Universidad Nacional Autónoma de México

Facultad de Química

pICNIK: Implementación en python de cálculos de isoconversión para cinética anisotérmica

T E S I S

que para obtener el título de:

Químico

P R E S E N T A :

Erick Ramírez Orozco

tutor

Dr. Jorge Balmaseda Era

Ciudad Universitaria, CD. MX., 2021

Agradecimientos

Agradezco el apoyo financiero proporcionado por la DGAPA-UNAM a través del proyecto PAPIIT IG100618.

Agradezco también al soporte técnico del Instituto de Investigaciones en Materiales:
Karla Eriseth Reyes Morales, Omar Alejandro Pompa García y Alberto Lopez Vivas.

A mi asesor, Jorge, por todo el apoyo que me otorgó durante el desarrollo de este proyecto y sus valiosos y pertinentes consejos.

Al Dr. Enelio Torres García y al M. en C. Sergio Hernández López por darme las herramientas y el apoyo para llevar a término este proyecto.

Por último quiero agradecer a mi madre, a mi padre, a mi familia y a mis amigos.

Gracias.

In memoriam al Dr. Luis Felipe del Castillo.

Dedicatoria

A quienes amo y me aman.
Sepan que sin ustedes ningún logro en mi vida sería posible.

Índice general

Índice de figuras

Índice de códigos

Índice de tablas

Introducción

En ciencia e ingeniería es muy común estudiar las propiedades de un material en función de la temperatura y el tiempo. A este tipo de estudios se les llama termocinéticos. Es relativamente sencillo estudiar la termocinética de procesos para sistemas homogéneos como lo son aquellos en fase gaseosa o en disolución a través de parámetros cinéticos como la energía de activación, que determina la cantidad de energía necesaria para pasar de un estado inicial a un estado final, gracias a la teoría del estado de transición y el complejo activado. Para realizar esa clase de estudios en sistemas sólidos o heterogéneos en general, al no haber una teoría que describa la reactividad en función de la temperatura, era necesaria otra aproximación para poder obtener información cinética.

A inicios del siglo XX Akahira y Kujirai [1] propusieron el primer modelo para estudiar la transformación de sólidos en función de la temperatura y el tiempo, proponiendo además, que se podría utilizar a condiciones anisotérmicas[1] y dieron inicio al análisis cinético anisotérmico. Esta clase de experimentos empezó a ganar popularidad en la segunda mitad del siglo con el desarrollo de diferentes métodos de cálculo para obtener los parámetros cinéticos, a saber, la energía de activación, el factor pre-exponencial y un modelo relacionado con el mecanismo de transformación. El más sencillo de dichos métodos es proponer que el proceso que se estudia sigue un comportamiento que se puede describir con algún modelo semiempírico y ajustar los parámetros a los datos experimentales. Otros métodos de cálculo están basados en múltiples experimentos, y consisten en comparar la evolución de una propiedad medible dentro de un sistema térmicamente controlado, a diferentes programas de calentamiento, para múltiples valores de dicha evolución, llamados de isoconversión.

Con el tiempo, los métodos de isoconversión fueron ganando popularidad hasta que en el año 2000, un proyecto de la ICTAC[2] determinó la validez de dichos métodos de cálculo además de sugerirlos como métodos para determinar cinética de pasos múltiples [2], ya que al evaluar los parámetros cinéticos a diferentes valores de conversión se puede descomponer el proceso global en múltiples subprocesos, teniendo cada uno parámetros cinéticos asociados.

Hoy en día se utilizan los métodos de isoconversión para determinar propiedades como son el tiempo de vida de cierto material o sustancia ante un prolongado estrés térmico [1], la viabilidad de aprovechar residuos orgánicos como biocombustible o para generar materias primas a partir de pirólisis [3, 4, 5, 6], también se han utilizado para determinar las condiciones de almacenamiento de material inflamable [7] o medicamentos incluso [8], así como las condiciones óptimas de síntesis de determinadas reacciones que involucran procesos térmicos. Sin embargo, estos métodos son un cuello de botella en el proceso de análisis termocinético ya que son muy laboriosos y consumen mucho tiempo involucrando cálculos como son: integración numérica, minimización de funciones, regresiones lineales y todos a cada valor de conversión por lo que pueden tomar varios días si no se cuenta con un software, generalmente comercial, para realizar dichos cálculos.

[1] Opuesto a isotérmicas, *i.e.* a diferentes temperaturas.

[2] Confederación de Análisis Térmico y Calorimetría por sus siglas en inglés.

Ahora bien, actualmente la computación es una herramienta para todos los campos de la ciencia y existen diversos lenguajes de programación capaces de resolver problemas numéricos de forma eficiente, aunque dependiendo del lenguaje puede llegar a ser una tarea tediosa. Python ofrece un ambiente interactivo de trabajo que, junto a un vocabulario y sintaxis intuitivos lo hacen un lenguaje fácil de aprender, versátil y eficiente para el trabajo científico. Sin embargo, no hay librerías en el repositorio de python (PyPI) con métodos de cálculo de isoconversión.

El objetivo de este trabajo es diseñar y desarrollar una herramienta para calcular la energía de activación de procesos térmicamente estimulados por medio de métodos de isoconversión en la forma de una paquetería de python con licencia libre. De este modo, la herramienta se podrá aprovechar por la comunidad científica. Con la ayuda de esta herramienta será posible realizar el cálculo de energía de activación en cuestión de minutos lo que da lugar a mayor tiempo para analizar los resultados, y a su vez, a mejores y más rápidas tomas de decisiones.

Dada la sintaxis de python, lo más conveniente es desarrollar un módulo de python con orientación a objetos, ya que de ese modo se puede trabajar paso a paso y de forma dinámica en la obtención de cada cantidad de interés durante el análisis y observar el comportamiento de cada variable experimental, con apoyo de otros módulos de python, mientras se trabaja con una cantidad conveniente de puntos o por secciones de temperatura o conversión que sean de particular interés para el análisis.

CAPÍTULO 1

Breve contexto del análisis termocinético

1.1. Desarrollo Histórico

En la década de 1920 Kujirai y Akahira, mediante un primer diseño de termogravimetría, estudiaron la estabilidad térmica de aislantes naturales como algodón y seda [1]. Realizando determinaciones isotérmicas de peŕdida de masa contra tiempo, a diferentes temperaturas, encontraron la relación empírica:

$$\ln(t) = \frac{Q}{T} - f(m) \tag{1.1}$$

donde: t y T son el tiempo y la temperatura respectivamente, Q es una constante característica de cada material y m el porcentaje de masa.

A partir de la ecuación (1.1), Akahira llegó a una ecuación diferencial muy similar a la definición de tasa de reacción para un sistema homogéneo [1]:

$$\frac{\mathrm{d}m}{\mathrm{d}t} = C\exp\left(-\frac{Q}{T}\right)f'(m) \tag{1.2}$$

Después de aplicar la ecuación (1.1) a varios datos, a diferentes temperaturas, propuso que el modelo era aplicable a condiciones experimentales anisotérmicas. Desde entonces se ha usado la ecuación (1.2) para estudiar la termocinética de procesos en sistemas sólidos y heterogéneos, proponiendo que el término que no depende de la masa es, por analogía, la ecuación de Arrhenius llegando a la ecuación general.

$$\frac{\mathrm{d}\alpha}{\mathrm{d}t} = A\exp\left(-\frac{E}{RT}\right)f(\alpha) = k(T)f(\alpha) \tag{1.3}$$

La ecuación (1.3) describe el proceso de transformación de toda la muestra siendo: E la energía de activación; R, la constante universal de los gases ($8.314\,\mathrm{J\,mol^{-1}\,K^{-1}}$); T la temperatura absoluta y α, la fracción descompuesta, también conocida como grado de conversión y está definida como la variación en el tiempo de una propiedad física que es proporcional al proceso [9], en análisis térmico usualmente es la masa o el flujo de calor (pero puede ser cualquier propiedad medible dentro de un sistema térmicamente controlado) y está definida por:

$$\alpha = \frac{\zeta_0 - \zeta(t)}{\zeta_0 - \zeta_f} \tag{1.4}$$

donde: ζ_0, ζ_f y $\zeta(t)$ son: la propiedad física medida inicial, final y al tiempo t respectivamente. $f(\alpha)$ es, entonces, una función que contiene información sobre el mecanismo del proceso y en

Tabla 1.1: Modelos cinéticos de reacción en sólidos con su código asociado, su forma diferencial ($f(\alpha)$ en la ecuación (1.3)) y su forma integral ($g(\alpha)$ en la ecuación (1.11)).

Código	Modelo	$f(\alpha)$	$g(\alpha)$
P1	Ley de potencia	$4a^{3/4}$	$a^{1/4}$
P2	Ley de potencia	$3a^{2/3}$	$a^{1/3}$
P3	Ley de potencia	$2a^{1/2}$	$a^{1/2}$
P4	Ley de potencia	$\frac{2}{3}a^{-1/2}$	$a^{3/2}$
R1	Orden Cero	1	α
R2	Reacción controlada en interfase	$2(1-\alpha)^{1/2}$	$1-(1-\alpha)^{1/2}$
R3	Reacción controlada en interase	$3(1-\alpha)^{2/3}$	$1-(1-\alpha)^{1/3}$
F1	Primer Orden	$(1-\alpha)$	-ln $(1-\alpha)$
F2	Segundo Orden	$(1-\alpha)^2$	$2\left[(1-\alpha)^{-\frac{1}{2}}-1\right]$
A2	Avrami-Eroféev (n=2)	$2(1-\alpha)[-\ln(1-\alpha)]^{1/2}$	-ln $(1-\alpha)^{1/2}$
D1	Difusión en una dimensión	$\frac{1}{2}\alpha$	α^2

general es desconocida pero se suelen usar modelos semi-empíricos como los que se muestran en la tabla 1.1 para describir descomposición de sólidos principalmente, aunque se pueden utilizar para describir cualquier proceso que tenga un buen ajuste estadístico con los datos experimentales [9].

La propuesta de que la ecuación (1.3) es aplicable ante condiciones anisotérmicas no fue del todo bien aceptada por toda la comunidad de análisis térmico de la época, de hecho hubo un gran debate durante décadas sobre cuál era la forma correcta de llevar acabo el análisis termocinétco [10]. En 1979 Sestak publica *"La filosofía del análisis anisotérmico"* [11] en el que, mediante el planteamiento y propuesta de respuestas de 12 preguntas, expone los problemas de aceptación de dicho análisis y menciona tanto puntos a favor como en contra. Algunos de estos puntos son: la falta de formalidad en la formulación de la ecuación (1.3); la forma de interpretar los parámetros A y E, ya que no se puede pensar en un estado de transición como se piensa en los sistemas homogéneos ni hay argumentos mecánico-estadísticos para darle interpretación a dichos conceptos en sistemas sólidos; la congruencia entre información obtenida por medio de análisis anisotérmico y otro tipo de caracterizaciones; la diferencia entre datos isotérmicos contra datos anisotérmicos; la relación entre propiedades termodinámicas del sistema y la cinética del proceso y el efecto que llegarían a tener las computadoras sobre el análisis cinético. Para 1988 se tenía la idea de que las transformaciones mediante aumento de temperatura podían describirse con la ecuación de Arrhenius [12]. También había autores que pensaban que los resultados obtenidos mediante análisis tanto isotérmico como anisotérmico, debían dar resultado coherentes, pero esto a veces sucedía y a veces no. Ya en la década de los 90, el análisis cinético anisotérmico había crecido en popularidad habiendo dos métodos generales: diferenciales e integrales. Finalmente, en el año 2000 la ICTAC publica resultados en los que se valida el uso de métodos cinéticos anisotérmicos [2] y recomiendan particularmente los de isoconversión para obtener información cinética incluso de procesos complejos con múltiples eventos.

1.2. Objetivo de los estudios termocinéticos

Los estudios cinéticos de procesos térmicamente estimulados se realizan con el fin de parametrizar la tasa de la evolución del proceso a través de los parámetros cinéticos A y E y de obtener pistas respecto al mecanismo del mismo con $f(\alpha)$.

El enfoque más simple para lograr esto es proponer que el proceso sigue un comportamiento que se puede describir con uno de los modelos de la tabla 1.1 y ajustar los parámetros A y E a los datos experimentales [9]. Este método se ve limitado por el hecho de que puede haber muchos modelos que estadísticamente ajusten bien para un conjunto de datos experimentales, lo que da lugar a diferentes parámetros cinéticos para un solo proceso. Para superar dicha limitación se desarrollaron métodos que aseguraran un solo valor para los parámetros cinéticos. Uno de ellos fue propuesto por Homer Kissinger durante una estancia en el NIST[1] en los 50's [13]. El método de Kissinger consiste en suponer que el proceso sigue el comportamiento del modelo F1:

$$\frac{\mathrm{d}\alpha}{\mathrm{d}t} = A\,(1-\alpha)\exp\left(-\frac{E}{RT}\right) \tag{1.5}$$

La tasa máxima de la razón conversión $\mathrm{d}(\mathrm{d}\alpha/\mathrm{d}t)/\mathrm{d}t = 0$ ocurre a cierta temperatura T_m, definida por:

$$\frac{E}{RT_m^2}\frac{\mathrm{d}T}{\mathrm{d}t} = A\exp\left(-\frac{E}{RT_m}\right) \tag{1.6}$$

a tasa de calentamiento constante $\mathrm{d}T/\mathrm{d}t \equiv \beta = cte$

$$\begin{aligned}
\frac{E\beta}{RT_m^2} &= A\exp\left(-\frac{E}{RT_m}\right)\\
\frac{\beta}{T_m^2} &= \frac{R}{E}A\exp\left(-\frac{E}{RT_m}\right)\\
\ln\left(\frac{\beta}{T_m^2}\right) &= \ln\left(\frac{RA}{E}\right) - \frac{E}{RT_m}
\end{aligned} \tag{1.7}$$

Por lo tanto, para un sistema que obedece una cinética de primer orden, la energía de activación puede obtenerse de la pendiente de una regresión lineal de los datos $\ln\left(\beta_i/T_{i,m}^2\right)$ vs ${T_{i,m}}^{-1}$, para lo que se necesitan resultados de múltiples programas de calentamiento lineal. Esto limita el método a procesos que puedan ser descritos con el modelo F1. Se puede observar que la función $f(\alpha)$ es el origen de las limitaciones para encontrar de manera confiable los parámetros cinéticos e información sobre el mecanismo del proceso con los métodos mencionados hasta ahora. En la siguiente sección se describen los métodos de isoconversión que no requieren información previa de la función de conversión y son capaces de dar resultados confiables de la energía de activación del proceso.

