UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS QUÍMICAS Y FARMACÉUTICAS

Uso de Python para la modelación y determinación de vida útil de matrices alimentarias mediante simulación de Montecarlo

Tesis presentada a la Universidad de Chile para optar al grado de Magíster en Ciencias de los Alimentos por:

Joseph Ananías Huequelef Huechun

Director/a de Tesis: Dr. Roberto Lemus Mondaca

Santiago-CHILE

Enero 2024

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS QUÍMICAS Y FARMACÉUTICAS

INFORME DE APROBACIÓN DE TESIS DE DOCTORADO

Se informa a la Dirección de la Escuela de Postgrado de la Facultad de Ciencias Químicas y Farmacéuticas que la Tesis de Magíster presentada por el candidato

Joseph Ananías Huequelef Huechun

Ha sido aprobada por la Comisión de Evaluadora de Tesis como requisito para optar al grado de Magíster en Ciencias de los Alimentos, en el examen público rendido el día

_____

Director de Tesis:

Dr. Roberto Lemus Mondaca _____

Comisión Evaluadora de Tesis:

Dr. Luis Puente _____

Dr. Jaime Ortiz _____

Dr. Cristian Ramírez _____

Abstract

The modeling and mathematical simulations are helpful tools for predict behaviors in mass transfer process, energy etc. Thank to these two processes, it is possible to reduce time, experiment numbers, and cost. In this work were developed two independent programs, where a program determined the shelf life of a food and other that modeled the dehydration of other foods.

The shelf life of food is defined as the period in which the food retains its developed in python that carried out a probabilistic simulation of Montecarlo for determine the shelf life of dehydrated tomato. The data needed to determine the shelf life include gab isotherms, moisture and another constant which were obtained from the literature. The Montecarlo simulation requires many iterations for to function. In this work it was observed that the greater the number of iterations, the more similar was the value obtained to the literature, with the shelf life estimated by the program being 8.2 month for WVTR value of 1.222 $\frac{g}{day*m^2}$ and with a moisture loss of 10%. The program worked for different numbers of iterations, affecting the response time in the execution of the program. The program was tested up to 2000000 of iterations.

For its part, drying is the dehydration process where there is migration of water from the food to the environment. A program was created to model nonlinear equations by reading a comma separated text file (CSV). This program was able to obtain the coefficients associated with the model, as well as the diffusivity for the case of modeling using fractional Arrhenius together with its graphs.

The mission of the programs was to brings existing knowledge and computer tools closer to the area of foods science, given the great utility and uses that computing has today.

The advantage of having created these programs is that they could be modified or adapted according to the user's needs, such as the modification of the iterations or variables.

The programs created are available online for consultation. The links are found in the annex of this work.

Resumen

El modelamiento y la simulación matemática en el ámbito de los alimentos son herramientas útiles para predecir comportamientos en procesos como transferencia de masa, energía, etc. Gracias a estos dos procesos es posible reducir tiempos, numero de experimentos y costos. En este trabajo se desarrollaron dos programas independientes; uno que estimo la vida útil para un determinado alimento y otro que modelo la deshidratación de otros alimentos.

La vida útil de un alimento se define como el periodo de tiempo en el que este conserva sus características organolépticas y nutricionales que lo mantienen inocuo. Se desarrollo un programa en Python que llevo a cabo una simulación probabilística de Montecarlo para determinar la vida útil de tomate deshidratado. Los datos necesarios para determinar vida útil incluyen isotermas de GAB, humedad, y otras constantes, los cuales fueron obtenidos de la literatura. En este trabajo se observó que a mayor cantidad de iteraciones, más semejante fue el valor obtenido a la literatura, siendo la vida útil estimada por el programa de 8.2 meses para un valor de WVRT de 1.222 $\frac{g}{dia*m^2}$ y con una pérdida de humedad del 10%. El programa funciono para distintos números de iteraciones, eso sí, viéndose afectado el tiempo de respuesta en la ejecución del programa. Él programa fue capaz de trabajar hasta con 5.000.000 de iteraciones.

Por su parte el secado es un proceso de deshidratación donde hay migración de agua desde el alimento hacia el medio. Se creó un programa para modelar ecuaciones no lineales mediante la lectura de un archivo de texto separado por comas (CSV). Este programa fue capaz de obtener los coeficientes asociados al modelo, así como la difusividad para el caso del modelado mediante difusión anómala, y también sus gráficas.

La misión de los programas fue acercar los conocimientos y herramientas informáticas existentes al área de la ciencia de los alimentos, dada la gran utilidad y usos que tiene la computación hoy en día. La ventaja de haber creado estos programas es que se pudieron modificar o adaptar según las necesidades del usuario, como, por ejemplo, la modificación de las iteraciones o variables.

Los programas creados se encuentran disponibles en línea para su consulta. Los enlaces se encuentran en el anexo de este trabajo.

Agradecimientos

Quiero agradecer a Dios por todas las oportunidades y experiencias que me ha permitido tener. Gracias a su ayuda me permito ya culminar con mis estudios universitarios rodeado de hermosas personas. Agradecer a mi pareja, compañera, colega y amiga Fabiola por hacer más amena mi vida universitaria. A mi familia (José, Inés y Daniel) por brindarme todo el apoyo y comprensión y dedicación de manera incondicional desde un comienzo. A mis amigos que hice en estos largos años de estadía universitaria (Maestro, Rogers, Cepeda, Chalo, JP y Thalía). También agradecer a los "pollos" Clarita, Cucho y su cría. A todos ellos, muchas gracias.

Índice

Índice de Figuras

Índice de ecuaciones

Índice de tablas

1. Introducción

1.1. Ciencia de la computación

La computadora es una herramienta para realizar diversos tipos de tareas, desde programar una página web, hasta realizar complejos cómputos que son difíciles de desarrollar de manera analítica. Las computadoras reciben datos de entrada los cuales son procesados, obteniéndose uno o más datos en las salidas (Rosen 1969). Desde el inicio de la computación, han aparecido 4 generaciones de computadoras.

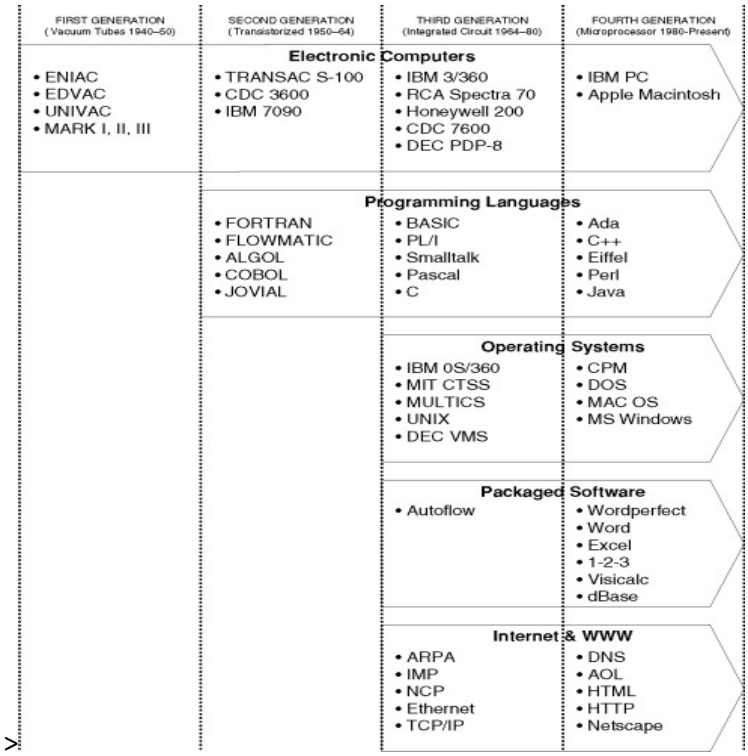


Figura 1. Cronología de las generaciones de computadoras (Rico et al, 2008)

La ciencia de la computación por el contrario de lo que se piensa, no es el estudio de las computadoras o de los componentes de esta. Si bien el computador es importante a la hora de desarrollar la ciencia de la computación, esta no es parte de su estudio.

La principal pregunta de esta ciencia es “¿Qué es lo que puede ser computado?” o “¿qué procesos pueden ser descritos?”. Esta ciencia usa numerosas técnicas de investigación para responder estas preguntas, donde las tres principales son el diseño, el análisis y la experimentación.

La solución a un problema se puede crear paso a paso a través de líneas de un programa (Software), llegando así a un resultado deseado, a esto se le conoce comúnmente como “ALGORITMO” (Zelle, 2004).

1.2. Lenguaje de programación

Para la década de los 70 existían alrededor de 170 lenguajes en los Estados unidos. Actualmente, existen sobre los 8500 lenguajes de programación en todo el mundo.
Los lenguajes de programación aparecieron junto a la segunda generación de computadoras, es decir, cuando las computadoras dejaron de ocupar tubos de vacío y empezaron a ocupar transistores. Entre los lenguajes primigenios que destacan están “la traducción de fórmulas” más conocido como FORTRAN, el “traductor automático de diagrama de flujo” o FLOWMATIC, el lenguaje común orientado a los negocios más conocido como COVOL y el lenguaje de procesamiento de listas LISP (Sammet, 1972).

1.2.1 Lenguaje de programación Python

Dentro de los múltiples lenguajes de programación modernos existentes actualmente (Java, C, C++, RUBY, PEARL, etc.), existe uno que se encuentra en pleno auge, este es el lenguaje de programación Python. Este lenguaje se está utilizando de forma científica debido a su gran capacidad de analizar datos (realizar ciencia de datos), así como su popular uso en el desarrollo de la inteligencia artificial y redes neuronales (Machine Learning y Deep Learning). Python es un lenguaje de alto nivel, conciso y eficiente, el cual evita excesivo uso de caracteres cuando se declara una variable o alguna instrucción. Los bloques en este lenguaje están delimitados por una sangría de espacio en blanco (Zelle, 2004). Por ejemplo, al escribir en Python un mensaje sencillo en pantalla como el clásico “Hola mundo” (“Hello world”) se requieren pocos caracteres en su implantación, (ANEXO 1 Y 2), en cambio al escribir el mismo mensaje en el lenguaje de programación JAVA, el desarrollo del código se hace más extenso y el número de caracteres aumenta. (ANEXO 3).

1.3. Simulación probabilística de Montecarlo

El método de simulación de Montecarlo se desarrolló en la década de los 40 cuando físicos de partículas del laboratorio nacional Los Álamos (Estados Unidos) presentaron problemas en resolver el transporte de partículas subatómicas. El método de Montecarlo es una técnica estadística basada en iteraciones, las cuales reducen el error a un nivel aceptable (disminuyen la incertidumbre), lo que conlleva a que mayores cantidades de iteraciones tomen más tiempo para poder completar la simulación (Frame, 2008).

La base de este método es la generación de números pseudoaleatorios, en donde la solución es determinada por los muestreos aleatorios de las interacciones o las relaciones que describen el fenómeno (Bielajew, 2000; Haghighat, 2015; Johansen, 2010). En este método se agrupan un conjunto de procedimientos para analizar las variables aleatorias usando una simulación con números aleatorios, pero que tienen una distribución uniforme (Frame, 2008; Klenke, 2013). Este método se reconoce como uno que involucra procesos estocásticos e inferencia estadística para desarrollarse. Un proceso de características estocásticas corresponde a una familia de variables aleatorias cuyas características pueden variar a lo largo del tiempo.

El primer gran hito de este método es que no se requieren experimentos físicos para poder llevarlo a cabo, si no que se genera una matriz o colección de datos de forma pseudoaleatoria para desarrollar los cálculos, y lo segundo, es que no es necesario utilizar colecciones de variables aleatorias independientes de distribución conocida siendo este concepto que conduce a una simulación de Montecarlo (Johansen, 2010).

El método de Montecarlo es muy versátil, ya que se puede aplicar a un gran número de disciplinas, desde el área de las ciencias químicas hasta el área económica, tan solo con iteraciones en una computadora. Otro aspecto destacable de este método es que se puede aplicar tanto a procesos estocásticos como a proceso determinísticos (Kroese et al, 2014). Los procesos determinísticos son contrarios a los estocásticos, es decir, se tiene la misma solución para un conjunto de datos iniciales en donde todos los datos se conocen con certeza (Ríos et al, 2004).

1.4. Simulación y modelación matemática

El modelado es el proceso de construcción de un modelo y un modelo es la representación de la construcción y funcionamiento de algún sistema de interés. Uno de los propósitos del modelado es predecir los efectos cuando se cambian las variables del sistema de estudio. Los modelos generados deben ser una aproximación al fenómeno real estudiado y deben contener la mayoría de las variables involucradas, haciendo hincapié en la variable más importante del sistema. El modelo generado tendrá validez cuando se haga una comparativa entre las variables de entrada y el resultado final, en donde este último debe ser coherente al valor esperado por el sistema real estudiado (Anu, 1997).

Los modelos pueden ser de cuatro tipos. Los modelos deterministas consideran variables de entrada y salida con valores fijos. Los modelos estocásticos consideran que al menos una de las variables de entrada o salida sea de naturaleza probabilística. Los modelos estáticos no tienen en cuenta la variable tiempo y por el contrario los modelos dinámicos si tienen en cuenta las variables con respecto al tiempo (Anu, 1997).

