302 | 0.3316 | 0.9618 | 328.1500 | 0.6552 | 0.3448 |
| 0.7000 | 0.3000 | 1.0565 | 1.3705 | 0.7038 | 0.2697 | 0.9735 | 328.1500 | 0.7229 | 0.2771 |
| 0.8000 | 0.2000 | 1.0245 | 1.5029 | 0.7799 | 0.1972 | 0.9771 | 328.1500 | 0.7982 | 0.2018 |
| 0.9000 | 0.1000 | 1.0060 | 1.6659 | 0.8616 | 0.1093 | 0.9709 | 328.1500 | 0.8874 | 0.1126 |
| 1.0000 | 0.0000 | 1.0000 | 1.8660 | 0.9516 | 0.0000 | 0.9516 | 328.1500 | 1.0000 | 0.0000 |

## Conclusiones

Python es un lenguaje de programación de fácil acceso, flexible y completo, el cual permitió resolver problemas del área de la ingeniería química, los métodos numéricos y la física computacional. Los resultados presentados demostraron que Python es una herramienta numérica capaz de ejecutar cálculos complejos y repetitivos de manera rápida, ya que tiene integrada en sus bibliotecas solucionadores de raíces y de sistemas de ecuaciones.
Además de realizar los cálculos, Python permitió la presentación de resultados al tener integrado funciones de generación de gráficas y tablas. Es por eso que la conjunción de estos factores permitió demostrar que Python es un lenguaje de programación ideal para la enseñanza de la ingeniería y la ciencia.

## Bibliografía/Referencias

[1] Amazon Web Services. (2023.). *¿Qué es Python?*. Recuperado el 26 de julio de 2024, de https://aws.amazon.com/es/what-is/python/#:~:text=Los%20desarrolladores%20utilizan%20Python%20porque,aumenta%20la%20velocidad%20del%20desarrollo

[2] GitHub. (2024). *Acerca de GitHub y Git*. Recuperado el 26 de julio de 2024, de https://docs.github.com/es/get-started/start-your-journey/about-github-and-git

[3] Free Software Foundation. (2022.). *A quick guide to GPLv3*. Recuperado el 26 de julio de 2024, de https://www.gnu.org/licenses/quick-guide-gplv3.html

[4] Seader S.D. , Henley E. , Roper K. (2011) *Separation Process Principles: Chemical and Biochemical Operations* (3erd ed.), John & Wiley Sons

[5] McCabe W. , Smith J. , Harriott P. (2022) *Unit Operations of Chemical Engineering* (7th ed.), McGraw-Hill

[6] Cortés-Castañeda A. (2003) *Diseño de un Fascículo sobre el Método de McCabe-Thiele* Universidad Nacional Autónoma de México

[7] Geankopolis J.C. (2003) *Procesos de transporte y principios de procesos de separación* (4ta ed.) CECSA

[8] Robert H. Perry. (2008). *Perry's chemical engineers' handbook*. (8th ed.): McGraw-Hill.

[9] Felder R. & Rousseau R. (2004) *Principios elementales de los procesos químicos* (3ra ed.) Limusa-Wiley

[10] Cao, E. (2004). *Transferencia de Calor en Ingeniería de Procesos* (1ra ed.).

[11] Banderas, A, V. (2013). *Ingeniería del Calor*. UNAM

[12] Chapra S. C. & Canale R. P., (2015) *Métodos Numéricos para Ingenieros* Editorial McGraw Hill Education.

[13] Garcia A. L., *Numerical Methods for Physics (*2° Ed*).* Editorial Prentice Hall

[14] GitHub. *(2018.). MonteCarloIsing2D.ipynb.* Recuperado el 26 de julio de 2024, de https://github.com/jdalzatec/minicurso-monte-carlo-vegas/blob/master/ising2D/MonteCarloIsing2D.ipynb

[15] DeVries P. L., (1994) *A Firts Course in Computational Physics* Editorial John Wiley & Sons, Inc.

[16] Walas, S. M. (1985). *Phase equilibria in chemical engineering.* Butterworth.

[17] Prausnitz, J. M., Lichtenthaler, R. N., & Edmundo Gomez De, Azevedo. (2000). *Termodinámica molecular de los equilibrios de fase* (3.a ed.). Prentice Hall.

[18] Bazúa E. (2007). .*Equilibrio De Fases Con Modelos Simples En Notas De Curso Termodinámica Química*. UNAM Recuperado el 26 de Julio de 2024 de https://www.academia.edu/36691202/3_EQUILIBRIO_DE_FASES_CON_MODELOS_SIMPLES