1.3. Métodos de Isoconversión

Estos métodos parten del principio de isoconversión, que parte de la hipótesis de que la tasa de conversión $(\mathrm{d}\alpha/\mathrm{d}T)$ solo depende de la temperatura, a conversión constante. Esta suposición es válida en los casos para los que la forma de $f(\alpha)$ es independiente de la razón de calentamiento (β), *i.e.* las regiones para las que se observe un desplazamiento paralelo de la conversión hacia temperaturas mayores como se muestra en la figura 1.1.

[1] Instituto Nacional de Estándares y Tecnología, por sus siglas en inglés (instituto de E.U.A.).

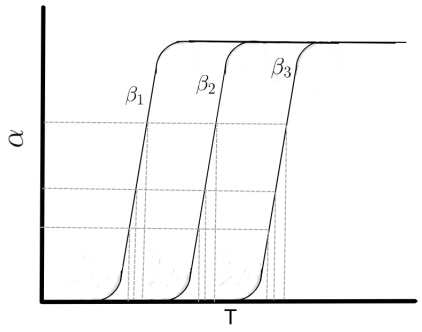


Figura 1.1: Ejemplo de una curva que muestra un proceso en que $f(\alpha)$ es independiente de β por el desplazamiento de la conversión a temperaturas mayores ($\beta_3 > \beta_2 > \beta_1$), pero con el mismo perfil

Si la suposición es válida, se puede linealizar la ecuación (1.3) mediante la aplicación del logaritmo natural:

$$\ln\left(\frac{\mathrm{d}\alpha}{\mathrm{d}t}\right) = \ln\left[Af\left(\alpha\right)\right] - \frac{E}{RT} \tag{1.8}$$

y derivarla respecto al recíproco de la temperatua:

$$\left[\frac{\partial \ln\left(\frac{\mathrm{d}\alpha}{\mathrm{d}t}\right)}{\partial T^{-1}}\right]_{\alpha} = -\frac{E_\alpha}{R} \tag{1.9}$$

con lo que se obtiene una expresión para la energía de activación que no depende de la función de conversión $f(\alpha)$. El subíndice α indica que ese valor de conversión se está manteniendo constante. Estos métodos requieren información experimental a múltiples programas de temperatura. Su principal ventaja es que no se hace ninguna suposición respecto a $f\left(\alpha\right)$. La siguiente ventaja es que se pueden identificar múltiples procesos a través de la variación de la energía de activación a isoconversión (E_α) (que es una energía de activación efectiva o aparente) con la conversión del proceso (α). Existen dos tipos de métodos generales de isoconversión: los diferenciales y los integrales.

1.3.1. Métodos Diferenciales

Los métodos diferenciales parten de la ecuación (1.3). El más popular es el de Friedman [14] que usa la ecuación (1.8) pero considerando la hipótesis de isoconversión:

$$\ln\left(\frac{\mathrm{d}\alpha}{\mathrm{d}t}\right)_{\alpha,i} = \ln\left[A_\alpha f\left(\alpha\right)\right] - \frac{E_\alpha}{RT_{\alpha,i}} \tag{1.10}$$

El subíndice i indica que son los datos del i-ésimo programa de calentamiento. Al graficar $\ln\left(\frac{\mathrm{d}\alpha}{\mathrm{d}t}\right)_{\alpha,i}$ vs $T_{\alpha,i}^{-1}$ para diferentes valores de conversión, se puede obtener el valor de E_α multiplicando el valor de la pendiente, obtenida mediante regresión lineal, por $-R$.

1.3.2. Métodos Integrales

Los métodos integrales parten de la forma integral de la ecuación (1.3):

$$g(\alpha) \equiv \int_0^{\alpha} \frac{\mathrm{d}\alpha}{f(\alpha)} = A \int_{t_0}^{t} \exp\left(-\frac{E}{RT}\right) \mathrm{d}t = AJ(E, T(t)) \tag{1.11}$$

a tasa de calentamiento lineal $\frac{\mathrm{d}T}{\mathrm{d}t} = \beta$ con β constante:

$$g(\alpha) = \frac{A}{\beta} \int_{T_0}^{T} \exp\left(-\frac{E}{RT}\right) \mathrm{d}T = \frac{A}{\beta} I(E, T) \tag{1.12}$$

La integral de temperatura $I(E,\ T)$ no puede resolverse analíticamente. Puesto que la cantidad de integrales que se tienen que calcular es grande, en la década del 60 no era práctico usar métodos de integración numérica, debido al costo computacional que ello implica, por lo que se desarrollaron múltiples métodos de aproximación para ella 1.12 [15, 16, 17].

Dos métodos integrales de isoconversión muy usados son el de Ozawa [18] y Flynn-Wall [19] (OFW) y el de Kissinger-Akahira-Sunose (KAS), que involucran aproximaciones polinómicas de la integral de temperatura, que resultan en ecuaciones lineales. El método OFW utiliza la aproximación de Doyle [15]:

$$I(E_\alpha, T_\alpha) \approx \left(\frac{E_\alpha}{R}\right) \exp\left(-5.331 - 1.052 \frac{E_\alpha}{RT_{\alpha,i}}\right) \tag{1.13}$$

con la que se puede llegar a una ecuación lineal en $T_{\alpha,i}^{-1}$:

$$\ln(\beta_i) \approx \left[\ln\left(\frac{A_\alpha E_\alpha}{g(\alpha)R}\right) - 5.331\right] - 1.052 \frac{E_\alpha}{RT_{\alpha,i}} \tag{1.14}$$

El método KAS utiliza la aproximación de Coats-Redfern [16] y llega a la expresión:

$$\ln\left(\frac{\beta_i}{T_{\alpha,i}^2}\right) \approx \ln\left[\frac{A_\alpha R}{E_\alpha g(\alpha)}\right] - \frac{E_\alpha}{RT_{\alpha,i}} \tag{1.15}$$

Un tercer método integral parte del hecho de que la función $g(\alpha)$ vale lo mismo para cierta conversión dada, por lo que para cada conversión alcanzada a diferentes programas de calentamiento se debe cumplir que:

$$g(\alpha = cte) = \frac{A_\alpha}{\beta_i} I(E_\alpha, T_{\alpha,i}) = \frac{A_\alpha}{\beta_j} I(E_\alpha, T_{\alpha,j}) = ... = \frac{A_\alpha}{\beta_n} I(E_\alpha, T_{\alpha,n}) \tag{1.16}$$

Entonces se tiene que para todo par de programas de calentamiento i y j:

$$\beta_j I(E_\alpha, T_{\alpha,i}) = \beta_i I(E_\alpha, T_{\alpha,j}) \tag{1.17}$$

$$\frac{\beta_j I(E_\alpha, T_{\alpha,i})}{\beta_i I(E_\alpha, T_{\alpha,j})} = 1 \tag{1.18}$$

al sumar sobre todos los pares experimentales se obtiene la doble suma:

$$\sum_{i}^{n} \sum_{j \neq i}^{n-1} \frac{\beta_j I(E_\alpha, T_{\alpha,i})}{\beta_i I(E_\alpha, T_{\alpha,j})} = n(n-1) \tag{1.19}$$

Bajo condiciones experimentales, al existir el ruido provocado por las condiciones ambientales y errores asociados a las mediciones, la ecuación 1.19 no es una igualdad estricta, pero permite plantear la siguiente condición [20, 21]:

$$\phi = n(n-1) - \sum_{i}^{n} \sum_{j \neq i}^{n-1} \frac{\beta_j I(E_\alpha, T_{\alpha,i})}{\beta_i I(E_\alpha, T_{\alpha,j})} \tag{1.20}$$

La energía de activación es aquella que minimiza la ecuación (1.20). La integral de temperatura se resuelve con alguna aproximación o con métodos numéricos. En este trabajo se utilizan la regla del trapecio, la regla de Simpson y la aproximación de Senum y Yang [17]:

$$I\left(E_\alpha, T_\alpha\right) \approx \frac{E_\alpha}{R} p\left(x\right) \tag{1.21}$$

donde

$$p\left(x\right) = \frac{e^{-x}}{x} \frac{x^3 + 18x^2 + 88x + 96}{x^4 + 20x^3 + 120x^2 + 240x + 120} \tag{1.22}$$

con $x = E_\alpha / RT_\alpha$.

Existe una modificación a este método para poder percibir cambios más sutiles en la variación de la energía de activación que se pierden porque al trabajar con la integral (1.12) se supone que la energía de activación es constante durante todo el intervalo de integración, entonces si llega a haber alguna variación, sería promediada. El método conocido como Vyazovkin Avanzado [22, 23] consiste en sustituir la integral $I\left(E_\alpha, T_\alpha\right)$ por:

$$I\left(E_{\Delta\alpha}, T_{\Delta\alpha}\right) = \int_{T_{\alpha-\Delta\alpha}}^{T_\alpha} \exp\left(-\frac{E_{\Delta\alpha}}{RT}\right) \mathrm{d}T \tag{1.23}$$

Con el software Wolfram|Alpha[2], se encontró una solución analítica para la integral (1.23), que solo está definida para valores reales positivos de $E_{\Delta\alpha}$, lo que es consistente con el concepto de energía de activación[3]:

$$\begin{aligned} I\left(E_{\Delta\alpha}, T_{\Delta\alpha}\right) = \frac{E_{\Delta\alpha}}{R} F\left(T_{\Delta\alpha}\right) + T_\alpha \exp\left(-\frac{E_{\Delta\alpha}}{RT_\alpha}\right) \\ - T_{\alpha-\Delta\alpha} \exp\left(-\frac{E_{\Delta\alpha}}{RT_{\alpha-\Delta\alpha}}\right) \end{aligned} \tag{1.24}$$

con

$$F\left(T_{\Delta\alpha}\right) = E_i\left(-\frac{E_{\Delta\alpha}}{RT_\alpha}\right) - E_i\left(-\frac{E_{\Delta\alpha}}{RT_{\alpha-\Delta\alpha}}\right) \tag{1.25}$$

donde E_i es la integral exponencial definida por $-E_i\left(-x\right) \equiv \int_x^\infty \left(\exp\{(-t)\}/t\right) dt$.

Una forma aún más general es realizar la integración sobre el tiempo y de ese modo trabajar con un programa de temperatura que no necesariamente sea lineal:

$$J\left[E_{\Delta\alpha}, T\left(t_{\Delta\alpha}\right)\right] = \int_{t_{\alpha-\Delta\alpha}}^{t_\alpha} \exp\left[-\frac{E_{\Delta\alpha}}{RT\left(t\right)}\right] \mathrm{d}t \tag{1.26}$$

Al integrar en intervalos pequeños y equidistantes de α se vuelve más sensible el método, por lo que es recomendable mantener $\Delta\alpha$ lo más pequeño posible.

[2] https://www.wolframalpha.com/

[3] Aunque se han obtenido valores negativos para la energía de activación, esto se adjudica al artefacto matemático y no a un fenómeno físico [23].

1.3.3. Predicciones

Para realizar predicciones sin hacer ninguna suposición sobre el modelo cinético $f(\alpha)$, se puede partir de la ecuación (1.10) y reemplazar los diferenciales por diferencias finitas y despejar el valor de conversión que se desea conocer [24]:

$$\alpha_i = \alpha_{i-1} + \Delta t \left[A_{\alpha_i} f\left(\alpha_i\right)\right] \exp\left[-\frac{E_{\alpha_i}}{RT\left(t_{\alpha_i}\right)}\right] \tag{1.27}$$

donde $\Delta t = t_{\alpha_i} - t_{\alpha_{i-1}}$ y t_{α_i} es el tiempo que el proceso tarda en llegar a la conversión α_i. Teniendo los valores de energía de activación en función de la conversión para un proceso dado, se puede predecir la conversión a un programa de calentamiento propuesto, el término $[A_\alpha f\left(\alpha\right)]$ se puede calcular a partir de la ordenada al origen de la ecuación (1.10).

CAPÍTULO 2

Introducción exprés a Python

Para que quede mejor ilustrada la idea de lo que es python me parece útil explicar, aunque sea de modo muy superficial, cómo funciona una computadora. Esencialmente una computadora es una herramienta de cálculo, esta herramienta puede describirse a través de sus partes físicas (*hardware*) y de las funciones que puede realizar (*software*). El *hardware* consiste en un circuito electrónico compuesto de métodos de entrada de información, procesamiento de información y salida de información. Los de entrada de información están conectados al teclado, al micrófono y a la cámara que se pueden adaptar a una computadora. Los de salida están conectados a la pantalla, a las bocinas etc. Mientras que la parte que procesa la información está conectada al CPU (unidad central de procesamiento por sus siglas en inglés) y a un par de memorias y todos esos elementos están conectados entre sí en un "tablero madre" (*mother board*). El CPU como su nombre lo indica, es el principal encargado del procesamiento de la información que llega al sistema, pero también se encarga de dar las instrucciones para que los métodos de salida de información, la entreguen de forma adecuada y todo lo realiza mediante un proceso bastante complejo basado en un principio muy sencillo: bloquear o permitir el paso de corriente. El CPU trabaja a través de estos dos estados: bloquear|permitir, encendido|apagado, 1|0. El que el CPU trabajé de ese modo permite programarle operaciones mediante el uso del álgebra booleana (desarrollada en el siglo XIX por George Boole[25]) y el sistema binario (1|0). Esto es lo que se conoce como lenguaje de máquina. El lenguaje de máquina es bastante complicado por su sintaxis, por lo que es fácil equivocarse al intentar programar una función; depende de la arquitectura de la computadora y muy pocos programadores llegan a hacerlo ya que se han desarrollado lenguajes a través de la abstracciones de las operaciones físicas del CPU, llamados lenguajes de medio nivel y de alto nivel. Los lenguajes de medio nivel consisten en un código de bytes que son un tipo de formulario que relaciona mnemotecnias (como "ADD" para sumar) con números en binario. Los lenguajes de alto nivel, lo son en el sentido de la abstracción de las operaciones del CPU. A través de comandos en lenguaje inglés es posible programar operaciones en la computadora, por lo que se necesitan traductores para que la computadora sepa lo que se le está pidiendo.