Comúnmente los modelos son del tipo dinámico y estocástico. Por otro lado, la simulación es una herramienta para evaluar el desempeño de un sistema propuesto o ya existente, bajo diferentes configuraciones de interés durante largos periodos de tiempo real. Mediante una simulación, se puede determinar la mejor combinación de variables, observar el efecto de las modificaciones al sistema modelado, por nombrar algunas (Anu, 1997).
Otra definición de simulación es la siguiente: Simulación es la aplicación de un modelo con el objetivo de derivar estrategias que ayuden a resolver un problema, o la respuesta a una pregunta perteneciente al sistema (Vasta, 2009).

Las etapas involucradas en el desarrollo y simulación de un modelo o también en el diseño de un experimento de simulación y el posterior análisis son:
Se inicia identificando el problema y enumerando los problemas asociados al sistema. Se seleccionan los limites medidas y diferentes configuraciones del sistema y para que marco de estudio se desarrollara el modelo, luego se recolectan datos reales del sistema así como las variables de entradas asociadas al sistema. El modelo desarrollado debe ejecutarse correctamente comprobando que la salida de resultados se resultados coherentes con el sistema. Los resultados

obtenidos deben comparase con los valores reales del sistema y evaluar su confianza. El modelo debe documentar los detalles de las variables de entrada y de cómo la variación de estas puede generar diferencias significativas repercutiendo en el resultado final. El diseño experimental debe ser adecuado y debe considerar todas las variables involucradas. también se deben considerar las condiciones de inicio de la simulación, determinar si el sistema estacionario o no y el tamaño de la muestra (figura 2). De esta manera se puede iniciar el proceso de simulación, obtener datos y dar recomendaciones a futuro (Anu, 1997).

Figura 2. Etapas del desarrollo de una simulación.

1.5. Vida útil de los alimentos

La vida útil de los alimentos o “SHELF-LIFE” está definida como el periodo de tiempo en que un alimento permanece inocuo, conserva sus características sensoriales, fisicoquímicas, microbiológicas, funciones deseadas y cumple toda declaración nutricional (rótulos, tabla nutricional, etc) o propiedades saludables que se indiquen en el envase, mientras se cumplan las condiciones de almacenamiento recomendadas (Alapont et al, 2020).

1.5.1. Determinación de vida útil de los alimentos

La manera en que un alimento se deteriora y altera puede ser muy compleja, pues se puede dar el caso en el que ocurran varios mecanismos de deterioro (ANEXO 5). Los mecanismos de descomposición de los alimentos se pueden clasificar en mecanismos físicos, mecanismos químicos y/o bioquímicos o mecanismos microbiológicos (patógenos o biota alterante).

Debido a las múltiples causas de deterioro y/o alteración de los alimentos, no existe una metodología única para establecer o validar la vida útil, por lo tanto, la forma de determinación de vida útil de los alimentos es distinta para cada matriz alimentaria (Alapont et al, 2020). Juan Alvarado (2018) estudio la vida útil del dulce de leche midiendo el tamaño de los cristales en el tiempo mediante microscopia óptica y microscopia ocular encontrando que el tamaño de estos cristales crece de forma directamente proporcional al tiempo y que su tamaño puede ser expresado como una ecuación de cinética de primero orden. Crespo (2011) estudio el enranciamiento del maní en el tiempo mediante el cálculo del índice de peróxidos, complementado con cromatografía de gases y pruebas microbiológicas.

La determinación de la vida útil de los alimentos debe estar integrada en los procedimientos basados en HACCP y en las buenas prácticas de higiene (Alapont et al, 2020).

1.5.2. Factores que afectan la vida útil de los alimentos

Existen varios factores que afectan la vida útil de los alimentos, en forma general, se hablan de factores intrínsecos y extrínsecos. Los factores intrínsecos son aquellos factores inherentes al alimento como, su microbiota asociado, PH, conservantes utilizados en su elaboración, sal y su actividad de agua (Jurado y Pacheco, 2019). Dependiendo del macronutriente predominante será la reacción que se lleve a cabo. Si el producto tiene un gran contenido de proteínas, tiene una alta probabilidad de que desarrolle bacterias, si su contenido es alto en grasas, el producto final tiene una alta probabilidad de enranciarse y si su contenido es alto en carbohidratos, se corre el riesgo de desarrollo de hongos y levaduras (Carrillo y Reyes, 2007). Por otra parte, los factores extrínsecos son aquellos que no dependen del alimento. Estos pueden ser, la humedad relativa del ambiente, la temperatura, las condiciones de envasado, etc. Por ejemplo, en un estudio de cosecha de tomates y pimientos (Marvasi y George, 2015) se informó que cosechar estas frutas en su punto máximo de madurez produjo un aumento en el número de colonias de *salmonella*, elevando su promedio de 10

a 100% más alto en comparación con la fruta verde madura, lo cual disminuye su inocuidad y por consiguiente su vida útil.

1.5.3. Vida útil de los alimentos según el RSA

En al Reglamento Sanitario de los Alimentos se encuentran las siguientes definiciones:

- **Fecha o plazo de duración mínimo:** "Es aquella fecha o aquel plazo en que expira el periodo en que el fabricante garantiza que el producto, conservado bajo determinadas condiciones de almacenamiento (si las hubiera), mantiene todas las cualidades significativas que se le atribuyen, tacita o explícitamente, sin que esto signifique que el producto no puede ser comercializado más allá de esta fecha o plazo. El uso de fecha o plazo de duración mínimo es optativo.
Esta fecha o plazo de duración mínimo podrá indicarse en forma de recomendación pudiendo utilizarse la expresión "consumir preferentemente antes de" u otras equivalentes".

- **Fecha de vencimiento o plazo de duración:** "Aquella fecha o aquel plazo en que el fabricante establece que, baja determinadas condiciones de almacenamiento termina el periodo durante el cual el producto conserva los atributos de calidad esperados.
Después de esa fecha o cumplido este plazo el producto no debe ser comercializado.
Para los efectos de utilizar el plazo de duración, se entenderá que este empieza a regir a partir de la fecha de elaboración".

El RSA también establece que: "La fecha de vencimiento o el plazo de duración deberán ser completamente definidos, no aceptándose en estos casos expresiones tales como "consumir preferentemente antes de" u otras equivalentes, que resten precisión o relativicen la fecha de vencimiento o plazo de duración (RSA).

1.5.4. Definición de variables involucradas en la determinación de vida útil

Presión de vapor saturado (Pv): Es la presión a la que la temperatura de fase liquida y vapor se encuentran en equilibrio dinámico. Estos valores se encuentran tabulados para cada temperatura en la literatura.

Permeabilidad: Las características del envase del alimento también influyen en la vida útil. Un envase metálico tiene diferentes propiedades que uno elaborado con polietileno. El parámetro diferenciador de un envase con respecto a otro es su permeabilidad, que es la capacidad de un material para permitir que un fluido atraviese su estructura interna.

Tasa de transmisión de vapor de agua (Water Vapor Transmission Rate) (WVTR): Es la medida de la velocidad con la que el agua se transfiere a través del material de envase o recubrimiento del alimento en una unidad de área y tiempo determinados $\left(\frac{g}{m^2 * dia}\right)$. La industria por lo general busca que este valor sea lo más bajo posible para garantizar la estabilidad del producto.

Gradiente de presión de vapor: Corresponde a la presión de vapor a una temperatura dada, la cual a su vez corresponde a la temperatura a la cual se envaso el producto.

Superficie del envase: Corresponde a la multiplicación de la altura por el ancho del envase. Su unidad está en metros cuadrados (m^2).

B: Corresponde al intercepto de la regresión lineal. Se obtiene de la relación de la isoterma de GAB la cual relaciona la Aw de las sales saturadas con la humedad.

Actividad de agua (Aw): Es la fracción de contenido de agua de un producto que está libre y disponible para el crecimiento de microorganismos y para llevar a cabo diversas reacciones químicas en el alimento que afectan su estabilidad (Oliag, 2017).

1.6. Isotermas de Sorción

La isoterma de sorción de agua relaciona a una temperatura constante el contenido de humedad de equilibrio con Aw del producto en un intervalo de humedad o actividad. El conocimiento de la isoterma tiene aplicaciones en el análisis y diseño de procesos alimentarios, como el secado y el mezclado, por ejemplo, así como también para la determinación de las condiciones de almacenamiento optimas del producto y su predicción de vida útil.

Se han desarrollado numerosas ecuaciones para describir las isotermas, llegando incluso a existir 200 modelos propuestos, los cuales varían en los parámetros utilizados y el rango de Aw en el que estos son adecuados. Los dos modelos más utilizados actualmente son el modelo de BET y el modelo

de GAB, aunque regularmente también se utilizan los modelos propuestos por Caurie o Henderson por nombrar algunos.

1.6.1. Isotermas de BET y GAB

La ecuación de BET relaciona la humedad en el equilibrio (We) con la actividad del agua (Aw) por medio de 2 parámetros Wo y C, en donde Wo corresponde al valor de la mono capa (capa mono molecular de agua adsorbida) y C es una constante característica del material relacionada con el calor desprendido en el proceso de sorción (Ecuación 1)

El modelo de BET presenta ciertas limitaciones, teniendo un buen ajuste para ciertos intervalos de Aw, siendo su rango óptimo de 0 a 0.55. Sin embargo, este modelo se acepta como referencia del contenido de humedad de mayor estabilidad para alimentos secos. Estas limitaciones del modelo de BET hicieron que se generaran más modelos como GAB, SMITH y OSWIN. El modelo de GAB representa de mejor manera los datos experimentales, abarcando el intervalo Aw 0 a 0.95 para la mayoría de los alimentos de interés practico. Las constantes asociadas a GAB son las siguientes, M (valor de monocapa), C (constante característica del producto asociada al calor de adsorción de la monocapa) y K (Constante relacionada al calor de la multicapa) (Ayala, 2011)
Los modelos de GAB y BET están basados en los mismos principios de la monocapa, pero el modelo de GAB agrega un grado de libertad adicional (K). Con esta constante, el modelo de GAB asume que las moléculas en multicapas tienen interacciones con el adsorbente en valores energéticos similares a los que tienen las moléculas en la monocapa (Escobedo et al, 2012).

$$M_e = \frac{M_O * C * A_W}{(1 - A_W) * (1 + (C - 1) * A_W)}$$

Ecuación 1. Isoterma teórica de BET

$$M_e = \frac{M_O * C * K * A_W}{(1 - K * A_W) * (1 + (C - 1) * K * A_W)}$$

Ecuación 2. Isoterma teórica de GAB

Para determinar la isoterma de una matriz alimentaria de peso conocido, así como su humedad inicial (en base húmeda) a una temperatura determinada, el alimento deberá ser posicionado en una cámara hermética, la cual contenga una solución saturada con la sal de interés, por ejemplo, *LICL*, CH_3COOK , *MgCl*$_2$, *K2CO*$_3$, $Mg(NO_3)_2$, *NaBr*, *SrCl*$_2$, *NaCl*, NH_4Cl, $(NH_4)_2SO_4$ etc. En donde cada solución saturada dará un valor de monocapa distinto.

1.6.2 Secado de alimentos

El agua es el componente más abundante en los alimentos, teniendo influencia directa en sus características organolépticas y condiciones de almacenamiento. También es el componente más costoso de eliminar. Desde tiempos remotos en la historia, los entes humanos se percataron que los alimentos que deshidrataban se conservaban por más tiempo. La deshidratación de alimentos puede mejorar la palatabilidad, la digestibilidad, sabor y color además de que se hacen menos susceptibles a los ataques mohos e insectos (Digvir, 2016).

Cuando se utiliza aire caliente para transferir calor a los alimentos, la humedad final queda condicionada por la humedad relativa y la temperatura del aire. Si el alimento se deja en contacto con el aire durante un tiempo indefinido este alcanzara el equilibrio con el aire. Este contenido de humedad se conoce comúnmente como contenido de humedad de equilibrio (EMC) (Digvir, 2016).

Una isoterma de sorción representa la relación entre EMC-ERH (humedad relativa del aire). Los alimentos pueden alcanzar el equilibrio ganando o perdiendo agua. El alimento ganará o perderá agua dependiendo de la presión de vapor. Si la presión de vapor del entorno es más baja que la presión de vapor del alimento ocurrirá un proceso de desorción, por el contrario, si la presión de vapor del entorno es más alta que la presión de vapor del alimento ocurre el proceso de adsorción (Iglesias y Chiriffe, 1982).

A continuación se muestra un diagrama el cual explica las etapas llevadas a cabo en el desarrollo de este trabajo.