Para la traducción del lenguaje de alto nivel al de máquina existen dos tipos de traductores: compiladores e intérpretes. Los compiladores reciben un archivo con el algoritmo programado en cierto lenguaje de alto nivel y después lo transforman a un archivo en lenguaje de máquina, eso implica que se deba escribir todo el programa completamente y después intentar compilarlo, si existe algún error de sintaxis, el programa no compila y el compilador regresa un mensaje indicando qué clase error se cometió. El intérprete, por otro lado, es un traductor interactivo, se pueden ir escribiendo funciones, renglones, o incluso solo variables individuales como entrada en el intérpretes, y éste inmediatamente los evalúa y regresa un resultado.

Python es un lenguaje de alto nivel cuyo traductor es un intérprete, funciona en todos los

sistemas operativos y es gratuito y de licencia de software libre [26].

El interprete llamado *shell* se instala en la computadora junto con python, y es de tipo terminal. Una ilustración de este intérprete se muestra en el código (2.1). Los signos ">>>" que en conjunto se conoce como *prompt* indican que python está listo para recibir indicaciones. Se le escribe un comando (2+2), el intérprete lo traduce, la máquina lo ejecuta e inmediatamente después se obtiene uno de dos posibles resultados: un mensaje de error, o el resultado del comando (4) y el *prompt* de nuevo, indicando que python está listo para seguir trabajando.

```
Python 3.8.6 (default, Oct  6 2020, 04:20:00)
[GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> print 2+2
  File "<stdin>", line 1
    print 2+2
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(2+2)?
>>> print(2+2)
4
>>>
```

Código 2.1: Ejemplo del *shell* de python.

El *shell* de python es útil para experimentar con expresiones cortas, estudiar la documentación de un módulo y nuevas implementaciones por cambio de versión, pero para programas más complejos que requieren de decenas o hasta cientos de líneas de código es mejor crear documentos llamados *script* en un editor de texto como son gedit o vim, entre otros. Estos *scripts* pueden ser ejecutados desde la terminal o desde un ambiente de desarrollo integrado. Los ambientes de desarrollo integrado (IDE por sus siglas en inglés) son programas que permiten trabajar con un editor de texto y el intérprete de forma dinámica y pueden ser de escritorio como spyder o de navegador como jupyter.

Python, como la gran mayoría del software, tiene actualizaciones y nuevas versiones van saliendo conforme se agregan funciones o se arreglan errores, los famosos *bugs*. El programa para este proyecto se desarrolló con la versión 3.8.6 de python y la información que describiré a continuación es referente específicamente a esta versión. La versión más reciente de python y su documentación se pueden consultar en su página web [26].

2.1. Vocabulario

Python tiene un vocabulario amplio y una sintaxis que lo hacen fácil de leer y de escribir (sabiendo un poco de inglés). Como todo lenguaje, es necesario conocer las reglas y los elementos gramaticales que se usan en python. En esta sección y las siguientes presentaré algunos conceptos clave en la escritura de código en python, así como algunos de los módulos más útiles para trabajo científico.

2.1.1. Tipos de datos

string: Datos de "lectura". Son todas las letras, mayúsculas y minúsculas, desde la 'a' a la 'z', el guión bajo '_' y los dígitos del '0" al '9'. Siempre deben ir entre comillas, pueden ser sencillas o dobles, y se pueden concatenar con el signo '+'.

```
>>> 'a'
'a'
>>> 'a'+'j'+'a'
'aja'
>>> ``aja''
'aja'
```

Código 2.2: Datos tipo *string* en python.

integer: Números enteros compuestos por los dígitos del 0 al 9. No llevan comillas, si no, el intérprete los reconoce como *string*. El intérprete puede reconocer como entrada números del sistema decimal [dígito(s)], octal ['0'+'o'+dígito(del 0 al 7)], hexadecimal ['0'+'x'+dígito (0-9)+('a'...'f')] o binario ['0'+'b'+ dígito (1/0)] y regresa el número en decimal.

```
>>> (9, 0o25, 0b1001, 0x12a)
(9, 21, 9, 298)
```

Código 2.3: Datos tipo *integer* en python.

floating: Números de "coma flotante" o punto decimal.

```
>>> 3.14, .001, 10., 3.14e-10, 0e0
(3.14, 0.001, 10.0, 3.14e-10, 0.0)
```

Código 2.4: Datos tipo *float* en python.

complex: Se construyen con la suma de un número real (entero o flotante) más un número imaginario que consiste en un dígito más la letra 'j' (minúscula o mayúscula).

```
>>> 3j, 5.21+ 8j
(3j, (5.21+8j))
```

Código 2.5: Datos del tipo *complex* en python.

boolean: Valores lógicos. Solo pueden ser **False** (Falso) o **True** (Verdadero)

2.1.2. Palabras clave

Python cuenta con palabras clave que interpreta de forma específica, estas palabras clave no pueden asignarse para nombrar variables ni funciones y algunas de ellas se muestran en la tabla (2.1). El símbolo # se utiliza para comentarios, *i.e.*, lo que esté escrito frente a ese símbolo será ignorado por el intérprete.

Cada palabra clave tiene su uso y significado específico, por ejemplo:

if/else/elif: Ejecutan condicionalmente un bloque de código.

for: Realiza operaciones iterativas sobre objetos iterables (como listas o tuplas)

while: Ejecuta un bloque de código mientras su condición sea cierta.

try: Permite atrapar excepciones o errores que surjan a través de condiciones con **except**.

class: Ejecuta un bloque de código y le asigna su nombre local a una clase (para programación orientada a objetos)

def: Define una función o método.

pass: Salta a la acción que sigue.

raise: Devuelve un mensaje, ya sea de error o advertencia, dada una condición.

Tabla 2.1: Palabras clave en python. Deben escribirse tal cual se muestran y no pueden ser nombre de variables o funciones.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

2.1.3. Operadores y Delimitadores

También cuenta con operadores binarios que incluyen a los lógicos (o booleanos) y a los aritméticos como se muestra en la tabla (2.2).

Tabla 2.2: Operadores binarios. Representan operaciones lógicas o aritméticas entre dos objetos.

+	−	*	**	/	//	%
<<	>>	&	|	ˆ	∼	:=
¡	¿	¡=	¿=	==	!=	@

Tabla 2.3: Delimitadores. Envuelven colecciones de datos o indican una operación dentro de un ciclo.

(	)	[	]	{	}	
,	:	.	;	@	=	− >
+=	-=	*=	/=	//=	%=	@ =
&=	| =	ˆ=	>>=	<<=	**=	

2.2. Programación Orientada a Objetos

Python es un lenguaje de programación que puede ser utilizado para el paradigma de programación orientada a objetos (POO), de hecho, toda la información en python es representada por objetos o relaciones entre objetos. Este tipo de programación permite descomponer un algoritmo en un conjunto de partes fundamentales que realizan tareas específicas. Su principal ventaja es que permite tener bien organizados códigos con cientos o miles de líneas.

La programación orientada objetos consiste en abstraer funciones, objetos físicos o conceptos, dentro de algoritmos para despues poder invocarlos mediante palabras clave, definidas dentro del mismo algoritmo. En python se llaman clases a esas abstracciones que van a representar a los objetos y cuentan con propiedades de los objetos. Estas propiedades se dividen en dos tipos: métodos y atributos. Los atributos son aquellas propiedades que sirven para describir al objeto y que pueden ser operadas por los métodos, o ser su resultado, mientras que los métodos son funciones del objeto, algunos son operaciones que pueden realizarse sobre los atributos del objeto. Hay características que distinguen la POO, como lo es el encapsular. El concepto de encapsular es bastante directo, se trata de poner un conjunto de variables y funciones (métodos y atributos) detrás de un programa que las pueda invocar mediante comandos clave, así se obtiene una forma elegante de ir escribiendo código. Para ilustrar el concepto de la POO, en el código (2.6) se muestra la implementación de la *caguama* como clase de python.

```
class caguama(object):
"""
Clase en python para representar una caguama
"""
    def __init__(self, marca):                          #Constructor
        self.marca  = marca                             #Atributo
        self.estado = 'tapada'                          #Atributo
        self.tragos = 50                                #Atributo
        if marca == 'victoria':                         #Condicion
            self.color  = 'obscura'                     #para
        elif marca == 'corona':                         #determinar
            self.color = 'clara'                        #atributo

    def destapar(self):                                 #Metodo
        print('chssk')
        self.estado = 'destapada'

    def trago(self):                                    #Metodo
        if self.estado == 'tapada':
            print('Tienes que detaparme primero wey')
        else:
            print('gluc gluc')
            self.tragos = self.tragos -1

    def cuanta_chela(self):                             #Metodo
        print('quedan '+str(self.tragos)+' tragos')
```

Código 2.6: Abstracción de una caguama como objeto de python.

Para definir clases se utiliza la palabra clave **class** y el *object* entre paréntesis indica que la clase creada hereda los métodos y atributos de la clase *object* que ya está predefinida en python. Python usa indentación para separar bloques de código, por lo que después de la declaración **class caguama(object):**, el resto del código tiene una indentación, indicando que las funciones definidas en ese bloque corresponden a la clase definida. Después, tenemos los métodos, caracte-

rizados por la palabra clave **def** que indica que una función va a ser definida. El primer método de una clase siempre es el *constructor* con el comando **__init__(self,...)** el argumento *self* es una auto-referencia al objeto. Con el constructor se definen la mayoría de, si no es que todos, los atributos de la clase. En el ejemplo los atributos son: **marca**, **estado**, **tragos** y **color**; que precedidos por *self* indican que pertenecen al objeto. Los métodos definidos son **destapar()**, **trago()** y **cuanta_chela()**.

```
>>> import chelas
>>> corona = chelas.caguama('corona')
>>> corona.color
clara
>>> corona.estado
tapada
>>> corona.destapar()
chssk
>>> corona.estado
destapada
>>> corona.trago()
gluc gluc
```

Código 2.7: Ilustración de una interacción con python mediante el modulo *chelas* y el objeto *caguama*.

Python trabaja con varios objetos ya predefinidos como son *string*, *int*, *float*, *tuple*, *list* y *dict*. Las listas (*list*), tuplas (*tuple*) y diccionarios (*dict*) son objetos contenedores de otros objetos. Las listas y tuplas contienen secuencias de objetos de cualquier tipo (que también pueden ser listas, tuplas o diccionarios), se puede acceder al valor de cada elemento a través de indexación como se muestra en el código (2.8), los índices van de 0 a $n-1$, donde n es el número total de elementos en la secuencia. Las listas se crean escribiendo una secuencia separada por comas, delimitada por paréntesis cuadrados, "[ ]". Las tuplas se crean igual que las listas, pero son delimitadas por paréntesis redondos, "( )". La diferencia entre una lista y una tupla es que la primera es modificable, y la segunda no lo es. Con modificable me refiero a que se le pueden agregar o eliminar elementos a la secuencia sin tener que eliminar la secuencia actual y crear una nueva.

Los diccionarios son contenedores modificables que relacionan parejas de objetos mediante una llave (*key*) y un valor (*value*). Se puede acceder a los valores de un diccionario a través de su llave y se puede acceder a las llaves mediante el método **keys()**.

```
>>> k = [0,1,0,5,2]
>>> k[0]
0
>>> l = ['a',0x15F, k, 4.5]
>>> l[1:]
[351, [0,1,0,1,5,2], 4.5]
>>> t = (0,1,2,3)
>>> d = {'a':1, 'b':2, 'c':3}
>>> d.keys()
dict_keys(['a', 'b', 'c'])
>>> d['a']
1
>>> type(k), type(l), type(t), type(d)
(<class 'list'>, <class 'list'>, <class 'tuple'>, <class 'dict'>)
>>>
```

Código 2.8: Listas, tuplas y diccionarios en python.

2.3. Numpy, Scipy, Pandas y Matplotlib

Python tiene cientos de módulos disponibles, algunos vienen incluidos cuando se instala python y el resto se pueden descargar. Cuatro de las librerías más famosas para análisis de datos en python se utilizaron en la implementación de los métodos de isoconversión, a saber: Numpy, Scipy, Pandas y Matplotlib. A continuación describiré brevemente algunas características de cada librería, si el lector está interesado a profundizar en cualquiera, le recomiendo dirigirse a la documentación correspondiente [27, 28, 29, 30] o la referencia [31] para una buena introducción.

2.3.1. Numpy

Numpy (acrónimo de numeric python) es una librería escrita en C que trabaja con el objeto *ndarray* como base, que es un contenedor multidimensional de una colección de datos del mismo tipo y del mismo tamaño (en memoria), el número de dimensiones y elementos en el contenedor está indicado por su "forma" (*shape*). Como convención se suele importar el módulo bajo el alias de "np". El objeto *ndarray* es muy similar al objeto *list* de python, y también se puede acceder a sus elementos mediante indexación, pero la estructura de numpy le otorga un gran poder a *ndarray* a través de dos elementos: vectorización y transmisión. La vectorización se refiere a la capacidad de realizar operaciones sobre todo un arreglo de una vez, *i.e.*, la operación se realiza sobre todos los elemento de la colección de una vez en lugar de hacer la operación un elemento a la vez, como se haría con un ciclo *for*. Esto hace la escritura del código más simple y la operación más rápida. Mientras que la transmisión se refiere a un conjunto de reglas para realizar operaciones entre dos colecciones de distintos tamaños. El objeto *ndarray* puede comportarse como un vector o como una matriz, y de hecho vienen implementadas en numpy operaciones sobre matrices y entre matrices, así como el producto punto entre dos vectores.

```
>>> import numpy as np
>>> n = np.array((2,4,8))
>>> n
array([2, 4, 8])
>>> n.shape
(3,)
>>> M = np.array(([1,2,3],[4,5,6],[7,8,9]))
>>> M
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> M.shape
(3,3)
>>> M.T
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
>>> M+n
>>> n+M
array([[ 3,  6, 11],
       [ 6,  9, 14],
       [ 9, 12, 17]])
>>> M.dot(n)
array([ 34,  76, 118])
```

Código 2.9: El objeto *ndarray* de numpy permite realizar operaciones vectoriales o matriciales de forma eficiente.

2.3.2. Scipy

La paquetería Scipy (Scientific python) está escrita sobre numpy, por lo que trabaja con sus objetos *ndarray*. Scipy es un conjunto de algoritmos numéricos que incluyen resolución de ecuaciones diferenciales, minimización de funciones, integración y diferenciación numérica, transformadas de Fourier y funciones especiales como la función gamma.

Al importar scipy, solo se importan funciones de numpy, para importar las funciones de scipy es necesario importar explícitamente el subpaquete, o la función del subpaquete como se muestra en el código (2.10):

```
from scipy import interpolate
from scipy.optimize import minimize_scalar
```

Código 2.10: Los subpaquetes y funciones de scipy se deben importar explícitamente.

Algunos de los subpaquetes de scipy son:

fft: Transformadas de Fourier discretas.

integrate: Rutinas de integración.

interpolate: Herramientas de interpolación.

io: Entrada y entrega de archivos.

linalg: Rutinas de álgebra lineal.

optimize: Herramientas de optimización.

signal: Herramientas para procesamiento de señales.

special: Funciones especiales.

stats: Funciones estadísticas

2.3.3. Pandas

Pandas es una paquetería diseñada para trabajar con tablas de datos como en una hoja de cálculo pero mucho más flexible. Las tablas de datos en pandas se llaman *DataFrames* y son uno de varios objetos que contiene pandas. Es posible abrir hojas de cálculo o importar bases de datos, así como el proceso inverso, guardar resultados en archivos de varios formatos incluyendo csv, excel, sql y json. El objeto *DataFrame* tiene métodos para organizar, limpiar y analizar los datos que contiene. Los datos pueden ser numéricos o textuales.

Para importar la paquetería pandas se usa como convenio la abreviación pd. Pandas trabaja con dos estructuras de datos que son los objetos *Series* y *DataFrame*. Los primeros son arreglos unidimensionales de información, y cada elemento tiene asociado un índice (como en los objetos *list* y *ndarray*) pero estos índices corresponden a un objeto en pandas llamado *Index*, lo que permite asignar como índices algún arreglo de datos como más convenga en el análisis. Los objetos *Series* se comportan igual que los *ndarray*, con la diferencia de que un objeto *Series* puede contener elemetos textuales (tipo *string*). El objeto *DataFrame* es un contenedor bidimensional de datos (una tabla), que se puede caracterizar por sus columnas y sus índices, que son los atributos de este objeto, esto permite poder acceder a la información de la tabla ya sea por elemento, por columna o por renglón. En el código (2.11) se ilustra el uso de pandas con datos

de coeficientes de compresibilidad y de dilatación para compuestos puros. Primero se generan tres listas conteniendo la información que se va a usar, una con el nombre de los compuestos, otra con los valores del coeficiente de dilatación en $1 \times 10^4\,\mathrm{K}^{-1}$ y una tercera con los valores del coeficiente de compresibilidad en $1 \times 10^6\,\mathrm{bar}^{-1}$, el primer *DataFrame* se genera a partir de un diccionario en el que las llaves son el nombre de la columna y los valores son las listas creadas. El segundo *DataFrame* se genera indicando que los índices sean los nombres de los compuestos y las propiedades de cada compuesto como columnas. Finalmente se muestra como acceder a los elementos de un *DataFrame* a través del índice explícito con el método **loc()**, que corresponde al índice impuesto, o del índice implicito **iloc()**, que corresponde al índice implementado de python.

```
>>> import pandas as pd
>>> compuesto = ['agua', 'benceno', 'mercurio', 'etanol', 'cobre']
>>> dil = [2.1, 12.4, 1.8,11.2, 0.501]
>>> comp = [49.0,90.9,38.2,75.8,0.725]
>>> DF = pd.DataFrame({'compuesto':compuesto,'dilatacion':dil, 'compresibilidad':
    comp})
>>> DF
  compuesto  dilatacion  compresibilidad
0      agua       2.100           49.000
1   benceno      12.400           90.900
2  mercurio       1.800           38.200
3    etanol      11.200           75.800
4     cobre       0.501            0.725
>>> DF = pd.DataFrame({'dilatacion':dil, 'compresibilidad':comp},index=compuesto)
>>> DF
           dilatacion  compresibilidad
agua            2.100           49.000
benceno        12.400           90.900
mercurio        1.800           38.200
etanol         11.200           75.800
cobre           0.501            0.725
>>> DF.loc['agua']
dilatacion          2.1
compresibilidad    49.0
Name: agua, dtype: float64
>>> DF.iloc[0]
dilatacion          2.1
compresibilidad    49.0
Name: agua, dtype: float64
>>>
```

Código 2.11: El objeto *DataFrame* de pandas permite manipular, purgar y analizar series de datos.

2.3.4. Matplotlib

Matplotlib es un visualizador de datos construido sobre los arreglos de numpy. Tiene una sintaxis basada en matlab (matplotlib.pyplot) y otra basada en POO (matplotlib). Al igual que pandas y numpy, tiene un alias para ser invocado, bueno, dos en este caso, que son los que se muestran en el código (2.12).

Los objetos base de matplotlib son figure.Figure y axes.Axes. Se puede considerar al objeto figure.Figure como un lienzo y los objetos axes.Axes como partes de ese lienzo en el que se plasman visualizaciones. Figure.figure puede tener tantos axes.Axes como el usuario deseé, y el objeto axes.Axes puede tener dos o tres ejes, dependiendo de si el gráfico es 2D o 3D. La forma más sencilla de construir una figura es con pyplot que es la interfase capaz de entender la sintaxis

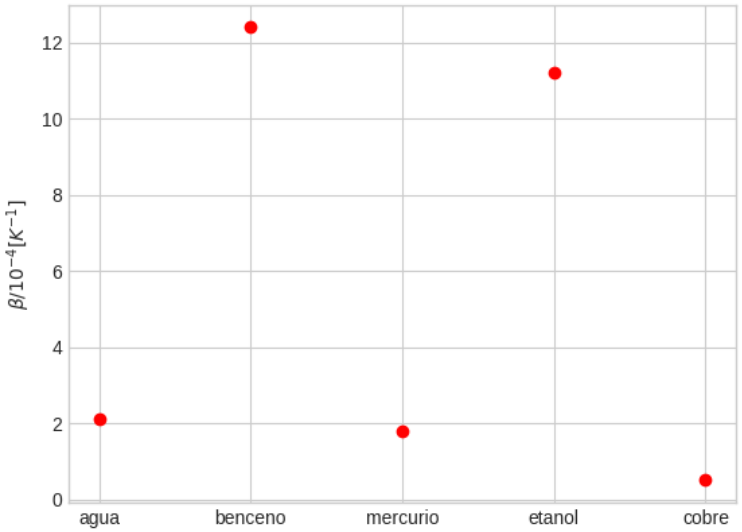


Figura 2.1: Coeficientes de dilatación de sustancias puras.

procedimental de matlab como se muestra en el código (2.12), en el que se grafican los coeficientes de dilatación de sustancias puras del ejemplo de la sección 2.3.3. Para construir figuras complejas con muchas especificaciones, es más conveniente usar la interfase orientada a objetos. Todos los gráficos presentados en este trabajo se construyeron con matplotlib.pyplot

```
>>> import matplot as mpl
>>> import matplotlib.pyplot as plt
>>> plt.style.use('seaborn-whitegrid')
>>> plt.plot(compuesto, dil,'or', label='coef. de dilatacion')
[<matplotlib.lines.Line2D object at 0x7fce83a3b250>]
>>> plt.ylabel(r'$\beta/10^{-4} [K^{-1}]$')
Text(0, 0.5, '$\\beta/10^{-4} [K^{-1}]$')
>>> plt.show()
>>>
```

Código 2.12: Ejemplo de uso de matplotlib mediante la interfaz tipo matlab.

Aunque python fue diseñado para ser un lenguaje de programación para cualquier propósito en general, estos cuatro módulos (entre otras que se pueden revisar en https://pypi.org) hacen que python sea una herramienta que permite resolver problemas científicos de manera muy eficiente.

CAPÍTULO 3

El módulo pICNIK

pICNIK[1] es un módulo con dos clases: *DataExtraction* y *ActivationEnergy*. La primera tiene métodos para extraer la información de los archivos que contienen los datos experimentales y organizarlos de manera conveniente, además de un método para guardar los datos de energía de activación como archivos .csv o .xlsx. La segunda tiene métodos para usar la información ya organizada para realizar los cálculos de energía de activación, utilizando los tratamientos de isoconversión de Friedman (ecuación (1.10)), OFW (ecuación (1.14)), KAS (ecuación (1.15)), Vyazovkin (ecuación (1.20)) y Vyazovkin avanzado (ecuación (1.23)). En esta sección se describen los procesos de diseño y desarrollo del módulo, así como un ejemplo de uso con termogramas simulados de tres procesos distintos.

3.1. Diseño

Para desarrollar un programa se parte de la pregunta: ¿qué tiene que hacer el programa? En este caso el programa tiene que ser una función tal que reciba n archivos con información de las termogravimetrías de algún analito de interés a n diferentes programas de calentamiento y regrese la energía de activación (E_α) en función de la conversión del proceso (α), a partir de métodos de isoconversión, como se ilustra en el esquema siguiente.

Figura 3.1: La función a programar debe ser tal que obtenga datos de varios experimentos de termogravimetría para una muestra y regrese la energía de activación por medio de métodos de isoconversión.

El primer paso que tiene que realizar esta función es extraer la información necesaria de múltiples archivos de termogravimetría para realizar los cálculos, estos son: el tiempo (t), la temperatura al tiempo t ($T = T(t)$) y la masa al tiempo t ($m = m(t)$). Después, esta información se puede acomodar en orden ascendente de razón de calentamiento (β). A partir de eso, se puede calcular el valor de conversión al tiempo t ($\alpha = \alpha(t)$) con la ecuación (1.4) y la tasa de conversión ($\frac{\mathrm{d}\alpha}{\mathrm{d}t}$) con métodos de diferenciación numérica.

Ahora, para aplicar los métodos de isoconversión, la función debe alinear los conjuntos de datos de t, T y $\frac{\mathrm{d}\alpha}{\mathrm{d}t}$ entre los n experimentos, a los mismos valores de conversión.

[1] python package for Isoconversional Computations for Non-Isothermal Kinetics: "paquetería de python para cálculos de isoconversión para cinética anisotérmica"

Habiendo logrado organizar los datos en función de un conjunto de valores de conversión, para calcular las energías de activación por los métodos de aproximación lineal (Friedman, OFW y KAS) hay que realizar la regresión lineal para los pares: $\ln\left(\mathrm{d}\alpha/\mathrm{d}t\right)_{\alpha,i}$ vs $T_{\alpha,i}^{-1}$; $\ln\left(\beta_i\right)$ vs $T_{\alpha,i}^{-1}$ y $\ln\left(\beta_i/T_{\alpha,i}^2\right)$ vs $T_{\alpha,i}^{-1}$ para cada valor de conversión. Nótese que cada regresión lineal tendrá tantos puntos como programas de calentamiento se hayan utilizado, por lo que se recomienda realizar experimentos a por lo menos 3 programas de calentamiento [9].

Para los métodos no lineales de Vyazovkin, el programa tiene que resolver la integral (1.12) para el primer método o la ecuación (1.23) para el método avanzado, la doble suma de la ecuación (1.20) y minimizar la misma respecto a la energía de activación, para cada valor de conversión.

Habiendo realizado todo lo anterior es posible obtener los valores de E_α por cinco métodos de isoconversión, y se puede analizar su relación con la conversión y con la temperatura. El procedimiento descrito se ilustra en el esquema 3.2.

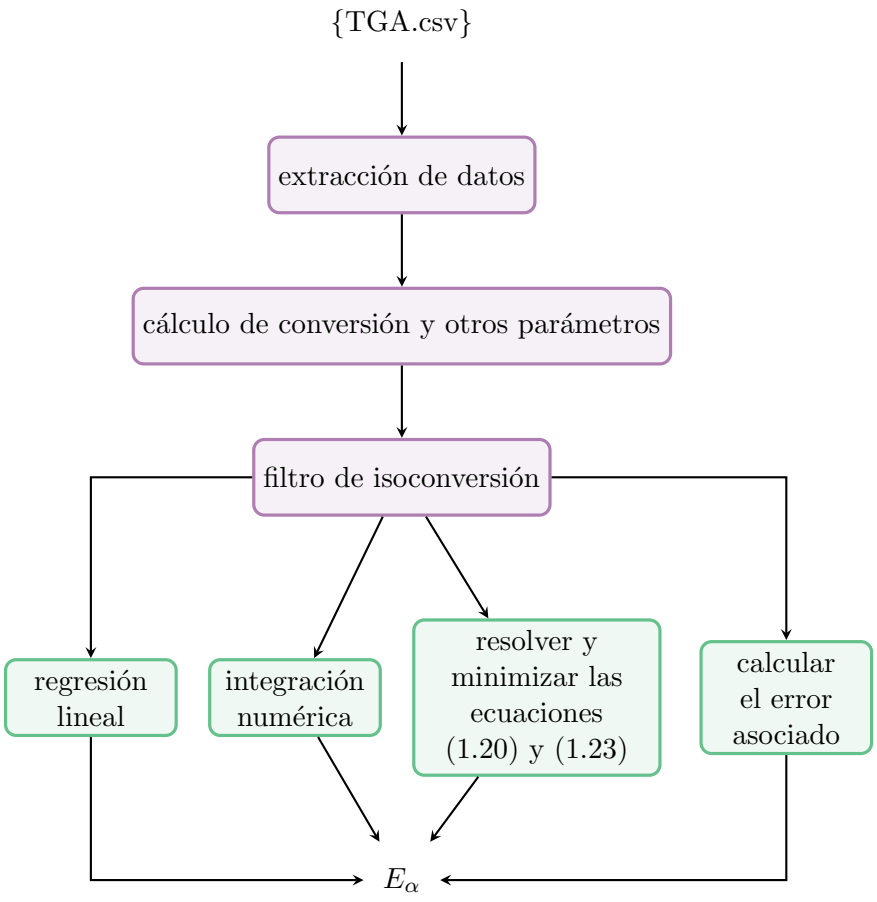


Figura 3.2: Diagrama de flujo mostrando los métodos que debe tener pICNIK para obtener la energía de activación y cómo pueden agruparse en dos clases: extracción de datos y cálculo de energía.

De la figura 3.2 se observa que la función puede dividirse en dos partes, una encargada de extraer y organizar los datos experimentales (morado) y la otra encargada de calcular las energías de activación (verde).