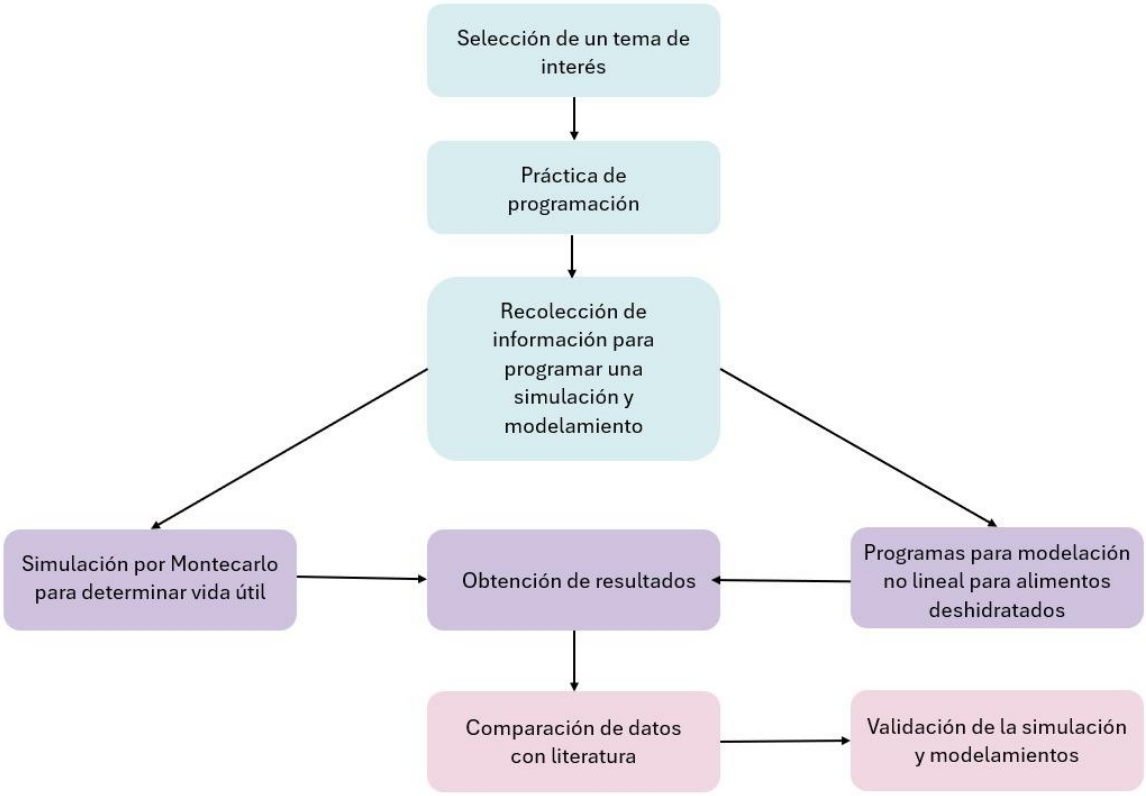


Figura 3. Etapas de desarrollo del trabajo

2. Hipótesis y objetivo

Dados los avances en nuevas técnicas de modelación y el avance en la simulación de procesos, es menester adquirir conocimientos que puedan hacer frente a estos cambios, es por ello, que en este trabajo se darán algunas directrices y se indagara en algunas simulaciones aplicadas en matrices alimentarias (específicamente en el tomate deshidratado), con el fin de que las nuevas generaciones de ingenieros en alimentos tengan noción de una simulación (simulación por Montecarlo) y modelamientos no lineales aplicada en el área de las ciencias los alimentos. A partir de lo anterior, se puede generar la siguiente hipótesis:

2.1 Hipótesis

El desarrollo de algoritmos en el lenguaje Python presenta mayor eficiencia computacional para llevar a cabo simulaciones (método por Montecarlo) y modelaciones (modelos empíricos) del comportamiento de matrices alimentarias durante un procesamiento (secado) y almacenamiento (vida útil) con respecto a métodos tradicionales de generación de datos.

2.2. Objetivo general

Generar códigos computacionales en lenguaje Python para determinar la vida útil del tomate deshidratado mediante una simulación de Montecarlo y otro código que modele curvas de secado no lineales de matrices alimentarias (albaricoque y manzana).

2.3. Objetivos específicos

- Estudiar los conceptos y principios del lenguaje programación en Python
- Desarrollar algoritmos computacionales bajo lenguaje Python
- Aplicar el método por Montecarlo y modelamiento sobre el comportamiento de las matrices alimentarias durante el proceso de vida útil y secado respectivamente.
- Comparar los resultados simulados y/o modelados con respecto a los datos de literatura

3. Materiales

Para realizar este trabajo se utilizo

- Computador HP Omen 15 de procesador Ryzen 7 de la serie 4000 y 16 GB de memoria RAM.
- Modulo Spyder 5.4.1 contenido en la interfaz de Anaconda 3 Navigator.
- Bibliotecas Pandas, Numpy, Matplotlib (colors, cm), Statistics, Math (sqrt, pi) Re, Random, Scipy.optimize, Csv, Lmfit (model, parameters), Os.

4. Metodología de la simulación

4.1. Recopilación de datos experimentales

Escobedo et al (2012) en el artículo "Inclusion of the variability of model parameters on shelf-life estimations for low and intermediate moisture vegetables" determinaron la vida útil de varios vegetales mediante la simulación por Montecarlo, entre ellos se encuentra el tomate deshidratado empacado en polietileno. Para corroborar que la simulación por Montecarlo llevado a cabo en el lenguaje de programación Python fue correcta, se utilizaron los mismos datos usados por los autores, es decir, se tomó los datos publicados por Timmermann et al (2011) para las constantes de la isoterma de GAB para el tomate a 30 °C. Las sales utilizadas para esta isoterma fueron *LICL*, CH_3COOK , *MgCl*$_2$, *K2CO*$_3$, $Mg(NO_3)_2$, *NaBr*, *SrCl*$_2$, *NaCl*, NH_4Cl, $(NH_4)_2SO_4$ (figura 3).

Estos datos son necesarios para la primera parte del programa, parte que se encargara de generar una serie de resultados que distribuyan de manera normal respecto a la media, generando así, una nube de valores de Aw asociados a cada sal. De la literatura también se obtuvieron los valores para la permeabilidad, dimensiones del envase, presión de vapor, y de todas las variables utilizadas en la ecuación de vida útil.

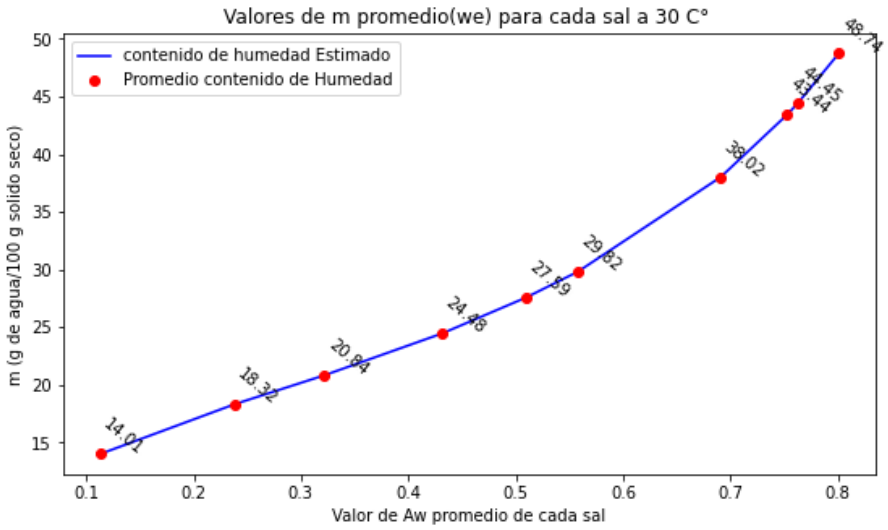


Figura 3. Valores de humedad calculados de forma clásica al reemplazar los parámetros en la ecuación de GAB y la Aw asociada para cada sal de *LICL*, CH_3COOK , *MgCl*$_2$, *K2CO*$_3$, $Mg(NO_3)_2$, *NaBr*, *SrCl*$_2$, *NaCl*, NH_4Cl, $(NH_4)_2SO_4$ en orden ascendente respectivamente.

4.2. Elección del modelo de GAB

El desarrollo del modelo y las ecuaciones utilizadas en Montecarlo se escogen según lo que se desea determinar. El modelo de GAB suele utilizarse por sobre el modelo de BET por su gran rango de Aw lo que permite un buen ajuste en distintas matrices alimentarias. En la literatura se pueden encontrar estudios de determinación vida útil utilizando el modelo GAB para diversos alimentos como harina de yuca (Ayala-Aponte, 2011), pulpa de rosa mosqueta (Demarchi et al, 2013), te negro (Arslan y Tog'rul, 2006), granola (Macedo et al, 2013), comida de perro (Uana Da Silva et al, 2022), uchuva (Vega-Galvez et al, 2014) y nanocompositos de galletas saladas (Karhanis et al, 2021).

4.3. Cálculo de distribución normal inversa en Python

La distribución normal inversa se requiere para la utilización de la simulación de Montecarlo en la ecuación de GAB. La distribución normal inversa se define como el inverso de la distribución normal acumulada para la media y desviación estándar específicas y se utiliza para determinar un valor critico de "Z" correspondiente a una distribución normal en función de una probabilidad (figura 4) (Soporte de Microsoft). Para calcular la distribución normal inversa para cada sal utilizando Python

se requiere el promedio de las constantes M, C y K, sus respectivas desviaciones estándar y una matriz de probabilidades. Esta matriz es generada mediante una secuencia de instrucciones en Python entregando valores pseudoaleatorios comprendidos entre]0,1[.

```python
import scipy.stats as stats

# Valor de probabilidad (entre 0 y 1)
p = 0.95

# Media y desviación estándar de la distribución normal
media = 0
desviacion_estandar = 1

# Calcula la distribución normal inversa
valor_inverso = stats.norm.ppf(p, loc=media, scale=desviacion_estandar)

print("Valor inverso:", valor_inverso)
```

Figura 4. Declaración de la función normal inversa en Python

4.4. Aplicación de iteraciones

Los resultados obtenidos en 4.3 se reemplazaron en la ecuación de isoterma GAB cierta cantidad de veces, donde 1 vez equivale a 1 iteración y donde por cada iteración cambia el valor de la probabilidad. En este trabajo se utilizaron 500 iteraciones pues Escobedo et al (2012) aplico esta misma cantidad de iteraciones en su trabajo. También se llevó a cabo 5000 y 5.000.000 de simulaciones para visualizar si existe cambio en los resultados. La ecuación de GAB entrega una nube de valores de monocapa asociado a cada sal. En la figura 5 se observan las nubes de valores de monocapa y el promedio de estos se denota por un punto rojo.

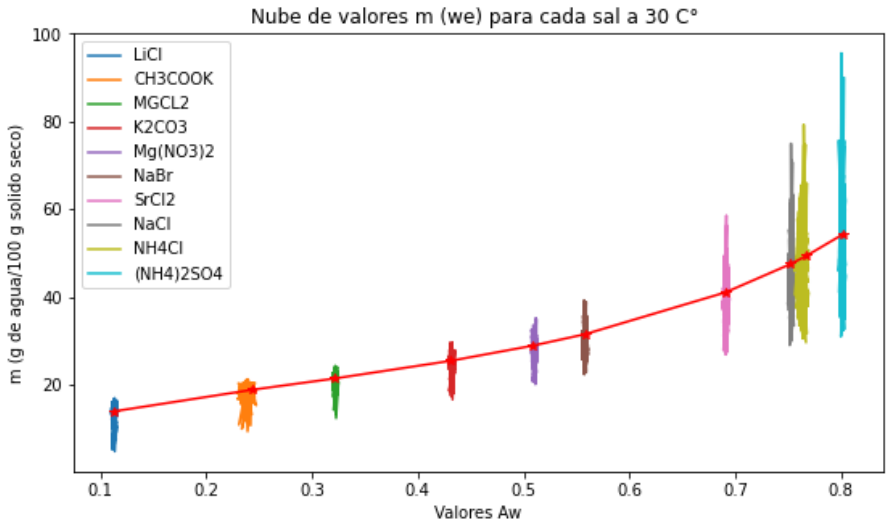


Figura 5. Nube de Valores de We asociados a cada sal calculados mediante 5000 iteraciones de Montecarlo.

4.5. Determinación de vida útil mediante Python

Los datos de monocapa obtenidos junto a los datos de dimensiones del envase, tasa de transmisión de vapor, cantidad de sólidos y presión de vapor en la que se envase se reemplazaron en la ecuación de vida útil para su determinación. Se asumieron las mismas condiciones Escobedo y avellaneda para la perdida de humedad la cual es de un 10%, y la taza de transmisión de vapor de agua (wvtr) igual a 1.222 $\frac{g}{day*m^2}$.

A continuación se describe todo el proceso, desde la recepción de datos iniciales para el comienzo del programa (como numero de variables y numero de simulaciones), así como la incorporación numérica de las variables asociadas en la determinación de vida útil, cuando el programa vaya en la mitad de su ejecución.

4.5.1. Creación de programa de determinación de vida útil en Python

Utilizando Python V 3.1 y la interfaz “Spider V5” de “ANACONDA”, se creó una matriz para recolectar los resultados de las iteraciones de Montecarlo para cada sal, y otra matriz para las probabilidades pseudoaleatorias. Las dimensiones de las matrices quedaron condicionadas por la cantidad de sales a utilizar, junto con las propias constantes de la ecuación de GAB para poder obtener los valores de

la monocapa. Estos parámetros en conjunto (Aw asociadas a las sales y constantes de GAB) corresponden al número de columnas y las iteraciones (número de simulaciones) corresponden al número de filas.