Dada la sintaxis de python, lo más conveniente es desarrollar un módulo con clases que contengan métodos para realizar estas tareas, ya que de ese modo se puede trabajar paso a paso

y de forma dinámica en la obtención de cada cantidad de interés durante el análisis y observar el comportamiento de cada variable experimental con apoyo de otros módulos de python, mientras se trabaja con una cantidad conveniente de puntos, o por secciones de temperatura o conversión que sean de particular interés. Como se mencionó, la clase que extrae la información es llamada *DataExtraction* y la clase que calcula las energías de activación se llama *ActivationEnergy*.

Entonces, las funciones que, por lo menos, deben llevar a cabo las clases son:

- *DataExtraction*:
  - Abrir los n archivos de TGA.
  - Calcular y organizar la temperatura absoluta, la conversión y la tasa de conversión.
  - Calcular y organizar las razones de calentamiento a partir de los valores medidos de tiempo y temperatura.
  - Filtro de isoconversión:
- *ActivationEnergy*:
  - Regresión lineal de los pares: $\ln\left(\frac{d\alpha}{dt}\right)_{\alpha,i}$ vs $T_{\alpha,i}^{-1}$; $\ln(\beta_i)$ vs $T_{\alpha,i}^{-1}$ y $\ln\left(\beta_i/T_{\alpha,i}^2\right)$ vs $T_{\alpha,i}^{-1}$.
  - Resolver las integrales (1.12) y (1.23).
  - Minimizar la ecuación (1.20).
  - Evaluación del error asociado al cálculo de enerrgía de activación.

3.2. Desarrollo

El módulo se desarrolló con los ambientes interactivos de desarrollo jupyter y spyder. Se utilizaron las dependencias externas: numpy [27], pandas [29], matplotlib [30], scipy [28] y derviative [32].

3.2.1. Dataextraction

El primer paso para la programación de objetos es definir su constructor como se mecionó en la sección 2.2. En este caso el constructor se encarga únicamente de generar al objeto, invocar sus métodos y generar atributos vacíos que se encargaran de llenar los métodos, como se muestra en el código 3.1. Los atributos de la clase incluyen ocho listas, cuya cantidad de elementos es igual al número de archivos que se le van a entregar, cinco *DataFrames* de pandas y un valor predeterminado para $\Delta\alpha$.

```
class DataExtraction(object):
    def __init__(self):
        self.DFlis          = []
        self.Beta           = []
        self.BetaCC         = []
        self.files          = []
        self.da_dt          = []
        self.T              = []
        self.t              = []
        self.TempIsoDF      = pd.DataFrame()
        self.timeIsoDF      = pd.DataFrame()
        self.diffIsoDF      = pd.DataFrame()
        self.TempAdvIsoDF   = pd.DataFrame()
        self.timeAdvIsoDF   = pd.DataFrame()
```

```
        self.alpha          = []
        self.d_a            = 0.00001
```

Código 3.1: Códgo fuente del iniciador de la clase *DataExtraction*.

Ahora, los métodos se escriben en función de los puntos de la lista que se generó en la sección anterior que describe las funciones que debe realizar la clase. Para abrir los n archivos se puede utilizar la función **read_csv()** de pandas, por lo que será necesario tener los datos experimentales como archivos .csv y tener una lista con las direcciones de cada archivo. Esa lista es conveniente tenerla guardada como atributo de la clase y para esto se programó el método **set_data()** que le asigna el valor de la lista de direcciones al atributo **files** de la clase. Al abrir cada archivo como un *DataFrame* de pandas se pueden calcular y añadir como columnas los valores de la temperatura absoluta y la conversión del proceso. Los valores de la tasa de conversión se pueden calcular con la función **dxdt()** de derivative. Finalmente, teniendo las columnas de tiempo y temperatura se puede realizar la regresión lineal con la función **linregress()** del submódulo "stats" de scipy. Los valores de las pendientes se guardan como un arreglo de numpy con el nombre **Beta**. Ahora, para tener la información organizada, se crea una lista para los *DataFrame*s creados y se le llamará **DFlis**, pero además es conveniente crear listas que contengan la información de la conversión, la temperatura, el tiempo y la tasa de conversión para cada experimento, por separado. Esto último se puede implementar con un algoritmo que, además de organizar las diferentes variables, las filtra en orden ascendente de conversión. Estas listas se asignan a los atributos **alpha**, **T**, **t** y **da_dt** respectivamente. Este procedimiento se programó como el método **data_extraction()**.

Para agregarle a la clase el filtro de isoconversión hay que obtener valores de las variables de estudio termocinético a valores establecidos de conversión, para lo que se necesita conocer cómo es la dependencia de dichas variables con la conversión, lo que se puede aproximar bien mediante interpolación cuando se tienen muchos datos y en los estudios termogravimétricos actuales se llegan a obtener hasta 18000 datos en un experimento. En este caso se implementó el cálculo de funciones de interpolación con splines cúbicos con la función **interp1d()** del submódulo "interpolate" de scipy, con lo que se obtienen las funciones:

$$\begin{aligned} T &= \phi(\alpha) \\ t &= \theta(\alpha) \\ \frac{\mathrm{d}\alpha}{\mathrm{d}t} &= \rho(\alpha) \end{aligned} \tag{3.1}$$

Estas funciones serán evaluadas para tantos arreglos de numpy como programas de calentamiento se hayan utilizado. Por lo tanto, para cada variable se puede crear un *DataFrame* en el que el índice sean los valores de conversión que se usen para evaluar las funciones 3.1, que serán los que correspondan a los obtenidos para el experimento con la menor tasa de calentamiento y las columnas serán los valores obtenidos al evaluar dichas funciones. Los *DataFrames* creados de esta manera servirán para el primer método de Vyazovkin, así como para los métodos de Friedman, KAS y OFW, pero no para el método avanzado de Vyazovkin. Para éste último, los valores de α deben ser equidistantes entre sí por un valor definido de $\Delta\alpha$. Un arreglo de valores de conversión equidistantes entre sí se puede generar con la función **arange()** de numpy, asignando los valores inicial y final del experimento con la menor rapa de calentamiento al valor inicial y al valor final del arreglo de conversiones. Ahora se pueden crear otros dos *DataFrame*s para el método avanzado, correspondiendo al tiempo y a la temperatura, que son las variables sobre las que trabaja dicho método. Estos procedimientos se programaron en las funciones **isoconversional()** y **adv_isoconversional()**.

Toda la información generada y organizada se guarda como atributos de la clase en forma de listas, arreglos de numpy y *DataFrames* de pandas y se ilustrará su uso en la sección 3.3

3.2.2. ActivationEnergy

Esta clase contiene diez atributos como se muestra en el código 3.2. Además define el valor de la constante universal de los gases (R) en $\mathrm{kJ\,mol^{-1}\,K^{-1}}$.

```
class ActivationEnergy(object):
    def __init__(self, Beta, TempIsoDF=None, diffIsoDF=None, TempAdvIsoDF=None,
    timeAdvIsoDF=None):
        self.E_Fr          = []
        self.E_OFW         = []
        self.E_KAS         = []
        self.E_Vy          = []
        self.E_aVy         = []
        self.Beta          = Beta
        self.logB          = np.log(Beta)
        self.TempIsoDF     = TempIsoDF
        self.diffIsoDF     = diffIsoDF
        self.TempAdvIsoDF  = TempAdvIsoDF
        self.timeAdvIsoDF  = timeAdvIsoDF
        self.R             = 0.0083144626
```

Código 3.2: Código fuente del iniciador de la clase *ActivationEnergy*.

La clase va a trabajar con la información organizada con *DataExtraction*, por lo que sus funciones consistirán esencialmente en realizar los cálculos correspondientes.

Para los métodos lineales de Friedman, KAS y OFW se implementaron regresiones lineales haciendo uso nuevamente de la función **interp1d()** del submódulo "interpolate" de scipy. El error asociado al valor de energía de activación se calcula al 95 % de confianza para una distribución t en el cálculo de la pendiente de la regresión lineal con el submódulo "stats" de scipy. El nombre de cada método dentro de la clase es **Fr()**, **KAS()** y **OFW()**.

Para los métodos de Vyazovkin se implementaron diferentes tipos de integración las funciones del submódulo integrate de scipy. Para el primer método de Vyazovkin se implementó un algoritmo con la aproximación de Senum-Yang (ecuación (1.21)) así como la regla del trapezoide y un método de la librería QUADPACK de Fortran implementado en scipy como la función **quad()**. Para el método avanzado se implementó la ecuación (1.24) para integrar sobre la temperatura y los métodos numéricos del trapezoide, la regla de Simpson y la función quad para integrar sobre el tiempo. Después de implementar las integrales (1.12) y (1.23) se programó la función (1.19) y su minimización con la función **minimize_scalar()** del submódulo "optimize" de scipy. Esta minimización depende de un intervalo de evaluación que debe contener el mínimo, para poder decidir si los límites de evaluación son o no pertienentes se implementó la visualización de la función (1.19) a un valor de conversión, para ambos métodos de vyazovkin, mediante el uso del módulo matplotlib.

El cálculo del error asociado a los métodos de Vyazovkin, con un 95 % de confianza, se implementó con base en la metodología propuesta y validada Por S. Vyazovkin y C. Wight [33]

Finalmente se implementó una función para regresar como archivos .csv o .xlsx los valores obtenidos de la energía de activación en función de la conversión como método de la clase *DataExtraction*, ya que esta clase es la que se encarga de recibir los archivos.

El esquema del uso de las clases de picnik se ilustra en la figura 3.3.

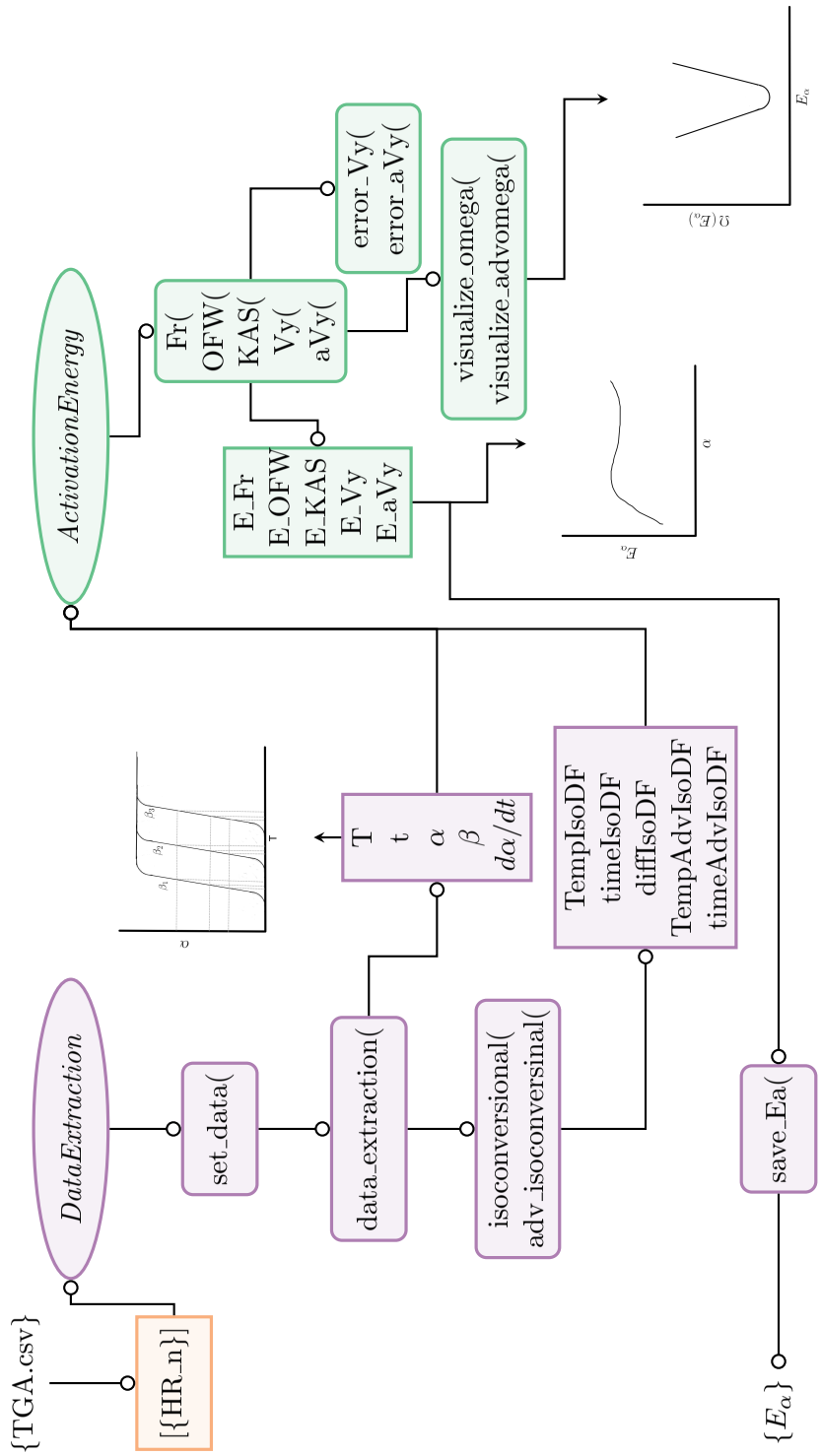


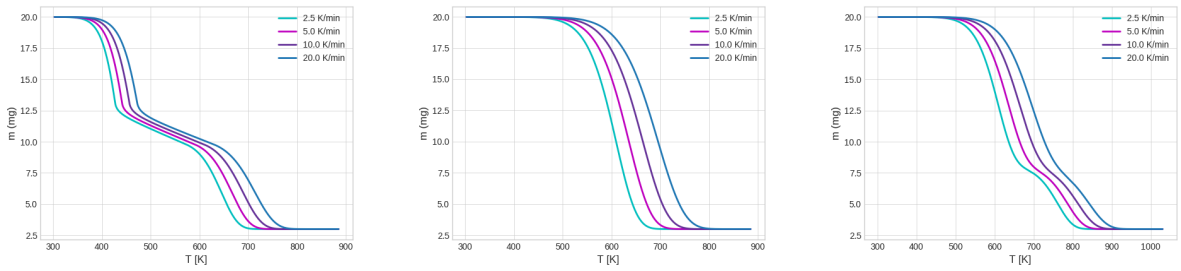
Figura 3.3: Diagrama de flujo describiendo el uso de pICNIK para obtener la energía de activación.

3.3. Ejemplo de uso

Con el fin de describir el código e ilustrar la intención del uso de la paquetería se simularon termogramas a cuatro razones de calentamiento (2.5, 5, 10 y 20 $\mathrm{K\,min^{-1}}$), utilizando el modelo F1 ($f(\alpha) = 1 - \alpha$) y la ecuación (1.27), para tres procesos hipotéticos:

1. Un paso con energía de activación variable ($E_\alpha = 70\,\mathrm{kJ\,mol^{-1}}$ para $\alpha < 0.4$, luego varía linealmente en el intervalo $0.4 \leq \alpha \leq 0.6$ y $105\,\mathrm{kJ\,mol^{-1}}$ para $\alpha > 0.6$, $\ln A_\alpha$ =17);
2. Un paso con energía de activación constante ($E_\alpha = 75\,\mathrm{kJ\,mol^{-1}}$, $\ln A_\alpha$ =12) y;
3. Dos pasos, cada uno con energía de activación constante ($E_{\alpha,1} = 75\,\mathrm{kJ\,mol^{-1}}$,$\ln A_{\alpha,1}$ =12 y $E_{\alpha,2}$=125 $\mathrm{kJ\,mol^{-1}}$,$\ln A_{\alpha,2}$ =17)

Los termogramas simulados se muestran en la figura 3.4.



(a) (b) (c)

Figura 3.4: Termogramas de los tres sistemas simulados: (a) E_α variable; (b) E_α constante y; (c) dos valores de E_α.

La paquetería requiere tener una versión python mayor a la 3.8 y menor a 3.10, además de las paqueterías numpy, pandas, matplotlib, scipy y derviative. La instalación se puede realizar desde PyPI[2] o con el comando pip desde la terminal:

```
pip install picnik
```

Código 3.3: Instalación de pICNIK

Es conveniente crear una lista que contenga la dirección de cada archivo que se va a usar. Nótese que los archivos deben ir en orden ascendente de razón de calentamiento:

```
>>> f_var = ['Evar2.5.csv','Evar5.csv','Evar10.csv','Evar20.csv']
>>> f_cnt = ['Ecnt2.5.csv','Ecnt5.csv','Ecnt10.csv','Ecnt20.csv']
>>> f_2S  = ['E2S2.5.csv','E2S5.csv','E2S10.csv','E2S20.csv']
```

Código 3.4: Dirección de los archivos de los datos simulados, organizados en listas de python

Después se crea una instancia del objeto extractor:

```
>>> import picnik as pnk
>>> xtr = pnk.DataExtraction()
```

Código 3.5: Instancia del objeto DataExtraction.

[2]https://pypi.org/

xtr es un objeto instanciado de la clase **DataExtraction** y siendo estrictos, se tendría que crear un objeto por cada muestra, ya que los atributos de cada clase son individuales y no puede tener varios al mismo tiempo. Por lo tanto, se tendrían que construir los objetos xtr_var, xtr_cnt y xtr_2S, o se puede trabajar una muestra a la vez. Con el fin de evitar redundancias, utilizaré el objeto xtr de forma general e iré ilustrando los resultados que se obtienen para cada sistema, entendiéndose que xtr puede ser cualquiera de xtr_var, xtr_cnt y xtr_2S. Para instanciar al objeto no es necesario ningún argumento. Lo siguiente es definir para la clase qué archivos que se van a utilizar, esto se logra con el método **set_data()**, que recibe como argumento la lista con las direcciones de los archivos csv. El método **data_extraction()** tiene como parámetro opcional "encoding" que indica el conjunto de caracteres en que están escritos los archivos en **files**, la opción predeterminada es utf-8.

```
>>> xtr.set_data(f_X)          # X = cnt, var o 2S
>>> xtr.data_extraction(encoding='utf8')
```

Código 3.6: Métodos del objeto *DataExtraction* para extraer la información experimental.

Al utilizar **data_extraction()** se le asignan valores a los atributos **DFlis**, **Beta**, **T**, **t**, **alpha**, y **da_dt**. Teniendo estos atributos se pueden obtener las gráficas que se muestran en la figura (3.5), pero también se pueden obtener con los métodos **get_avsT_plot()** (α vs Temperatura), **get_avst_plot()** (α vs tiempo), **get_dadtvsT_plot()** ($d\alpha/dt$ vs Temperatura) y **get_dadtvst_plot()** ($d\alpha/dt$ vs tiempo).

El siguiente paso sería crear los *DataFrames* de isoconversión con los método **isoconversional()** y **adv_isoconversional()**.

```
>>> xtr.isoconversional()
>>> xtr.adv_isoconversional(method='points', N=len(xtr.TempIsoDF.index))
>>> xtr.TempIsoDF
          HR 1.0 K/min  HR 1.5 K/min  HR 2.3 K/min  HR 3.5 K/min  HR 5.1 K/min
0.000000       298.4976      295.8371      297.8418      298.8092      294.6963
0.000061       298.5121      295.8426      297.8653      298.8207      294.6985
0.000126       298.5096      295.8469      297.8883      298.8366      294.7013
...                 ...           ...           ...           ...           ...
1.000004       519.4427      505.7866      409.8518      437.5472      516.4525
1.000005       519.7242      505.8616      409.4255      437.5607      519.5083
1.000009       520.1861      505.7890      411.1630      437.8749      519.5492

[2574 rows x 5 columns]
>>> xtr.timeAdvIsoDF
          HR 1.0 K/min  HR 1.5 K/min  HR 2.3 K/min  HR 3.5 K/min  HR 5.1 K/min
0.000000        0.0028        0.0009        0.0032        0.0037        0.0039
0.000389        0.0554        0.0472        0.0362        0.0353        0.0439
0.000777        0.1141        0.1002        0.0727        0.0736        0.0919
...                ...           ...           ...           ...           ...
0.999232      138.5286       73.0465       27.1360       21.4720       24.3568
0.999621      169.3704       92.2748       33.5341       27.0563       30.0027
1.000009      222.8257      139.1200       49.2051       39.7826       45.0042

[2574 rows x 5 columns]
>>> xtr.d_a
0.0003887
>>>
```

Código 3.7: Construcción de los *DataFrames* de isoconversión.

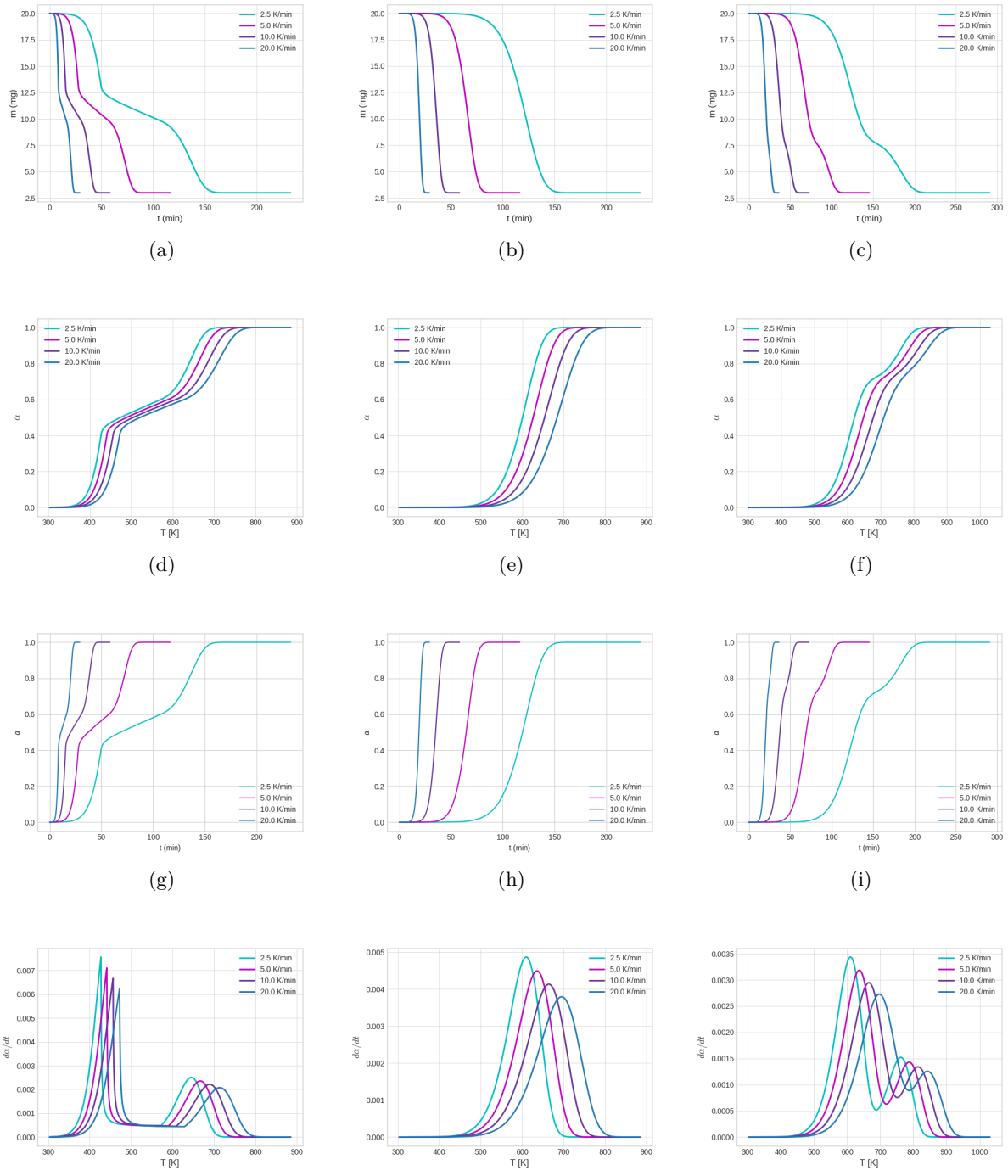


Figura 3.5: Visualización de la información extraída de la termogravimetría para cada sistema simulado. De izquierda a derecha se muestran los sistemas de energía de activación variable, constante y de dos pasos respectivamente. De arriba hacía abajo se muestran las gráficas de m [mg] *vs.* t [min], α *vs.* T [K], α *vs.* t [min] y $\mathrm{d}\alpha/\mathrm{d}t$ *vs.* T [K].

Con el primer método se asignan valores a tres atributos: **TempIsoDF**, **timeIsoDF** y **diffIsoDF**. Cada *DataFrame* comparte el mismo índice que es el elemento del atributo **alpha** con menos puntos. La información se guarda en los atributos.

El método **adv_isoconversional()** crea dos *DataFrames* de isoconversión para el método avanzado de Vyazovkin, uno de temperaturas y otro de tiempos, estos se crean aparte ya que requieren que el valor de $\Delta\alpha$ sea constante, lo que no está asegurado para los datos experimentales, por lo que se genera una lista de valores equidistantes de α. Se puede construir el arreglo de α a partir del número de puntos que se deseen en el arreglo, o a partir de un valor para el intervalo $\Delta\alpha$ con el parámetro "method" que acepta dos opciones: "points" e "interval" ("points" es la opción predeterminada). Si se selecciona la opción "points" se debe definir el parámetro "N" que indica la cantidad de puntos en el arreglo. Si se selecciona "interval" se debe determinar el parámetro "da" que indica el tamaño del intervalo entre α_n y α_{n+1}, los valores inicial y final de α corresponden con los valores inicial y final del índice de **TempIsoDF**. Después, se generan funciones interpoladas de T vs α y de t vs α para cada elemento de los atributos **t** y **T**. La función se evalúa para cada elemento del arreglo equidistante de α, y así obtener valores de temperatura y tiempo que correspondan a dichas α's. La información queda guardada en los atributos **TempAdvIsoDF** y **timeAdvIsoDF**, el valor de $\Delta\alpha$ se guarda el atributo **d_a**.

Ya que la información está organizada bajo el criterio de isoconversión, los cálculos de energía de activación se realizan con los métodos de la clase *ActivationEnergy*. Para instanciar un objeto de esta clase se necesitan cinco argumentos obligatorios. El primero corresponde a una lista de valores de tasa de calentamiento (**Beta**), los siguientes dos son *DataFrames* de isoconversión, de temperatura (**TempIsoDF**) y de tasa de conversión (**diffIsoDF**), los últimos dos son *DataFrames* de isoconversión para el método avanzado, de temperatura (**TempAdvIsoDF**) y de tiempo (**timeAdvIsoDF**), en ese orden. El constructor toma los cinco argumentos y los asigna como valores a los correspondientes atributos.

```
>>> ace = pnk.ActivationEnergy(xtr.Beta,
                               xtr.TempIsoDF,
                               xtr.diffIsoDF,
                               xtr.TempAdvIsoDF,
                               xtr.timeAdvIsoDF)
>>> E_Fr  = ace.Fr()
>>> E_OFW = ace.OFW()
>>> E_KAS = ace.KAS()
```

Código 3.8: Instancia del objeto ActivationEnergy y cálculo de energías de activación por los métodos Fr, OFW y KAS.

Los métodos de aproximación lineal se invocan directamente con los métodos **Fr()**, **OFW()** y **KAS()**, obteniendo de regreso una tupla que contiene un arreglo con las energías de activación y otro con el error asociado con un 95 % de confianza.

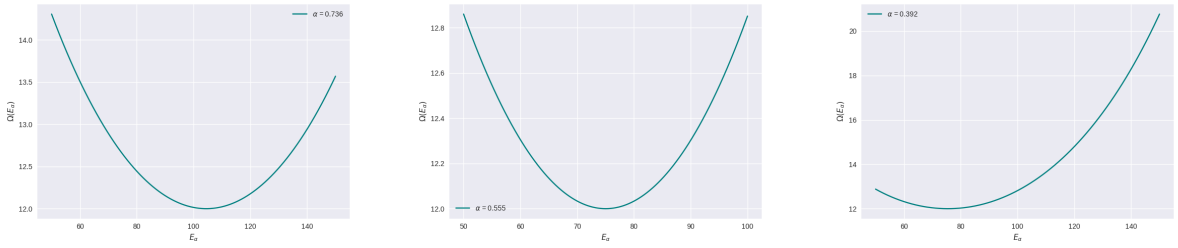
El siguiente método es el no-lineal de Vyazovkin, como mencioné en la sección 1.3.2, la energía de activación se obtiene al minimizar la ecuación (1.20). Para dicha minimización es útil visualizar la función $\Omega(E_\alpha)$ y poder observar dentro de qué intervalo se encuentra el mínimo ya que lo que tarde el cálculo depende de cuántos valores posibles de E_α se tienen que evaluar. El método **visualize_omega()** requiere como argumentos: un número entero, correspondiente al índice del valor de conversión (α_i) en el *DataFrame* **TempIsoDF** para el cuál se desea evaluar $\Omega(E_\alpha)$; y como parámetros opcionales están "bounds", el intervalo de E_α que debe ser una tupla, con valor preestablecido de (1,300), que se puede modificar; "N", el número de puntos para construir la curva $\Omega(E_\alpha)$ vs E_α, con valor predeterminado de 1000 y; "method" el método para resolver la integral (1.12), las opciones son: 'senum-yang', 'trapezoid' y 'quad'; "senum-

yang" es la opción predeterminada. El resultado es una gráfica de matplotlib indicando el valor de conversión para el que se realizó el cálculo. En la figura 3.6 se muestra el resultado de aplicar el método **visualize_omega()** para los sistemas simulados usando los índices 690, 420 y 420 para el sistema variable, el constante y el de dos pasos respectivamente.