4.5.1.1 Subprograma Inicio

En primer lugar, se debieron especificar la cantidad de filas y columnas. En donde las filas corresponden al número de iteraciones mientras que las columnas al número de variables involucradas en el modelo seleccionado**.** Este Subprograma conto con un filtro para descartar cualquier entrada de datos no numéricos. En caso de que fueran ingresados números decimales, estos se redondearon al menor entero con la función "int" (Figura 6). Estas variables fueron retornadas para ser utilizadas más adelante por subprogramas posteriores.

```
def inicio():
    print ("Bienvenido a la matriz prototipo de montecarlo")

    filas = input(
        "introduce el numero de filas,estas corresponderan a las repeticiones d
    num_format = re.compile(r'^\-?[1-9][0-9]*$')
    it_is = re.match(num_format,filas)
    if it_is:
        print("True")
        filas = int(filas)
    else:
        print
        (" Ha ingresado una opcion incorrecta, por favor ingrese numeros entero
        print()
        return inicio()
```

```
    columnas =(input("introduce el numero de columnas:(variables del experiment
    num_format = re.compile(r'^\-?[1-9][0-9]*$')
    it_is = re.match(num_format,columnas)
    if it_is:
        print("True")
        columnas = int(columnas)
    else:
        print(" Ha ingresado una opcion incorrecta, por favor ingrese numeros e
        print()
        return inicio()

    return filas, columnas
```

Figura 6. Código Primer subprograma llamado "inicio". Este se encarga de recibir las dimensiones de la primera matriz.

4.5.1.2. Subprograma entrada

La finalidad de este subprograma fue el de recibir los datos asociados a la ecuación de isoterma de GAB y los valores de humedad de cada sal saturada. Este algoritmo fue pidiendo el nombre de las variables involucradas luego su promedio y finalmente las desviaciones en ese orden (Figura 7). Las tres primeras columnas de la primera matriz generada correspondieron a los parámetros de la isoterma (Figura 8).

```
def entrada(filas,columnas):
    matriz = []
    casa =[]
    for i in range (3):
        matriz.append([])
        for j in range (columnas):
            if i ==0:
                print("ingrese el nombre de la variable")
                valor = str(input("fila {}- columna {} :".format(i+1,j+1)))
                matriz[i].append (valor)
                casa.append(valor)
            if i==1:
                print("ingrese el promedio de la variable")
                valor = (input("fila {}- columna {} :".format(i+1,j+1)))
                num_format = re.compile(r"[-+]?\d*\.?\d+")
                it_is = re.match(num_format,valor)
                if it_is:
                    print("True")
                    valor = float(valor)
                    matriz[i].append (valor)
```

Figura 7. Parte del código del subprograma entrada

La expresión “matriz = [ ]” indica que se creó una variable llamada “matriz”, pero esta corresponde a una variable de tipo “lista”, la cual está vacía. Por otro lado, una lista de listas corresponde a una matriz. El signo “= “es utilizado como símbolo de asignación, y este le asigna un espacio de memoria a una variable. Si se creara una variable llamada “matriz2” y a esta matriz se le asignara el valor de la “matriz” (matriz_2 = matriz), ambas tendrían la misma dirección de memoria. De esta forma se aprecia la eficiencia del lenguaje que, pudiendo asignar otra dirección de memoria, hace que compartan la misma dirección cuando estas tienen el mismo contenido y/o valor (ANEXO 6).

```
[ Mo C1 K LiCl CH3CooK MgCl2 K2CO3 Mg(NO3)2 NABR SRCL2 NACL NH4CL (NH4)2SO4 ]
[ 16.6 31.4 0.83 0.113 0.238 0.322 0.432 0.51 0.558 0.691 0.753 0.763 0.801 ]
[ 0.6 9.3 0.06 0.001 0.003 0.001 0.001 0.001 0.001 0.001 0.001 0.002 0.001 ]
```

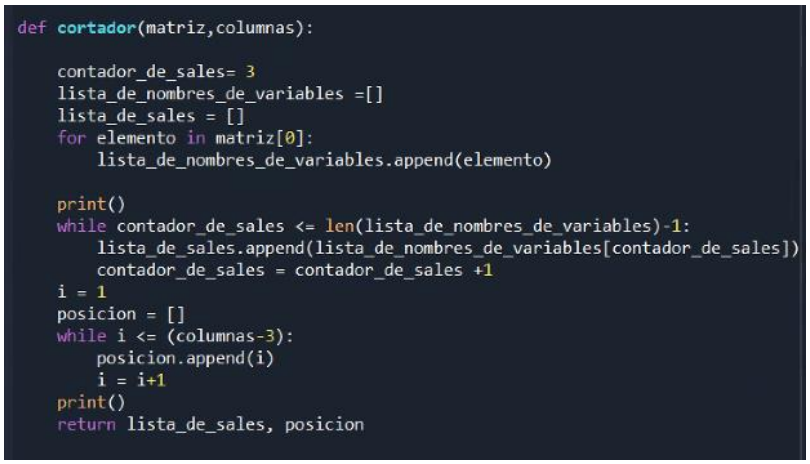
Figura 8. Matriz con datos ya ingresados: nombres de las sales, su promedio y desviación estándar.

4.5.1.3 Subprograma cortador

Este subprograma tuvo como finalidad sacar de la matriz anterior sus 3 primeras columnas, ya que estas tienen los valores de las constantes de la ecuación de GAB, y por eso se encuentra definido el valor de “3” dentro de este subprograma. Este valor puede ser modificado según el tipo de ecuación y la cantidad de variables asociadas. Aquí también fueron definidas dos listas vacías las cuáles serán las encargadas de almacenar los valores de Aw y de las constantes de GAB.

Un ciclo “for-in” fue agregado posteriormente para recorrer tuplas, arreglos (vectores) y /o matrices generadas. A este ciclo también se le puede asociar una operación a medida que las listas o tuplas se recorren (multiplicar por un escalar, sumar o almacenarlas en otra variable), o aplicar condiciones.

La función del ciclo “for- in” en el subprograma cortador, fue de almacenar solo los nombres de las variables (datos de tipo String), los cuales quedaron almacenados en la primera fila de la matriz generada. (Figura 9).



```
def cortador(matriz,columnas):

    contador_de_sales= 3
    lista_de_nombres_de_variables =[]
    lista_de_sales = []
    for elemento in matriz[0]:
        lista_de_nombres_de_variables.append(elemento)

    print()
    while contador_de_sales <= len(lista_de_nombres_de_variables)-1:
        lista_de_sales.append(lista_de_nombres_de_variables[contador_de_sales])
        contador_de_sales = contador_de_sales +1
    i = 1
    posicion = []
    while i <= (columnas-3):
        posicion.append(i)
        i = i+1
    print()
    return lista_de_sales, posicion
```

Figura 9. Separador de String de los datos numéricos. También se separan las 3 primeras columnas que están asociadas a la ecuación de GAB.

Luego del ciclo “for-in” se introdujo otro ciclo, el ciclo “while”, el cual quiere decir “mientras” y se aplicó hasta que las condiciones entregadas a este ciclo dejaron de ser ciertas. Por ejemplo, el “contador” de sales fue definido con el numero “3” por defecto, y la condición del subprograma dice

que si este número es menor o igual a la cantidad de sales ingresadas, ira agregando el nombre de cada sal en la matriz vacía previamente llamada “lista_de_sales”, siempre y cuando la condición sea verdadera. Al final de la ejecución, se le sumo un “1” al contador de sales, tomando el valor de “4” y así sucesivamente hasta que el contador alcanzo el mismo valor que la cantidad de sales. Si las sales ingresadas fueron 10, el contador alcanzo un valor de 10.

En Python, la primera ubicación de una lista está ocupada por la posición de índice 0. El primer elemento de una lista tendrá la posición 0, el segundo elemento la posición 1 y así sucesivamente hasta n-1.
La función “len ()” devuelve un valor numérico, el cual corresponde al tamaño de la lista de elementos (no confundir con la posición). Por ejemplo, el “len” de la siguiente colección de elementos es 6 (Figura 10).

```
elementos = ["alfa", 1, 0, "q", "gallina","!"]
print(len (elementos))
```

```
6
```

Figura 10. Ejemplo de la función len

Al final del programa se creó una matriz llamada “posición” la cual enumero las columnas de cada sal, siguiendo la lógica del ciclo “while” antes mencionado.

4.5.1.4. Subprograma Aleatorio

Este subprograma creo una matriz de probabilidad con datos “pseudoaleatorios” con números comprendidos entre]0,1[(Figura 11) con las dimensiones de filas y columnas definidas en un comienzo. Estos números generados simulan ser las probabilidades de ocurrencias de los resultados. Esta matriz se creó gracias a la función “np.random.rand (filas, columnas)”, función que está contenida en la biblioteca “random”. Al final de la ejecución del programa se retornó la matriz aleatoria llamada “matriz2”. Cuando se habla de retornar un resultado, quiere decir que el resultado

obtenido de las operaciones de un programa queda a disposición de otro programa o subprograma para su uso.

```
def aleatorio(filas,columnas):
    matriz2 = np.random.rand(filas,columnas)
    print (matriz2)
    print()
    print("Esta es la matriz aleatoria")
    return matriz2
```

Figura 11. Creación de una matriz con datos aleatorios comprendidos entre 0 y 1 con las dimensiones de fila y columna ingresadas.

4.5.1.5. Subprograma probabilidad inversa

Fue implementado para el desarrollo estadístico. Su finalidad fue el de calcular la probabilidad normal inversa, ocupando para ello la "matriz" que fue definida en un principio, el número de filas y columnas, y también la "matriz2" creada en el subprograma anterior (Figura 12).

Los valores más altos son aquellos asociados a valores de probabilidad muy altos (cercanos a 1), mientras que los bajos están asociados a valores de probabilidad muy bajos (cercanos a 0).
De esta forma se pudo calcular las distribuciones de probabilidad inversa para las constantes asociadas a la ecuación de GAB y para las humedades asociadas a las sales involucradas en la simulación.

El subprograma recorrió las columnas una a una, pero antes de pasar a la siguiente columna hizo un barrido de todas las filas asociadas a esa columna de la "matriz2" la cual contiene las probabilidades (Figura 12). La instrucción para que recorriera toda la matriz2 por defecto fue "matriz2[i][j]".

Se incorporo una nueva variable nombrada "nuevo", variable que fue el resultado del cálculo de la probabilidad de distribución normal inversa mediante la función "stats.norm.ppf (X, Y, Z)" en donde:
X: Corresponde a la probabilidad (0-1)
Y: Promedio de las variables
Z: Desviación estándar de las variables.

```
def probainversa(matriz,filas,columnas,matriz2):
    l =0

    matriz3 = []
    for j  in range(columnas):
        l =l+1
        for i in range (filas):
            proba = matriz2[i][j]

            nuevo = stats.norm.ppf(proba,matriz[1][l-1],matriz[2][l-1])
            matriz3.append(nuevo)
    return matriz3
    print()
```

Figura 12. Algoritmo de Cálculo de probabilidad normal inversa

Los valores obtenidos fueron a su vez almacenados en una nueva lista nombrada “matriz3” mediante la función “append”.

Si bien la “matriz3” fue nombrada matriz, esta corresponde a una lista vacía que fue capturando valores. Posteriormente se hizo una transformación para que la lista se convirtiera en una matriz. Finalmente se retornó la lista “matriz3”.

A continuación se muestra gráficamente lo expresado anteriormente (Figura 13). Por temas de velocidad de cálculo, no se muestra por pantalla esta parte del programa al usuario, quedando solo a modo de comentario para la eficiencia del cálculo.

```
probabilidad =  0.11991363588815529 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 15.894748819126134
probabilidad =  0.7251402205321101 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.958908248168715
probabilidad =  0.5897698315210685 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.736171763068445
probabilidad =  0.47653536363247084 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.564689355231717
probabilidad =  0.6558151184560174 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.840641041899108
probabilidad =  0.6809390782056857 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.882195835946913
probabilidad =  0.344630425427196 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.360084990153865
probabilidad =  0.15413267854235368 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 15.988678845734766
probabilidad =  0.2904467464869164 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.268751955401267
probabilidad =  0.27733359112565004 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.24553141351265
probabilidad =  0.8252273302549081 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 17.161282928796897
probabilidad =  0.04790132469212993 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 15.600668712879644
probabilidad =  0.7034524287524347 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.92061369737471
probabilidad =  0.3317197713485863 promedio = 16.6 desviacion =  0.6 su probabilidad inversa es: 16.33889841845178
```

Figura 13. Cálculo de probabilidades inversas.

4.5.1.6. Subprograma Almacenaje

En este subprograma se convirtió la lista creada anteriormente en una matriz, donde las filas fueron asociadas a las repeticiones de Montecarlo y las columnas al número de sales involucradas pero con 3 columnas extras (+ 3). Esto porque las primeras 3 filas se asociaron a las constantes propias de la ecuación de GAB (Mo, C, K) definidas por programación en ese orden de aparición.

La “matriz4” fue traspuesta para ser utilizada en futuras operaciones. La nueva matriz traspuesta quedo definida por la variable “tras”. Más abajo, se definió una nueva lista vacía llamada “vacío”, la cual se encargó de almacenar los valores obtenidos mientras se ejecutaba el ciclo while (Figura 14).