Es útil hacer esta evaluación para conversiones que cubran la mayoría del proceso como, por ejemplo, 0.1, 0.5 y 0.9 ya que para la mayoría de los casos reales, la energía de activación varía con la conversión.

```
>>> ace.visualize_omega(row, bounds=(1,300), N=1000, method='senum-yang')
```

Código 3.9: Método para visualizar la ecuación (1.12).



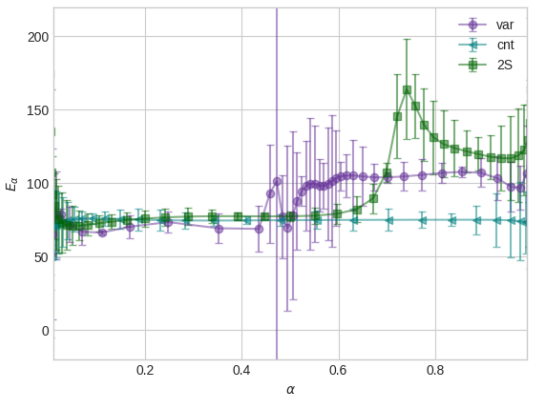
(a) var (b) cnt (c) 2S

Figura 3.6: Evaluación de los límites de E_α para $\Omega(E_\alpha)$.

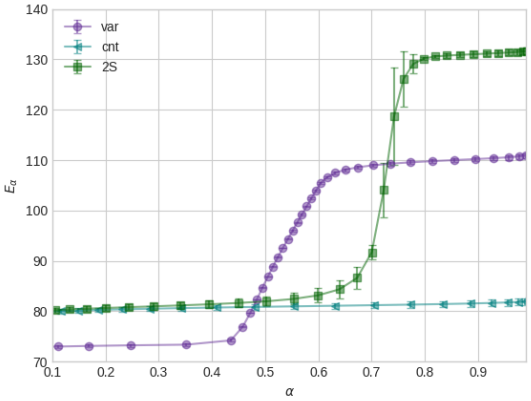
Una vez que se estableció el intervalo para evaluar $\Omega(E_\alpha)$, se calcula la energía de activación con el método **Vy()**, que requiere como argumento obligatorio el intervalo de evaluación, y tiene como parámetro opcional "method" que recibe las mismas opciones que en el método de visualización, esto es porque ambos métodos recurren a otra función implementada dentro de la clase que corresponde a la ecuación (1.20). El método básicamente minimiza esta ecuación para cada renglón del *DataFrame* mediante el método **minimize_scalar()** de scipy. Para calcular una estimación del error asociado al 95 % de confianza bajo una distribución t, se implementó un método validado [33] la función **error_Vy()**. Finalmente está el método avanzado de Vyazovkin que es prácticamente igual que el método anterior, solo que puede evaluar tanto la integral (1.23) como la integral (1.26). Los parámetros del método **aVy()** son: "bounds", con el mismo propósito que para el método **Vy()**; "var", acepta las opciones 'time", para evaluar la integral (1.26), y "Temperature" para la integral (1.23). En caso de seleccionar "Temperature", la integral se resuelve con la ecuación (1.24). Si se elige la opción "time", también hay que determinar el parámetro "method" que indica el método numérico para resolver la integral (1.26), las opciones disponibles son "trapezoid", "simpson" y "quad". La clase *ActivationEnergy* también tiene un método de visualización para la función a minimizar, **visualize_advomega()**.

```
>>> E_aVy = ace.Vy(bounds = (1,150),
                   method ='senum-yang')
>>> E_Vy_error = ace.error_Vy()
>>> E_aVy = ace.aVy(bounds = (1,150),
                    var = 'time',
                    method ='trapezoid')
>>> E_aVy_error = ace.error_aVy()
```

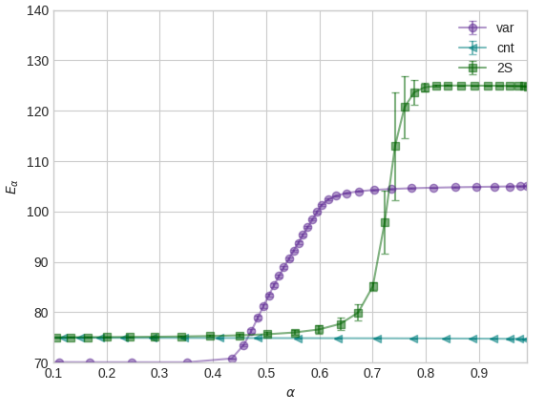
Código 3.10: Cálculo de la energía de activación por los métodos de Vyazovkin con su error asociado.



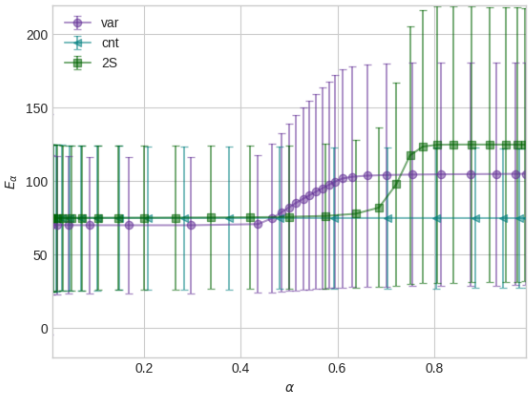
(a) Friedman



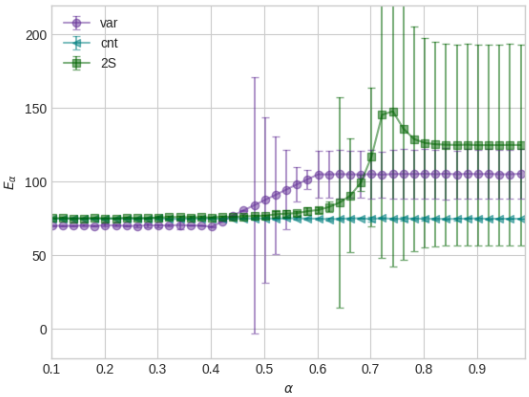
(b) OFW



(c) KAS



(d) Vy



(e) aVy

Figura 3.7: Energías de activación calculadas para los sistemas simulados. (a) Método de Friedman; (b) Método de Ozawa-Flynn-Wall; (c) Método de Kissinger-Akahira-Sunose; (d) Método de Vyazovkin y (e) Método avanzado de Vyazovkin.

Los resultados obtenidos para los sistemas simulados se muestran en la figura (3.7), en la que se puede observar que el método diferencial de Friedman es el más ruidoso. Los métodos KAS y OFW parecen tener nula incertidumbre para estos sistemas ideales. Sin embargo el método KAS da resultados más cercanos a los valores programados que el método OFW, que sobrestima un poco la energía de activación. Los resultados obtenidos por los métodos Vy y aVy también son consistentes con los valores programados, aunque los errores asociados al primer método de Vyazovkin son tan grandes que se puede argumentar que los resultados no son confiables.

Todos los métodos reproducen con un buen nivel de precisión, en promedio, los valores impuestos para las simulaciones, aunque los errores para cada método parecen poco confiables ya que son o muy grandes o inconsistentes. En la tabla (3.1) se observa que todos los métodos calculan el valor programado de energía de activación para el sistema de energía constante, a excepción del método OFW que la sobrestima por unos $6\,\mathrm{kJ\,mol^{-1}}$, y que los métodos con mayor error para este sistema son Fr y Vy.

Tabla 3.1: Energías de activación, errores promedio y errores relativos (comparados con el valor programado) del sistema simulado con $E_\alpha = 75\,\mathrm{kJ\,mol^{-1}}$, en un intervalo de conversión de 0.1 a 0.995.

Método	$\langle E_\alpha \rangle$ [$\mathrm{kJ\,mol^{-1}}$]	$\langle \mathrm{e} \rangle$ [$\mathrm{kJ\,mol^{-1}}$]	$\mathrm{e_{rel}}$ [%]
Fr	75	10 6	0
OFW	81.0	0.6	8
KAS	74.83	0.04	0.2
Vy	75	50	0
aVy	75.0	0.5	0

Por último, la información se puede guardar en archivos .csv o .xlsx con el método **save_df()**. Los parámetros son “E_Fr”, “E_OFW”, “E_KAS”, “E_Vy” y “E_aVy” y sus opciones son “None” (predeterminado) o el arreglo que contenga la información del respectivo método; y “file_t” que indica el tipo de archivo que se va a crear, las opciones son “csv” y “xlsx”.

```
>>> xtr.save_Ea(E_Fr = E_Fr[0],
                E_OFW = E_OFW[0],
                E_KAS = E_KAS[0],
                E_Vy = E_Vy,
                E_aVy = E_aVy,
                file_t="xlsx" )
```

Código 3.11: Método para guardar los datos de energía de activación calculada.

CAPÍTULO 4

Caso de estudio: n-decano

A finales del 2020 se publicaron los resultados de Bruno Ekawa, Victoria L. Stanford y Sergey Vyazovkin sobre el estudio de la cinética de evaporación del decano con métodos de isoconversión, por mediciones termogravimétricas [34]. Los autores utilizan la ecuación de Langmuir para modelar la evaporación del decano dentro de un flujo de gas inerte, como lo es en un sistema de termogravimetría:

$$-\frac{dm}{dt} = p\frac{DM}{zRT} \tag{4.1}$$

donde: p es la presión de vapor del decano; z es la distancia de la superficie del líquido al flujo de gas; R, la constante de los gases, M la masa molar del líquido, T la temperatura absoluta y D es el coeficiente de difusión del vapor en el gas inerte. Si se utiliza la definición de conversión, ecuación (1.4), con $\zeta = m$ y la presión de vapor como el modelo de Clausius-Clapeyron, se llega a la siguiente expresión:

$$\frac{d\alpha}{dt} = \frac{CDM}{\Delta mhRT}\exp\left(-\frac{\Delta H_v}{RT}\right) \tag{4.2}$$

donde: C es una constante de integración; $\Delta m = m_0 - m_f$ y ΔH_v es la entalpía de evaporación del decano. Para aplicar el criterio de isoconversión, se aplica el logaritmo natural a la ecuación (4.2) y se deriva respecto a la temperatura recíproca a conversión constante:

$$\ln\left(\frac{\mathrm{d}\alpha}{\mathrm{d}t}\right) = \ln\left(\frac{CDM}{\Delta mhR}\right) + \ln\left(\frac{1}{T}\right) - \frac{\Delta H_v}{RT} \tag{4.3}$$

$$\left[\frac{\partial \ln\left(\frac{\mathrm{d}\alpha}{\mathrm{d}t}\right)}{\partial T^{-1}}\right]_\alpha = T - \frac{\Delta H_v}{R} \tag{4.4}$$

De la ecuación (1.9), se observa que al multiplicar por -R se obtiene una expresión analítica para la energía de activación:

$$E_\alpha = -R\left[\frac{\ln\left(d\alpha/dt\right)}{dT^{-1}}\right]_\alpha = \Delta H_v - RT \tag{4.5}$$

El término RT obtiene un valor de $2\,\mathrm{kJ\,mol^{-1}}$ a $3\,\mathrm{kJ\,mol^{-1}}$ durante el intervalo de temperatura del experimento, por lo que la energía de activación debería ser aproximadamente constante y muy similar a la entalpía de evaporación del decano, cuyo valor promedio[1] es de $47.4\,\mathrm{kJ\,mol^{-1}}$.

[1] Dato obtenido del NIST Chemistry Webook https://webbook.nist.gov revisado el 6 de Agosto del 2021 a las 20h18.

Las condiciones experimentales que utilizaron Ekawa y su equipo fueron: razóns de calentamiento de 1.0, 1.5, 2.3, 3.5 y 5.0 $\mathrm{K\,min^{-1}}$; masas iniciales de aproximadamente 17.5 mg; flujo de N_2 de 80 $\mathrm{ml\,min^{-1}}$. Además utilizaron como método de cálculo el método avanzado de Vyazovkin y sus resultados coinciden con un comportamiento prácticamente constante para la energía de activación en función de la conversión, con un valor promedio de $51.6 \pm 1.3\,\mathrm{kJ\,mol^{-1}}$.

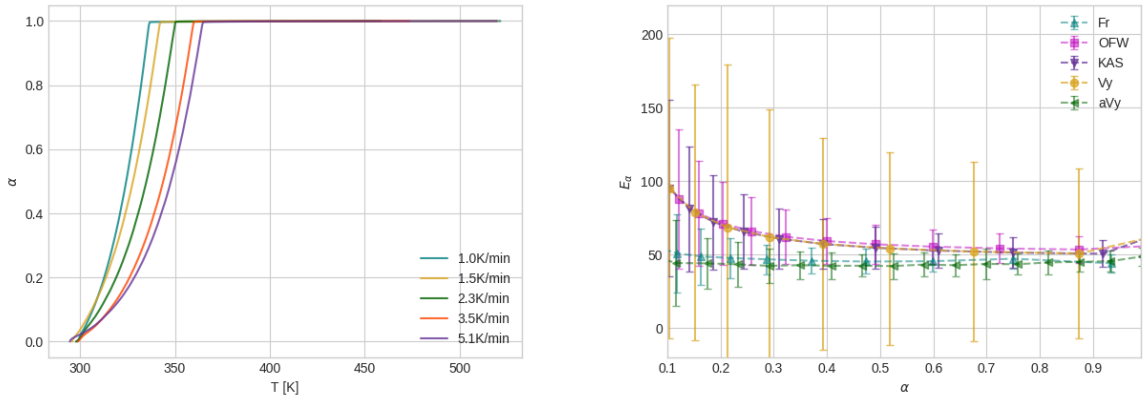
Como caso de estudio y para validar la paquetería desarrollada en este proyecto, se estudió la evaporación del n-decano por medio de termogravimetría y se compararon los resultados obtenidos con los recién mencionados.

4.1. Detalles experimentales

El n-decano fue adquirido de Sigma Aldrich con pureza $\geq 99\,\%$ y se utilizó sin purificación previa. El punto de ebullición del n-decano es 174 °C. Las mediciones termogravimétricas se realizaron con el instrumento SDT Q600 de TA Instruments, a cinco diferentes razones de calentamiento nominales, a saber: 1.0, 1.5, 2.3, 3.5 y 5.0 $\mathrm{K\,min^{-1}}$, con masas iniciales de 12.0 ± 1.4 mg en portamuestras de alúmina. Todas las mediciones se realizaron bajo un flujo de N_2 de 100 $\mathrm{ml\,min^{-1}}$ desde temperatura ambiente hasta 250 °C.

4.2. Resultados y Discusión

La termogravimetría de la evaporación del decano se muestra en la figura (4.1a) como la conversión en función de la temperatura. El perfil de conversión muestra un solo paso aparente en la conversión y es consistente con lo reportado por Ekawa y su equipo. El procedimiento para tratar los datos de la termogravimetría del n-decano se muestran en el código 4.1. En la figura (4.1b) se muestran las energías de activación obtenidas con su error asociado, calculado como se describió en la sección 3.2.2.



(a) (b)

Figura 4.1: Estudio cinético de la evaporación del n-decano por termogravimetría (a) conversión vs temperatura; (b) Energía de activación vs conversión.

Con 16 comandos y un par de minutos se calculó la energía de activación del proceso térmico por cinco diferentes métodos de isoconversión. Los resultados obtenidos con el método de Friedman coinciden en prácticamente todo el intervalo con los obtenidos con el método avanzado de

Vyazovkin, esto es lógico ya que ambos métodos son los que menos suposiciones requieren para realizar los cálculos. Ambos métodos muestran que la energía de activación es prácticamente constante durante todo el intervalos de conversión, como lo reportan Ekawa y su equipo. En cambio, los métodos que requieren de aproximaciones para resolver la integral (1.12), muestran una ligera sobrestimación de la energía de activación, en comparación con los dos métodos mencionados antes, y una tendencia creciente de la energía de activación a conversiones pequeñas. Aunque los tres métodos dan resultados similares, el primer método de Vyazovkin tiene un error asociado mucho mayor.

```
>>> import picnik as pnk.

>>> files = ['DEC 1.csv','DEC 1.5.csv','DEC 2.3.csv', 'DEC 3.5.csv','DEC 5.csv']

>>> xtr = pnk.DataExtraction()
>>> xtr.set_data(files)
>>> xtr.data_extraction()

>>> xtr.isoconversional()
>>> xtr.adv_isoconversional(method='points',
                            N = len(xtr.TempIsoDF))

>>> ace = pyace.ActivationEnergy(xtr.Beta,
                                 xtr.TempIsoDF,
                                 xtr.diffIsoDF,
                                 xtr.TempAdvIsoDF,
                                 xtr.timeAdvIsoDF,)
>>> E_fr  = ace.Fr()
>>> E_ofw = ace.OFW()
>>> E_kas = ace.KAS()
>>> bounds  = (5,120)
>>> E_vy = ace.Vy(bounds)
>>> E_avy =  ace.aVy(bounds)
>>> e_vy, e_avy = ace.error_Vy(), ace.error_aVy()

>>>>xtr.save_df(E_Fr = E_fr[0],
                E_OFW = E_ofw[0],
                E_KAS = E_kas[0],
                E_Vy = E_vy,
                E_aVy = E_avy, file_t="xlsx" )
```

Código 4.1: Tratamiento de datos de la termogravimetría de la evaporación del n-decano a través de la librería pICNIK.

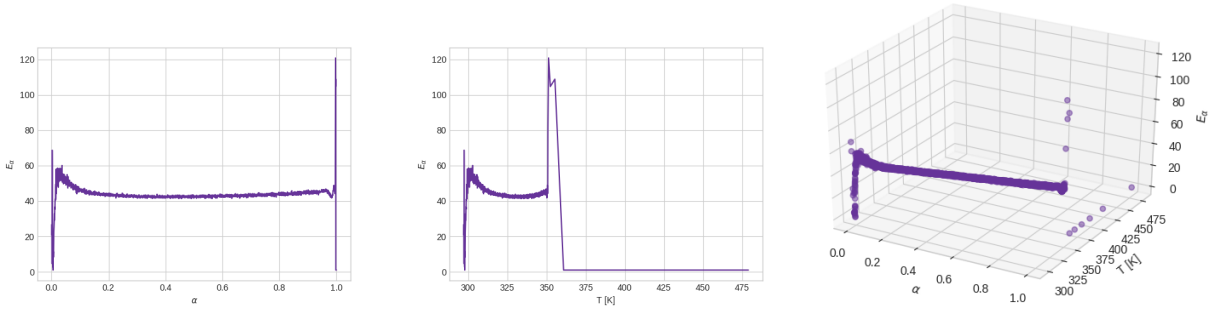
En la tabla (4.1) se muestran los valores promediados de la energía de activación y su error asociado. Se observa que el error calculado para el primer método de Vyazovkin es el mayor de todos, seguido por los métodos KAS y OFW, y los errores de los métodos de Friedman y avanzado de Vyazovkin son los menores aunque relativamente grandes aún. En la experiencia de este trabajo se ha observado que la dispersión de las masas iniciales es un factor muy importante cuando se hacen análisis por métodos de isoconversión ya que pueden afectar significativamente el resultado final. La dispersión de masas iniciales en este trabajo resultaron grandes y puede ser esa dispersión la responsable de los valores tan grandes de error asociado a cada método. Comparando directamente el resultado obtenido por el método avanzado de Vyazovkin con el valor reportado se observa una diferencia notable de $8\,\mathrm{kJ\,mol^{-1}}$. Sin embargo, al comparar ambas cantidades con el valor reportado de $47\,\mathrm{kJ\,mol^{-1}}$, se observa que ambas difieren por $4\,\mathrm{kJ\,mol^{-1}}$ del valor experimental de la entalpía de evaporación del decano, por lo que se puede concluir que

los resultados son coherentes y el módulo pICNIK es válido para realizar cálculos de isoconversión para determinaciones termogravimétricas.

Tabla 4.1: Energías de activación errores promedio para la evaporación del decano en un intervalo de conversión de 0.1 a 0.990.

Método	$\langle E_\alpha \rangle$ [kJ mol^{-1}]
Fr	47 ± 10
OFW	59 ± 20
KAS	57 ± 21
Vy	45 ± 58
aVy	43 ± 9

En la figura (4.2) se muestran los perfiles de la energía de activación, obtenidos por el método avanzado de Vyazovkin, contra la conversión y la temperatura. Nótese que el perfil de temperatura tiene valores únicamente en el intervalo en que el que se observa la evolución de la conversión en la figura (4.1a). También se nota la misma tendencia reportada de un comportamiento constante para la energía de activación y esencialmente similar a la entalpía de evaporación del decano.



(a) (b) (c)

Figura 4.2: Perfiles de energía de activación, obtenidos con el método avanzado de Vyazovkin, para la evaporación del decano en función de (a) conversión; (b) Temperatura y; (c) Ambas.

Conclusiones

- Se ha diseñado y desarrollado un módulo de python con los métodos de isoconversión de Friedman, OFW, KAS, Vyazovkin y Vyazovkin avanzado como medios para calcular la energía de activación de procesos térmicamente estimulados a través de datos experimentales de termogravimetría.
- El código es funcional. Adicionalmente, se puede aprovechar por la comunidad científica en general y de análisis térmico en particular, por su programación orientada a objetos y licencia de software libre, ya sea para análisis termocinético o para desarrollar una paquetería más completa.
- La validación del código se ha realizado con el estudio de la evaporación del n-decano y mediante la simulación de procesos térmicos.

Bibliografía

[1] T Ozawa, T Sunose y T Kaneko. «Historical review on research of kinetics in thermal analysis and thermal endurance of electrical insulating materials: I. Thermal endurance test and isoconversion methods». En: *Journal of Thermal Analysis and Calorimetry* 44.1 (1995), págs. 205-216.

[2] ME Brown y col. En: *Thermochimica Acta* 355.1-2 (2000), págs. 125-143.

[3] Enelio Torres-Garcia y Paola Brachi. «Non-isothermal pyrolysis of grape marc». En: *Journal of Thermal Analysis and Calorimetry* 139.2 (2020), págs. 1463-1478.

[4] E. Torres-García, L. F. Ramírez-Verduzco y J. Aburto. «Pyrolytic degradation of peanut shell: activation energy dependence on the conversion». En: *Waste Management* 106 (2020), págs. 203-212.

[5] JA Caballero y col. En: *Journal of Analytical and Applied Pyrolysis* 42.2 (1997), págs. 159-175.

[6] Maider Amutio y col. «Kinetic study of lignocellulosic biomass oxidative pyrolysis». En: *Fuel* 95 (2012), págs. 305-311.

[7] Xu Gao, Lin Jiang y Qiang Xu. En: *Journal of hazardous materials* 386 (2020), pág. 121645.

[8] Martina Maria Calvino y col. «Non-isothermal thermogravimetry as an accelerated tool for the shelf-life prediction of paracetamol formulations». En: *Thermochimica Acta* 700 (2021), pág. 178940.

[9] Sergey Vyazovkin y col. «ICTAC Kinetics Committee recommendations for performing kinetic computations on thermal analysis data». En: *Thermochimica Acta* 520.1 (2011), págs. 1-19.

[10] M Brown. «Steps in a minefield: Some kinetic aspects of thermal analysis». En: *Journal of Thermal Analysis and Calorimetry* 49.1 (1997), págs. 17-32.

[11] J. Šesták. «Philosophy of non-isothermal kinetics». En: *Journal of Thermal Analysis and Calorimetry* 16.2 (1979), págs. 503-520.

[12] David. Dollimore. «Thermal analysis». En: *Analytical Chemistry* 60.12 (1988). PMID: 3046422, págs. 274-279.

[13] Homer E Kissinger. «Reaction kinetics in differential thermal analysis». En: *Analytical chemistry* 29.11 (1957), págs. 1702-1706.

[14] Henry L Friedman. «Kinetics of thermal degradation of char-forming plastics from thermogravimetry. Application to a phenolic plastic». En: *Journal of polymer science part C: polymer symposia*. Vol. 6. 1. Wiley Online Library. 1964, págs. 183-195.

[15] Ch D Doyle. «Kinetic analysis of thermogravimetric data». En: *Journal of applied polymer science* 5.15 (1961), págs. 285-292.

[16] A. W. Coats y J. P. Redfern. «Kinetic parameters from thermogravimetric data». En: *Nature* 201.4914 (1964), págs. 68-69.

[17] G. I. Senum y R. T. Yang. «Rational approximations of the integral of the Arrhenius function». En: *Journal of thermal analysis* 11.3 (1977), págs. 445-447.

[18] Takeo Ozawa. «A new method of analyzing thermogravimetric data». En: *Bulletin of the chemical society of Japan* 38.11 (1965), págs. 1881-1886.

[19] Joseph H Flynn y Leo A Wall. «A quick, direct method for the determination of activation energy from thermogravimetric data». En: *Journal of Polymer Science Part B: Polymer Letters* 4.5 (1966), págs. 323-328.

[20] Sergey Vyazovkin y David Dollimore. «Linear and Nonlinear Procedures in Isoconversional Computations of the Activation Energy of Nonisothermal Reactions in Solids». En: *Journal of Chemical Information and Computer Sciences* 36.1 (1996), págs. 42-45.

[21] Sergey Vyazovkin. «Evaluation of activation energy of thermally stimulated solid-state reactions under arbitrary variation of temperature». En: *Journal of Computational Chemistry* 18.3 (1997), págs. 393-402.

[22] Sergey Vyazovkin y Charles A. Wight. «Estimating Realistic Confidence Intervals for the Activation Energy Determined from Thermoanalytical Measurements». En: *Analytical Chemistry* 72.14 (2000). PMID: 10939383, págs. 3171-3175.

[23] S. Vyazovkin. «Modification of the Integral Isoconversional Method to Account for Variation in the Activation Energy». En: *Journal of Computational Chemistry* 22.2 (2001), págs. 178-183.

[24] Lérys Granado y Nicolas Sbirrazzuoli. «Isoconversional computations for nonisothermal kinetic predictions». En: *Thermochimica Acta* 697 (2021), pág. 178859.

[25] Kenneth A Lambert y Martin Osborne. *Fundamentals of Python*. Delmar Learning, 2011.

[26] *Documentación de Python*. Ver. 3.10.0. Python Software Foundation. 2021. URL: `https://docs.python.org` (visitado 20-06-2021).

[27] *Documentación de Numpy*. Ver. 1.19.1. Numpy project. URL: `https://numpy.org/doc/` (visitado 25-06-2021).

[28] *Documentación de Scipy*. Ver. 1.6.3. URL: `https://docs.scipy.org/` (visitado 26-06-2021).

[29] *Documentación de Pandas*. Ver. 1.2.2. URL: `https://pandas.pydata.org/docs/` (visitado 28-06-2021).

[30] *Documentación de Matplotlib*. Ver. 3.1.2. Python Software Foundation. URL: `https://matplotlib.org/` (visitado 30-06-2021).

[31] Jake Van der Plas. *Python data science handbook: Essential tools for working with data*. O'Reilly Media, Inc., 2016.

[32] Andy Goldschmidt. *Documentación de Derivative*. Ver. 0.3.1. URL: `https://derivative.readthedocs.io/en/latest/` (visitado 20-06-2021).

[33] Sergey Vyazovkin y Charles A Wight. «Estimating realistic confidence intervals for the activation energy determined from thermoanalytical measurements». En: *Analytical chemistry* 72.14 (2000), págs. 3171-3175.

[34] Bruno Ekawa, Victoria L Stanford y Sergey Vyazovkin. «Isoconversional kinetics of vaporization of nanoconfined liquids». En: *Journal of Molecular Liquids* 327 (2021), pág. 114824.