El valor de “contador = 3” fue definido de esta manera para dejar fuera 3 filas que corresponden a las constantes de GAB. Es por eso por lo que desde el “contador =3” empieza a guardar los valores. Posteriormente los valores de columna son reemplazados por los valores de fila a medida que aumenta el contador y son almacenados en la lista “vacío”.

```
def almacenaje(matriz3,filas,columnas):
    matriz4 = []
    ñ = 0
    for i in range(columnas):
        matriz4.append([])
        for j in range(filas):
            beta = matriz3[ñ]
            ñ =ñ+1
            matriz4[i].append(beta)
    print()
    print()
    tras = np.transpose(matriz4)
    print(tras)
    print()
    print("esta es la matriz original(la de arriba) pero traspuesta")
    print()
    print("Matriz correspondiente a todos los valores ingresados, en el siguiente paso de eliminaran las constantes

    vacio = []
    contador = 3
    while contador < (len(tras[0])):
        columna = [fila[contador]for fila in tras]
        vacio.append(columna)
        contador = contador+1
    print()
    print()
    print("este es el tamaño de tras",len(tras))
    print()
    return vacio,matriz4
```

Figura 14. Código del subprograma almacenaje

4.5.1.7. GAB

En primer lugar, se ocuparon las funciones de biblioteca para calcular los promedios de cada columna de la matriz4, los cuales corresponden a los valores de Aw asociados a las sales. Estos datos fueron utilizados más adelante por otro subprograma, por lo que se retornaron estos valores.

```
matriz5 = []
for i  in range(len(lo)):
    l =l+1
    for j in range (len(lo[0])):
        sal = lo[i][j]
        mono = (matriz4[0][l-1]*matriz4[1][l-1]*matriz4[2][l-1]*sal)/((1-matriz4[2][l-1]*sal)*((1-matriz4[2][l-1]*
        matriz5.append(mono)
print("Aqui esta la matriz 5 la cual son las iteraciones con la ecuacion de Gab")

print()
return matriz5,lo,promediow
```

Figura 15. Cálculo del valor de la monocapa

En esta parte del subprograma se utilizó la ecuación de GAB junto a los valores de sus constantes y humedades asociadas a las sales obtenidas mediante simulación por Montecarlo. Estos valores se iteraron de modo que cada uno de los valores entraron a la ecuación obteniéndose un valor nuevo para cada uno. Los valores obtenidos correspondieron a los valores de humedad en g de agua/ 100 g de solido seco (valor de la monocapa) (Figura 15).

La matriz obtenida en el subprograma anterior se transpuso nuevamente por una comodidad matemática, de esta forma fue más sencillo ingresar los valores en la ecuación de GAB junto a sus constantes y los valores de Aw asociados a las sales obtenidos por la simulación.

Los nuevos datos obtenidos fueron almacenados en la "matriz5" que es una lista vacía. La variable "mono", que corresponde al valor de la monocapa de GAB, se definió mediante su ecuación al lado derecho de la asignación (=).

La matriz "lo" fue recorrida en forma completa, partiendo de la posición lo [0][0] hasta llegar a la posición lo[n][m]. Mientras que para la matriz4, solo se recorrieron las 3 primeras filas de la matriz, las cuales contenían los valores de las constantes "M", la constante "C" y la constante "K".
El programa ingreso los valores de las sales de la primera fila y una vez que terminó, el contador sumo un "1", para que este se desplace una fila más abajo y así repetir el ciclo completo hasta que no quedo ninguna fila sin recorrer. Finalmente se retornó la "matriz5", "lo", y" promediow".

4.5.1.8. Subprograma conversor

Este subprograma necesito dos argumentos para poder inicializar y estos fueron "matriz5" y "lo", esto se ve reflejado en que están dentro del paréntesis en donde se define la función (figura 16)". El resultado obtenido por este subprograma fue una matriz. Finalmente se retorna el resultado de las operaciones como "matriz6" (ANEXO 7).

```
def conversor(matriz5,lo):
    alfa = (len(lo[0]))

    matriz6 = []
    ñu = 0
    print("aqui se almacenan los datos")
    for i in range(len(lo)):
        matriz6.append([])
        for j in range(alfa):
            beta = matriz5[ñu]
            ñu =ñu+1
            matriz6[i].append(beta)
    print()

    print()

    print("matrices con humedades calculadas  g de agua/ 100 g de solido seco")
    print()

    print()
    return matriz6
```

Figura 16. Código subprograma conversor

4.5.1.9. Subprograma estadística

Aquí se calcularon valores estadísticos típicos como el promedio y desviación de cada columna, así como sus valores máximos y mínimos. El promedio se dividió por “100” (siguiendo la metodología ocupada por Escobedo et al, 2012) para obtener la proporción de g de agua /g de solido seco. Ambos valores de promedio se retornaron junto a la matriz6 nuevamente (Figura 17) (ANEXO 8)**.**

```
def estadistica(matriz6,promediow):
    maximos= np.amax(matriz6, axis = 0)
    print()
    print("los valores maximos de cada sal son:", maximos)
    print()
    minimos = np.amin(matriz6, axis = 0)
    print("los valores minimos de cada sal son:",minimos)
    print()
    promedio = np.mean(matriz6,axis = 0)
    print()
    promedio2 = (promedio*0.01)
    print()
    desviacion = np.std(matriz6,axis = 0)

    print()

    print()
    print()
    print()
    return matriz6,promedio,promedio2
```

Figura 17. Imagen del subprograma estadística, el cual calcula los valores mínimos, máximos y sus promedios.

4.5.1.10. Subprograma regresión

Se encargo de determinar la pendiente y el intercepto al realizar una regresión lineal sobre el conjunto de datos promedio generados. El eje "X" está asociado a la actividad del agua (Aw) y el eje "Y" a los gramos de agua/ g de solido seco.

La programación del código que ejecuto la regresión se basó netamente en la definición matemática, pero llevada a lenguaje de programación.

Luego de la ejecución completa de este subprograma no se retorna nada, ya que su única función es mostrar por pantalla los datos para llevar a cabo una nueva regresión lineal con los datos de interés ordenados en forma de pares ordenados. El algoritmo también mostro por pantalla el grafico de regresión lineal con todas las sales involucradas.

En el apéndice de este trabajo se puede consultar sobre otras variables involucradas en la ejecución del programa.

$$\hat{B}_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\ \bar{y}}{\sum_{i=1}^{n}(x_i)^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2}$$

$$\widehat{\boldsymbol{B}}_{\boldsymbol{0}} = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x_i}^{\boldsymbol{2}} * \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{y_i} - \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x_i y_i} * \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x_i}}{\boldsymbol{n}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x^2}_{\boldsymbol{i}} - (\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x_i})^{\boldsymbol{2}}} = \overline{\boldsymbol{y}} - \widehat{\boldsymbol{B}}_{\boldsymbol{1}}\overline{\boldsymbol{x}}$$

Ecuación 3: definición matemática de pendiente e intercepto de una regresión lineal.

4.5.1.11 Subprogramas variable x () y variable y ()

Son subprogramas que tienen como finalidad almacenar los primeros valores de "X" e "Y", aceptando solo valores numéricos. Estos se reiniciaron cuando que se le entregaron datos no numéricos. (Figura 18).

```python
def variablex():
    h = []
    a = input("ingrese el valor de la coordenada  X: ")
    num_format = re.compile(r'[0-9.]+$')
    it_is = re.match(num_format,a)
    if it_is:
        print("True")
        a = float(a)
        h.append(a)
        return h
```

```python
def variabley():
    l = []
    b = input("ingrese el valor de la coordenada Y: ")
    num_format = re.compile(r'[0-9.]+$')
    it_is = re.match(num_format,b)
    if it_is:
        print("True")
        b = float(b)
        l.append(b)
        return l
```

Figura 18. Imagen de subprogramas “variables” encargados de recibir pares ordenados.

El plus de este subprograma fue que el usuario pudo ingresar los datos que estimo conveniente, para analizar el proceso de deshidratación con las humedades de monocapa asociadas a Aw, pudiendo elegir sales que están asociadas a una baja actividad de agua (< 0.3) o sales asociadas a una alta actividad de agua (> 0.7) (ANEXO 10).

4.5.1.12. Subprograma Regresión lineal 2

Este algoritmo tuvo como finalidad realizar una nueva regresión lineal con los datos seleccionados por el usuario. Los ajustes de los datos fueron presentados en pantalla mediante la biblioteca “MATPLOTLYB”. Esta biblioteca se encarga de generar gráficos, así como de personalizarlos con los atributos definidos por el programador.

Dentro del programa “regresión lineal 2” se definieron los parámetros que tuvo que ingresar posteriormente el usuario, como la Aw inicial del producto, la Aw de equilibrio, la pérdida de humedad del producto, su valor de em, su WVTR, la gradiente de vapor y las dimensiones del envase y la presión de vapor asociada al almacenamiento para posteriormente incorporarlas en la ecuación de vida útil (ecuación 7).

4.5.1.12.1. Cálculo de humedad, crítica inicial y de equilibrio

La humedad inicial del alimento debió ser transformada para poder ser parte de la ecuación. La ecuación lineal se desprende de la regresión entre la Aw (eje x) y los valores de la monocapa (eje y) del programa "regresión lineal 2", por lo tanto al reemplazar la AW inicial en la ecuación se obtuvo la humedad inicial:

$$\boldsymbol{Mi = Aw * pendiente + intercepto}$$

Ecuación 4

Donde:

Aw: Es la actividad de agua inicial con la que cuenta el producto

Mi: Es la humedad inicial transformada con los datos de la regresión lineal.

Perdida de humedad: corresponde a la cantidad de humedad que pierde el alimento. Para este trabajo corresponde a una pérdida del 10%, es decir 0.1 siguiendo la metodología de Escobedo et al 2012.

Humedad critica (Mc): corresponde a la relación entre la perdida de humedad y la humedad inicial.

$$\boldsymbol{Mc = \left(1 - perdida\ de\ humedad\ (\%)\right) * Mi}$$

Ecuación 5

Humedad de equilibrio (Me): El valor de la Aw de equilibrio del producto fue reemplazado directamente en la ecuación de la recta para pasar de actividad de agua a humedad.

$$\boldsymbol{Me = Aw_{equi} * Pendiente + intercepto}$$

Ecuación 6

Ecuación de vida útil: La pérdida de vida útil está asociada a la disminución de "Aw" a valores por debajo de la monocapa. El valor de la monocapa puede ser asociado como el contenido de humedad critica el cual este asociado al valor critico de Aw. Al reemplazar todas las variables en la ecuación (7), se obtiene el tiempo de vida útil de la matriz alimentaria. El Genesis de esta ecuación es mediante la resolución de ecuación de difusividad de Fick en estado no estacionario sujetas a condiciones iniciales y de frontera (Labuza, Altunakar, 2007).

$$ln\left[\frac{m_i - m_e}{m - m_e}\right] = \frac{KAP_0}{XW_S b}t$$

Ecuación 7 Ecuación de vida útil

Luego de la aplicación del método de Montecarlo, se obtiene un único valor, es decir, todas las iteraciones junto con los datos ingresados por el usuario convergieron a un solo resultado, esto es debido a que solo se generó una regresión lineal con los datos promedio de las humedades asociadas a las sales. Si bien se aplicó Montecarlo, este se puede volver a utilizar para generar mayor variabilidad en los resultados y así aprovechar el comportamiento de la distribución normal.

4.5.1.13 Subprograma Regresiones Múltiples

Este subprograma realiza una simulación por Montecarlo con las Aw de interés del usuario. A diferencia de las otras regresiones que solo realizaron una regresión, este subprograma realizo "n" regresiones, donde "n" es igual a las simulaciones ingresadas en un comienzo al programa. Las nuevas pendientes e interceptos fueron reemplazados nuevamente en la ecuación de vida útil para obtener una distribución normal de resultados de días de duración del producto. Además, el programa da a elegir al usuario si quiere tomar más datos o menos datos de sales para hacer el nuevo computo. Como los datos ya están ingresados, no es necesario pedir nuevamente los valores (ANEXO 11). Con seleccionar el número asociado a la "Aw" en la columna identificadora, el programa entiende que se quiere hacer una nueva simulación desde ese número hacia abajo, por ejemplo, si se selecciona el número 7, el programa comenzara a realizar simulaciones desde la sal "$SrCl_2$" (sal que está en la posición 7) hasta la sal "$NH4_2SO4$".

4.5.1.14 Subprograma Ordenar

El programa por medio de este subprograma dio a elegir el intervalo de días que se quiere utilizar para el conteo de resultado, y mostrar por medio de una imagen los valores acumulados en ese intervalo , es decir, si el usuario ingresa un numero "10", el programa contara los resultados en

intervalo de 10 días, agrupando los resultados de [0-10[días y luego que termine de contar los resultados que coincidan con ese intervalo pasara al siguiente [10-20[y así sucesivamente hasta llegar a los intervalos superiores obtenidos por la simulación (ANEXO 12). Su funcionamiento se basa en encontrar dentro de la lista de resultados los valores mínimos y máximos y de los valores acumulados de frecuencia para cada intervalo (figura 21).

El programa también entrego un archivo "CSV" (archivo de texto separado por comas). Este archivo se creó en el directorio en donde se ejecutó el programa. El archivo generado debe modificarse en el panel de control de Excel para poder visualizar los datos de mejor manera (ANEXO 13).

5. Resultados y Discusión de la Simulación

El programa luego de su primera etapa de ejecución, entrego una gráfica de las AW asociadas a las todas las sales involucradas y los valores de monocapa correspondientes al reemplazo en la ecuación de GAB. También se mostró por pantalla los datos en forma de pares (Figura 19) (ANEXO 14).

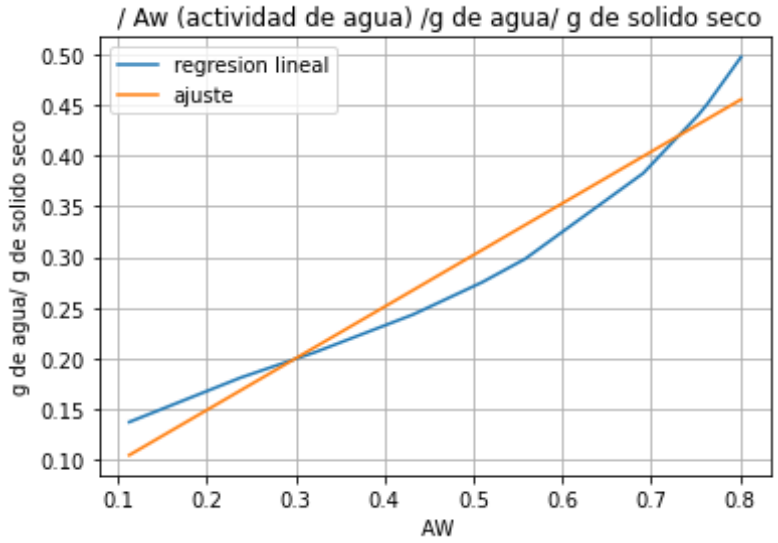


Figura 19. Gráfico de regresión lineal entregado por el programa en pantalla considerando todos los puntos de Aw asociados a cada sal, junto con su correspondiente ajuste.

Al realizar la regresión con todas las Aw correspondientes a cada sal, se observó un buen ajuste, teniendo este un R^2 de 0.9556 (ANEXO 15). Si bien los datos ajustan bien, se debieron tomar puntos

que ajusten de mejor forma al modelo lineal de la ecuación, eligiendo así las ultimas 4 sales (Tabla 1).

Identificador	Aw	g de agua/g sólido seco
7	**0.6909794369855028**	**0.38183128832533186**
8	**0.7530499690389322**	**0.4384636712657604**
9	**0.7629334379275952**	**0.4490478360754946**
10	**0.8010186170584891**	**0.4950674500648104**

Tabla 1. Datos de las 4 sales que presentaron una mayor Aw asociada y correspondiente relación con la humedad de equilibrio. A partir de estos datos se realiza una nueva regresión.

La nueva regresión lineal presento un mejor ajuste, debido a que se seleccionaron datos semejantes entre ellos. Los datos de interés seleccionados de la isoterma tuvieron un valor mínimo de 0.69 y un valor máximo de 0.8 de Aw (Figura 20).

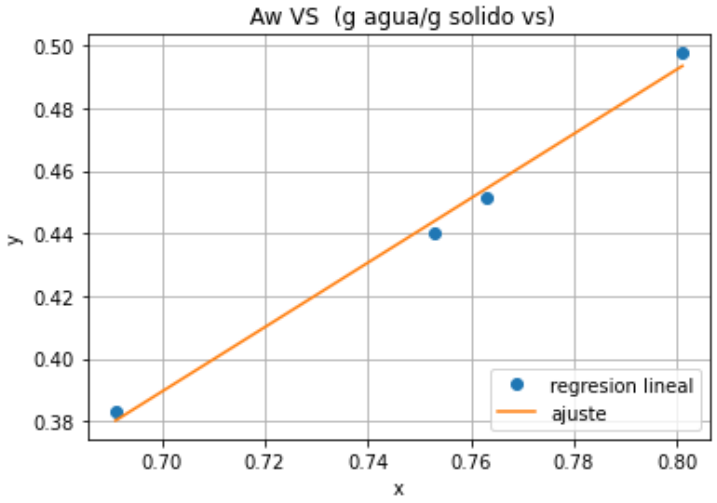


Figura 20. Regresión lineal utilizando las sales que presentan mayor migración de agua (Aw > 0.69). $SrCl_2$, $NaCl$, NH_4Cl, $(NH_4)_2SO_4$ ordenados de forma ascendente.

El R^2 de la nueva regresión es de 0.9926 lo que quiere decir que un 99,26 de los datos fueron explicados por el modelo lineal (ANEXO 16).

Una vez que fueron reemplazados los datos en la ecuación de vida útil, se mostró por pantalla la duración aproximada del producto, siendo esta de 8.1 meses, considerando que los meses tienen

una duración promedio de 30 días. Este resultado está en concordancia con los resultados expuestos por Escobedo et al (2012). (ANEXO 17).

Este valor corresponde a un valor determinista, es decir, todas las iteraciones convergen finalmente a un solo valor, cuando los datos promedio son reemplazados en la ecuación de vida útil. Este valor está asociado a solo una regresión lineal, por lo tanto, hubo que generar una mayor variabilidad de resultados finales. Para esto, se llevó a cabo una nueva simulación con aquellos puntos que mejor son explicados por el modelo lineal y se realizó una regresión para cada simulación de Montecarlo, es decir, se hizo un total de "n" regresiones lineales y evaluaciones, con el fin de obtener "n" resultados de duración del alimento en meses.

Las pendientes y los interceptos de todas las iteraciones fueron reemplazados en la ecuación de vida de útil con el fin de obtener mayor variabilidad (Figura 21). Una vez ordenados en intervalos de ancho seleccionado por el usuario, se pudo seleccionar un intervalo de confianza para obtener un resultado con mayor probabilidad de acierto. Como apoyo estadístico se puede utilizar el archivo CSV ya que a un número elevado de simulaciones ocurre solapamiento de las etiquetas de los gráficos. El intervalo de confianza fue elegido por defecto y no por exceso siguiendo la metodología de Escobedo et al, ya que es mejor que se acorte el periodo de vida útil a que se alargue, para de esta forma evitar una descomposición por exceso de confianza, evitando así problemas legales y de nutrición.

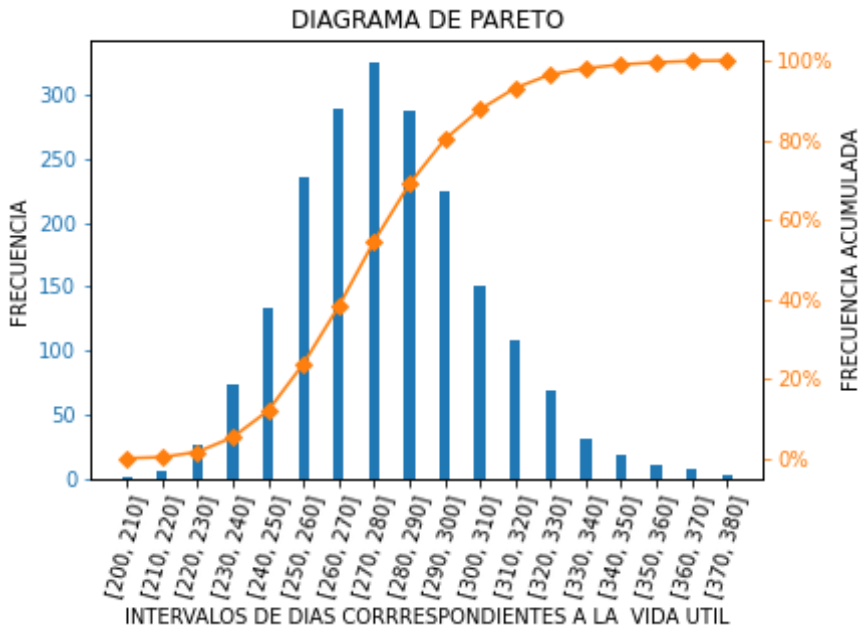


Figura 21. Distribución de frecuencias para 2000 iteraciones.

A mayor número de repeticiones mayor fue el rango (diferencia entre el valor menor y mayor valor), esto se debió a que aparece una mayor cantidad de combinaciones posibles y con ello nuevos resultados. A modo de ejemplo, en las imágenes se muestra el resultado del grafico para 2000 (Figura 21) y 5000000 simulaciones (ANEXO 18).

también trazando una línea perpendicular a las frecuencias desde la frecuencia acumuladas, se puede seleccionar un intervalo de confianza que se estime conveniente. Este tipo de metodología fue utilizado por Escobedo et al (2012). Esta metodología no fue considerada porque los autores consideraron el primer intervalo vacío y en este trabajo se consideró los días efectivamente generados y mayores a cero.

5.1. Problemas asociados a la simulación

5.1.1. Indeterminaciones

Si bien el programa pudo generar millones de iteraciones, no todas las simulaciones entregaron un resultado acorde a la experiencia evaluada, esto debido a las probabilidades y combinaciones que tomaron las simulaciones, siendo estas con tendencias muy extremas, por ejemplo, probabilidades que son casi 1 (0.999994567) y algunas que tienden a 0 (0.0000000001). Estos valores al ser

reemplazados en la ecuación de vida útil hicieron “caer” al programa (detuvo su funcionamiento) no terminando el cálculo. Esto se debió a que la ecuación de vida útil tiene incorporada una función logaritmo natural, y esta no tiene como dominio a todos lo reales, sino que solo a los reales positivos mayores a 0.

Para solucionar esto, se definió como una variable al argumento del logaritmo, variable que será testeada por una secuencia lógica antes de ser evaluada en la función logaritmo. Si la variable “argumento” es menor o igual a cero, el valor del argumento será reemplazado por un número muy pequeño cercano a cero. En caso de que sea mayor a cero, sigue su secuencia normalmente. Se hizo un análisis de resultados y se obtuvo que 72 datos pasaron por esta conversión, representando solo el 0.0036% del total de datos (para 2000000 de iteraciones).

5.1.2. Posibilidad de días negativos

Esto se produjo por la combinación de probabilidades que hicieron que se produzcan argumentos del logaritmo (relación entre Mi, Me y Mc) dentro del intervalo]0-1[, entregando así al final del cálculo, días negativos. Estos días no fueron relevantes ya que llegaron a representar solo el 0.004% del total de resultados (considerando 2000000 de casos). Para efectos de este trabajo, el grafico de frecuencias solo considero los días positivos, y es por eso, que al realizar el conteo de datos generados en el archivo CSV, no hacen el igual a las simulaciones ingresadas por el usuario, así como la sumatoria de la frecuencia relativa acumulada no fue igual a 1 (este dio 0.9999215). También, al aumentar el número de simulaciones, se generan días que se alejan demasiado de la media que fueron considerados como “Datos anómalos” o “outliers”.

5.1.3 Economización de recursos

Si bien parece que las celdas en Excel son infinitas, en realidad no lo son, existe un límite y este es de 1048576 filas y 16384 columnas (Soporte Microsoft, 2023). Como la cantidad de datos generados es enorme, se necesita que estos datos estén ordenados de forma matricial para poder facilitar el computo. Una gran cantidad de iteraciones es útil cuando se tiene un sistema con mucha incertidumbre, reduciendo estas a niveles aceptables. Este no fue el caso, pero de igual forma se hizo funcionar el programa con hasta 5 millones de simulaciones, tardando aproximadamente 7 horas en ejecutarse. Esto dependerá obviamente del sistema en donde se ejecute el programa.

La ventaja de programar la solución y no realizarla en Excel, es la economización de recursos del sistema, ya que, al llevar a cabo una iteración tan grande, gran parte del hardware del sistema es

consumido por las interfaces gráficas, es decir, los menús y todas las opciones que muestra Excel cuando se está ejecutando.

5.3 Aw, contenido de humedad y vida útil

Una isoterma de sorción de humedad es la representación de la compleja relación que existe entre el contenido de humedad y actividad de agua de un alimento. La curva de isoterma muestra la variación del contenido de humedad a medida que la Aw aumenta o disminuye, y de esta forma comprender y controlar por ejemplo, la formulación de productos, su estabilidad y almacenamiento (Simatos, 2002).

Por otro lado, la Aw se define como la medición del estado energético del agua sobre un sistema, la cual se define como la presión de vapor de agua de una muestra dividida por la presión de vapor de agua pura a la misma temperatura (Roudaut; Debeaufort, 2011).

En la gráfica de isoterma de desorción de un alimento se observa la relación directa entre la Aw y el contenido de humedad, es decir, si disminuye su Aw también lo hará su contenido de humedad (figura 3).

La migración de agua (perdida o ganancia de humedad) afecta al alimento produciendo ablandamiento, endurecimiento, y un deterioro de la calidad organoléptica en general, comprometiendo su estabilidad y limitando así su vida útil. (Roudaut; Debeaufort, 2011).

Se sabe que el agua es importante en la estabilidad y calidad de los alimentos, debido a que esta interactúa con otros componentes de la matriz alimentaria pudiendo alterarla. La Aw es un factor determinante para el crecimiento de los microrganismos y para llevar a cabo reacciones de descomposición, ya sean estas químicas, físicas o enzimáticas (Maltini; Torreggiani et al, 2003).

Un estudio sobre fideos fresco demostró que la vida útil del producto aumenta a medida que se disminuye la actividad Aw gracias a la ayuda de reductores de Aw, logrando que el alimento dudara hasta 7 veces más (Man Li; Kexue Zhu et al, 2011).

En el programa de simulación, se probaron distintos valores, tanto de perdida como de ganancia de humedad, entregando los siguientes resultados promedio:

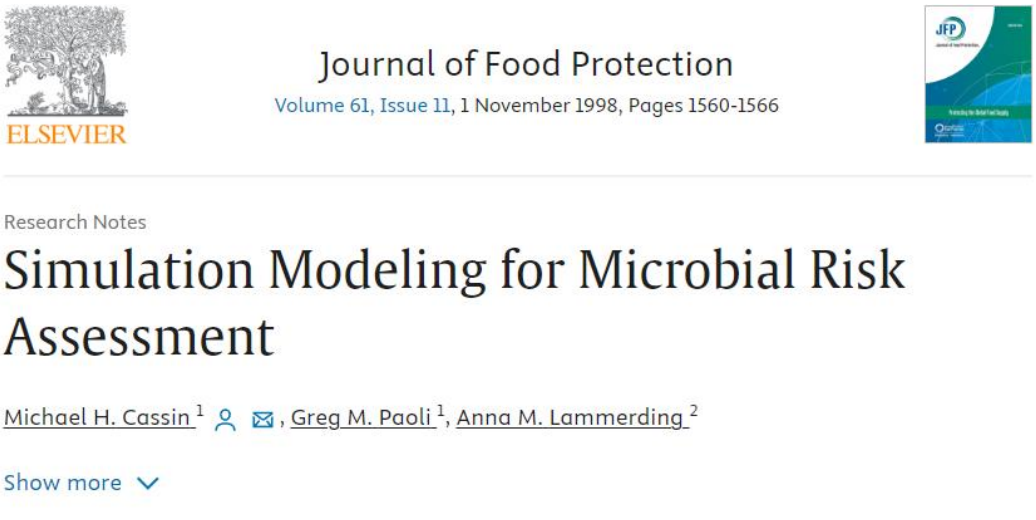
Humedad	Perdida 20%	Perdida 5%
Vida útil (Días)	**644**	**110**

Tabla 2. Días de vida útil promedio generados por una mayor y menor perdida de humedad respectivamente.

Por lo que se está en consistencia con los autores. Esto se debe a disminuye y aumenta la humedad critica respectivamente.

5.4 Aplicaciones de Montecarlo

Se ha aplicado la simulación de Montecarlo en los siguientes trabajos en el área de ciencias de los alimentos.



ELSEVIER

Journal of Food Protection

Volume 61, Issue 11, 1 November 1998, Pages 1560-1566

Research Notes

Simulation Modeling for Microbial Risk Assessment

Michael H. Cassin [1], Greg M. Paoli [1], Anna M. Lammerding [2]

Show more

Journal of Food Engineering
Volume 102, Issue 3, February 2011, Pages 247-256

Assessment of the uncertainty in thermal food processing decisions based on microbial safety objectives

Nattaporn Chotyakul [a], Gonzalo Velazquez [b], J. Antonio Torres [a]

LWT
Volume 47, Issue 2, July 2012, Pages 364-370

Inclusion of the variability of model parameters on shelf-life estimations for low and intermediate moisture vegetables

Zamantha Escobedo-Avellaneda [a], Gonzalo Velazquez [b], J. Antonio Torres [c], Jorge Welti-Chanes [a]

6. Metodología modelamiento de datos

Como se mencionó anteriormente, Python es una poderosa herramienta para el análisis y tratamiento de datos y gratuito. Como ejemplo número 2, se muestran los coeficientes obtenidos de regresiones no lineales. Para esto, se creó un programa que fue capaz de leer archivos “CSV” y de procesar sus datos según el modelo de interés previamente definido. El archivo CSV tiene en su interior la humedad en los diferentes instantes de tiempo para un proceso de deshidratación.
Se utilizaron las bibliotecas “Scipy.optimize” para importar la función “curve_fit”, y también la biblioteca “Lmfit” para importar sus funciones “model” y “parameters”.
El programa de modelación leyó los archivos en formato “CSV” e importo sus parámetros mediante las etiquetas de las columnas.

Se definió una función auxiliar con la función a modelar. Esta función hizo la comparación con los datos de entrada v/s datos de salida mediante la variación de parámetros. Esta función volvió a ser utilizada más adelante para comparar la ecuación modelada en relación con los datos de los experimentales originales.

Posterior a esto, se definió un modelo mediante la función “Model”, función que necesito como parámetro la ecuación previamente definida, la o las variables independientes, y los nombres de los parámetros definidos como Strings. Luego de esto, se declara la función “Parameters” asociándola a una variable para posteriormente declarar los parámetros junto a su valor inicial y sus posibles valores mínimos y máximos. Estos parámetros son definidos por el programador y debieron ser cambiados manualmente en caso de un mal ajuste luego de aplicar la función “model_fit”. El ajuste es informado por pantalla y los parámetros se deben ir cambiando de forma tal que el “R” ajuste de mejor forma, así como también se obtenga un bajo porcentaje de error. En caso de que él “%” de error no llegue al 0%, este debe ser lo más bajo posible.
Finalmente se hizo una gráfica con los de los valores reales y los valores modelados, reemplazando los parámetros por los obtenidos por el ajuste.

```
def fraccional():
    datos = pd.read_csv("datos.csv")
    x= (datos.tiempo.values.reshape(-1,1))*60
    y = np.log(datos.hr.values.reshape(-1,1)*(9.869604401/8))
    #print(x)
    #print(y)
    model = Model(ecuacion_2,independent_vars=['x'], param_names=["difusividad2", "alfa"])

    params = Parameters()
    params.add("difusividad2", value= 0.0000000000001,min =0,max = 0.0001)
    params.add("alfa", value=0.00000000000001,min=1,max = 2.6905)

    result = model.fit(data=y, params=params, x=x)

    print(result.fit_report())
```

Figura 22. Parámetros mínimos y máximos definidos en el ajuste por Covarianza en el modelo ononal.

6.1. Resultados y discusión modelamiento

6.1.1 Modelo Midilli-Kucuk

Se tomaron datos de deshidratación de “SAC APRICOT” a 40 C°, proporcionados por el director de tesis para comprobar el modelamiento, obteniéndose el siguiente resultado.

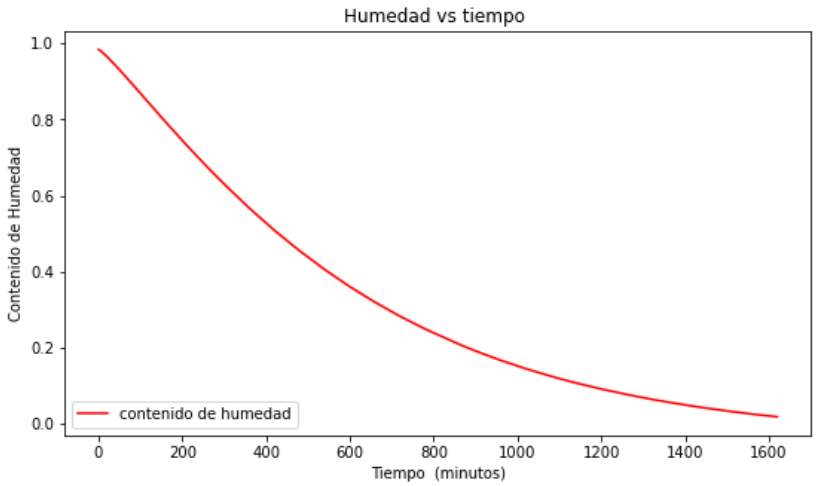


Figura 23. Grafica de contenido de humedad del albaricoque real a medida pasa el tiempo en un proceso de deshidratación por aire caliente.

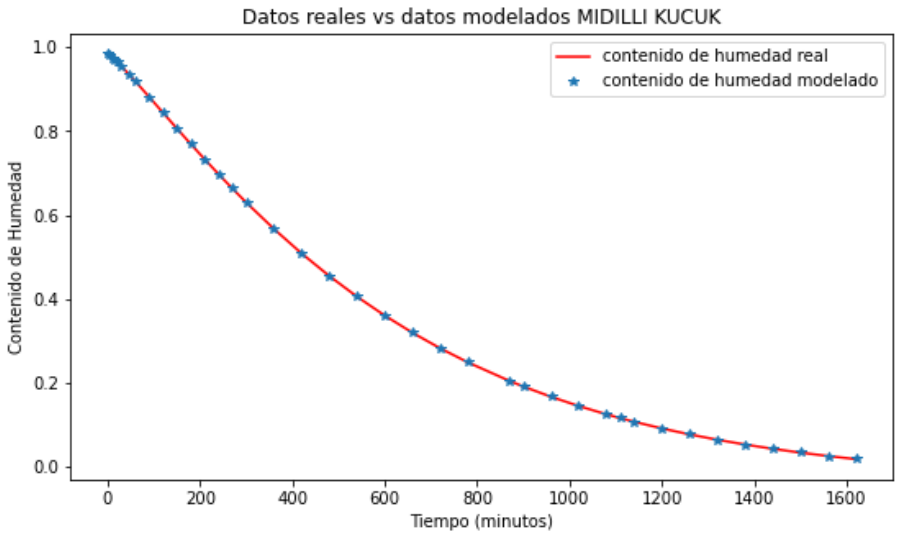


Figura 24. Comparación de datos reales vs datos modelados mediante una regresión no lineal por medio de Python.

El "R^2" de los datos modelados es igual a 1, es decir, los datos ajustaron perfectamente. Los coeficientes encontrados por el programa coinciden a los presentados en la literatura (ANEXO 19).

6.1.2. Caso 2 difusión anómala

De forma análoga se llevó a cabo un modelamiento de tipo de fraccional (difusión anómala) para la deshidratación de manzana mediante ventana de refractancia a 55 °C. Se tomaron estos datos debido a que Hernández (2017) realizo este tipo de modelamiento, y esta manera se pudo corroborar el modelado mediante Python.

Este fue un caso en especial difícil ya que esta fue la única forma en que se pudo modelar fue por minimización de mínimos cuadrados. El ajuste mediante covarianza (otra alternativa de Python) y Solver no pudieron con este tipo de modelado. Lmfit fue la única biblioteca en poder realizar un buen ajuste (ANEXO 20).

El ajuste fraccional estuvo dentro de los parámetros informados por Hernández [25], tanto para la difusividad así como al orden del exponente (ANEXO 21). Los datos modelados correspondieron a los valores promedio del secado por refractancia a 55 °C para fruta fresca, datos que fueron tomados de esta tesis.

El ajuste tipo modela de mejor forma procesos de deshidratación con humedad inicial intermedia (<0.9), esto debido a que para el “t” igual cero, el primer coeficiente corresponde al valor constante de la serie.

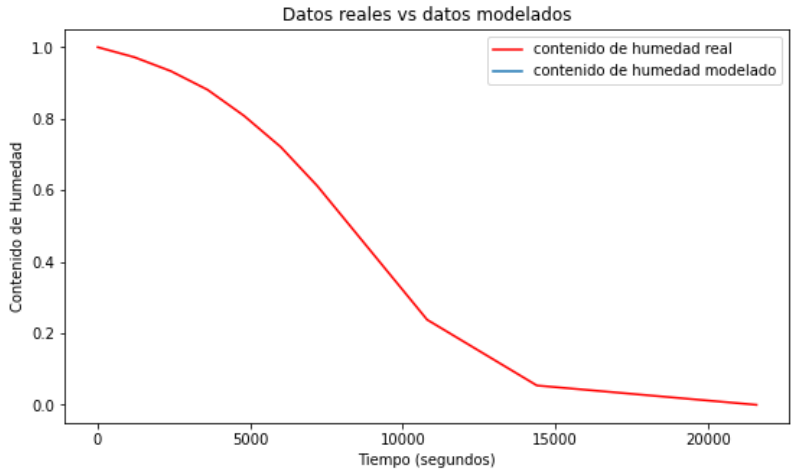


Figura 25. Datos de deshidratación experimentales

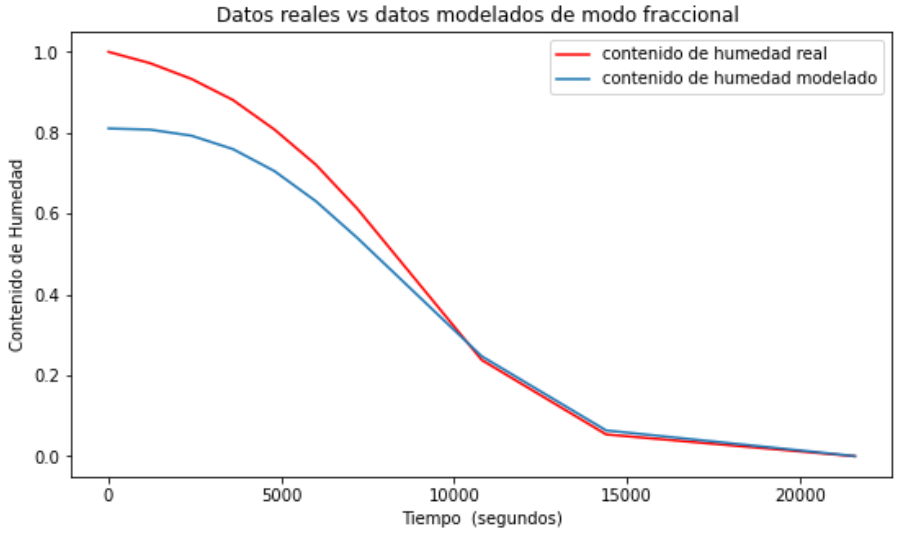


Figura 26. Modelamiento tipo Difusion Anomala. De forma analoga se grafican los datos reales y los datos modelados por medio del modelamiento de difusion anomala.

El programa desarrollado en Python pudo obtener los parámetros de la regresión no lineal de carácter fraccional gracias a la ayuda de sus librerías asociadas al tratamiento y análisis de datos. Los valores de los coeficientes están acordes a los presentados en literatura.

Si bien se logran excelentes ajustes en el modelado usando Python y el método de mínimos cuadrados, existe una dificultad asociada a cada modelo y esta es el de encontrar las condiciones de frontera que deben ser asumidas, es decir, las condiciones iniciales con las que se quiere que arranque el ajuste y los intervalos (valores máximos y mínimo) en que los valores a encontrar se suponen que se encuentran, valores que deben ser configurados de forma manual por el usuario, hasta encontrar un R^2 optimo y un bajo valor de χ^2 .

7. Conclusión

El programa creado en Python para la determinación de vida útil tuvo un desempeño exitoso al compararse con los datos presentados en la literatura. El programa no solo fue capaz de realizar 500 simulaciones, sino que pudo realizar hasta 5000000. Estas elevadas iteraciones podrían ser de utilidad para determinar valores que tengan asociados mucha incertidumbre. También se intentó mejorar la interfaz del usuario en el programa de Montecarlo, pero la programación web no soporto las iteraciones, además, solo desde el año pasado se hizo compatible la programación en HTML5 y Python.

El modelado de datos se desarrolló de manera satisfactoria y los resultados estuvieron de acuerdo con los valores presentados en literatura, tanto para el modelado mediante Midilli Kucuk y el modelamiento de difusión anómala. El uso de las bibliotecas y funciones para el modelamiento y tratamiento de datos por Python es gratuito a diferencia de otros programas que necesitan licencia para obtener todas las funcionalidades.

La utilidad de llevar cabo una simulación de Montecarlo en el campo de ciencias de los alimentos es el respaldo estadístico al momento de escoger un valor. Si bien la obtención de resultados puede ser apoyados por un intervalo de confianza del 95%, a la industria no le sería conveniente en términos económicos, ya que se limita la vida útil del producto en comparación con el valor obtenido de la forma determinística clásica, Pero como los alimentos son sistemas complejos y son muy susceptibles a distintos tipos de deterioro, este método entrega un resultado de vida útil más realista.

Se explico el funcionamiento, las bibliotecas y las funciones utilizadas en ambos programas, para que este trabajo sirva como guía a estudiantes de la carrera de ingeniería en alimentos que quieran realizar trabajos de modelamientos complejos así como simulaciones por Montecarlo.

La creación de ambos programas se llevó a cabo desde cero mediante ensayo y error. Para desarrollar estos algoritmos se tuvo que recurrir a literatura y otros recursos de internet, como foros y distintos canales de YouTube dedicados al tratamiento de datos. La habilidad de programación debe ser ejercitada y estudiada constantemente para así no perder la costumbre y mantenerse a la vanguardia en cuanto a las nuevas funciones y bibliotecas. Los enlaces a los programas se encuentran en anexo.

Dado que los dos programas cumplen su función de manera satisfactoria se puede que concluir que la hipótesis es verdadera.

8. Bibliografía

Alapont, C., Soriano, P., y Torrejón, M. (2020). Guía para la determinación de la vida útil de los alimentos. Conselleria de Sanitat Universal i Salud Publica. https://www.icoval.org/images/todoguiasappcc/vida_util.pdf

Alvarado, J. (2018). Cálculo de tiempos de vida útil en dulce de leche elaborado en Ecuador. Universidad de Santiago de Cali. https://libros.usc.edu.co/index.php/usc/catalog/download/74/79/1260?inline=1

Anderson, H.L. (1986). Metropolis, Monte Carlo and the MANIAC. *Los Alamos Science.* 14, 96-108. https://library.lanl.gov/cgi-bin/getfile?00326886.pdf

Anu, M. (1997). Introduction to modeling and Simulation. State of New York at Binghamton, Department of systems Science and Industrial Engineering Binghamton. doi: 10.1145/268437.268440

Arslan, N., y Tog'rul, H. (2006). The fitting of various models to water sorption isotherms of tea stored in a chamber under controlled temperature and humidity. *Journal of Stored Products Research*, 42(2), 112–135. doi:10.1016/j.jspr.2005.01.001

Ayala-Aponte, A. (2011). Estimación de las isotermas de adsorción y del calor isostéricos en harina de yuca. *Biotecnología en el sector agropecuario y agroindustrial*. 9(1), 88-96. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-35612011000100011&lng=en&tlng=es

Bielajew, A., y Sempau, J. (2000). Towards the elimination of Monte Carlo statistical fluctuation from dose volume histograms for radiotherapy treatment planning. *Physics in medicine and Biology*. 45, 13-57. doi: 10.1088/0031-9155/45/1/310

Cassin, M., Paoli, G y Lammerding, A.(1998).Simulation Modeling of microbial Risk Assessment. Journal of food Protection.
doi.org/10.4315/0362-028X-61.11.1560

Chotyakul, N., Velazquez,G y Torres, j.(2011).Assessment of the uncertainty in thermal food processing decisions based on microbial safety objectives.Journal of food engineering.
doi.org/10.1016/j.jfoodeng.2010.08.027

Crespo, L. (2011). Establecer el efecto del empleo de un antioxidante en la vida útil de dos variedades de maní ecuatoriano para confitería. Escuela superior Politécnica del Litoral.
https://www.dspace.espol.edu.ec/retrieve/6f5baba2-5e49-4079-a0d9-75484c0ef69a/D-79614.pdf

Demarchi, S; Quintero, N; De Micheles, A; y Giner, S. (2013). Sorption characteristics of rosehip, apple and tomato pulp formulations as determined by gravimetric and hygrometric methods. *LWT - Food Science and Technology*. 52, 21-26. doi:10.1016/j.lwt.2012.12.007

Escobedo-Avellaneda, S., Velázquez, G., Torres, J. A., y Welti-Chanes, J. (2012). Inclusion of the variability of model parameters on shelf-life estimations for low and intermediate moisture vegetables. LWT - *Food Science and Technology*. 47(2), 364–370. doi:10.1016/j.lwt.2012.01.032

Eckhardt, R. (1987)**.** Stan Ulam, John von Neumann, and the Monte Carlo Method. Los Alamos Science, Special Issue dedicated to Stanislaw Ulam. 15, 131-136. https://edisciplinas.usp.br/pluginfile.php/4946238/mod_folder/content/0/Eckhardt_LAS_1987.pdf?forcedownload=1

Frame, S.J. (2008). Simulation and Monte Carlo: With Applications in Finance and MCMC. *Journal of the American Statistical Association*, *Taylor & Francis*. 103, 890-891. doi: 10.1198/jasa.2008.s241

Hernández, N. (2017). Deshidratación de manzanas tipo Granny Smith en ventana refractiva con pretratamiento de deshidratación osmótica y campo eléctrico moderado. Tesis de pregrado, Universidad técnica Federico Santa María. https://repositorio.usm.cl/handle/11673/23645

Iglesias, H. (2012). Handbook of food isotherms: Water sorption parameters for food and food components. Elsevier. https://scholar.google.com/scholar_lookup?title=Handbook%20of%20Food%20Isotherms&publication_year=1982&author=H.A.%20Iglesias&author=J.%20Chiriffe

Jayas, D. (2016). Food Dehydration. *Reference Module in Food Science*. doi:10.1016/B978-0-08-100596-5.02913-9.

Johansen, A.M. (2010). Monte Carlo Methods. International Encyclopedia of Education (Third Edition). 296-303. doi:10.1016/B978-0-08-044894-7.01543-8.

Jurado, V., y Pacheco, C. (2019). Determinación de los factores extrínsecos e intrínsecos que afectan la vida útil de la malteada nutrangel en Cúcuta en el año 2018-2019. Universidad de Santander, San jose de Cúcuta. https://repositorio.udes.edu.co/bitstreams/5c42f351-3856-4235-a869-fd791d36fcdd/download

Karhanis, S., Stark, N., Sabo, R., y Matuana, L. (2021). Potential of extrusion-blown poly(lactic acid)/cellulose nanocrystals nanocomposite films for improving the shelf-life of a dry food product. Food packaging and shelf life. 29. doi: 10.1016/j.fpsl.2021.100689

Klenke, A. (2013). Probability Theory: A Comprehensive Course. Springer, 347. https://scholar.google.cl/scholar_url?url=http://103.62.146.201:8081/xmlui/bitstream/handle/1/8598/2013_Book_ProbabilityTheory.pdf%3Fsequence%3D1%26isAllowed%3Dy&hl=es&sa=X&ei=1524ZKmuJ4WM6rQPvqSN-A0&scisig=ABFrs3xWu6HBMND57LGLV_1NIb72&oi=scholarr

Kroese, D. P., Brereton, T., Taimre, T., y Botev, Z. I. (2014). Why the Monte Carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*. 6(6), 386–392. doi:10.1002/wics.1314.

Labuza, T. P., y Altunakar, B. (2007). Diffusion and Sorption Kinetics of Water in Foods. Water Activity in Foods, 215–237. doi: 10.1002/9780470376454.ch9

Man li,. Kexue zhu., Xu Guo., Wei Peng., Huiming Zhou.(2011). Effect of water activity (Aw) and irradiation on the shelf-life of fresh noodles. Innovative food science and emerging technologies. https://doi.org/10.1016/j.ifset.2011.06.005

Maltinia, E, Torreggianni,D., Venir,E y Bertolo,G (2003). Water Activity and the preservation of plant foods.Foods Chemistry. doi.org/10.1016/S0308-8146(02)00581-2

Marvasi, M., George, A., Giurcanu, M., Hochmuth, G., Noel, J., y Teplitski, M. (2015). Effect of the irrigation regime on the susceptibility of pepper and tomato to post-harvest proliferation of Salmonella enterica. *Food Microbiology*. 139-144. doi: 10.1016/j.fm.2014.07.014

Roudaut .,G Debeaufort., F(2011).Moisture loss, gain and migration in foods. Foods and Beverage Stability and Shelf Life. Woodhead Publishing series in Foods Science, Technology and Nutrition. doi.org/10.1533/9780857092540.1.63

Reglamento Sanitario de los Alimentos (RSA). (1996). Santiago, Chile: Jurídica Manuel Montt. https://www.dinta.cl/wp-content/uploads/2019/03/RSA-DECRETO_977_96_act_enero-2019_DINTA_.pdf

Rosen, S. (1969). Electronic Computers: A Historical Survey. ACM Computing Surveys. 1(1), 7–36. doi:10.1145/356540.356543

Rico, D., Sayani, H., y Field, R. (2008). History of Computers, Electronic Commerce and Agile Methods, Advances in Computers. 73, 1-55. doi:10.1016/S0065-2458(08)00401-4

Sammet, J. E. (1972). Programming languages: history and future. Communications of the ACM. 15(7), 601–610. doi:10.1145/361454.361485

Soporte de Microsoft. (2023). Especificaciones y límites de Excel. https://support.microsoft.com/es-es/office/especificaciones-y-l%C3%ADmites-de-excel-1672b34d-7043-467e-8e27-269d656771c3

https://support.microsoft.com/es-es/office/funci%C3%B3n-distr-norm-inv-87981ab8-2de0-4cb0-b1aa-e21d4cb879b8

Talens, P. (2007). Determinación de la isoterma de sorción de agua un alimento. Universidad politécnica de valencia. http://hdl.handle.net/10251/83506

Timmermann, E., Chirife, J., e Iglesias, H. (2011). Water sorption isotherms of foods and foodstuff: Bet or GAB parameters? *Journal of Foods Engineering*.Volume 48, 19-31. doi.org/10.1016/S0260-8774(00)00139-4

Uana da Silva,M., Sato, J., Martins, P., Janeiro, L,. y Souza, R. (2022). Modeling moisture adsorption isotherms for extruded dry pet foods. Animal Feed Science and Technology. 290. doi:10.1016/j.anifeedsci.2022.115318.

Vasta, M. (2009). Kai Velten: Mathematical modeling and simulation. Introduction for scientists and engineers. *Meccanica Journal*. 44(6), 767–768. doi:10.1007/s11012-009-9215-1

Vega-Gálvez, A., Puente-Díaz, L., Lemus-Mondaca, R., Miranda, M., y Torres, M. J. (2012). Mathematical Modeling of Thin-Layer Drying Kinetics of Cape Gooseberry (Physalis peruviana L.). *Journal of Food Processing and Preservation*. 38(2), 728–736. doi:10.1111/jfpp.12024

Zelle, J.M (2004). Python programming: An Introduction to computer science. Franklin, Beedle and Associates inc. https://scholar.google.com/scholar_lookup?title=Python%20Programming%3A%20An%20Introduction%20to%20Computer%20Science&publication_year=2004&author=J.M.%20Zelle

9. ANEXO

ANEXO 1. Código para imprimir por pantalla el mensaje “hola mundo” en Python

```
def hola():
    print("Hola mundo")

hola()
```

ANEXO 2. Mensaje mostrado en pantalla

```
In [1]: runfile('C:/Users/jc
Hola mundo

In [2]:
```

ANEXO 3. Secuencia de instrucciones para imprimir “hola mundo” en lenguaje de programación Java.

```
public class prueba{
    Run | Debug
    public static void main (String[] args){

        System.out.println(x: "hola mundo");

    }
}
```

ANEXO 4. Redondeo al menor entero posible por parte de la funcion “int” en python

```
>>> int(3.9)
3
```

ANEXO 5. Distintos tipos de mecanismos de deterioro en distintas matrices alimentarias.

GRUPO DE ALIMENTOS	PRODUCTOS	MECANISMOS DE DETERIORO	CAMBIOS LIMITANTES
	Refrescos bajos en calorías	Hidrólisis	Pérdida de dulzor
DETERIORO EN LECHE Y PRODUCTOS LÁCTEOS En general pueden experimentar oxidación, rancidez hidrolítica, crecimiento bacteriano, pardeamiento y cristalización lactosa.	Helados	Pérdida de humedad y oxidación.	Formación de cristales de hielo y rancidez (oxidación).
	Leche	Oxidación, reacciones hidrolíticas. Crecimiento microbiano	Rancidez y sabores extraños.
	Leche en polvo	Adsorción de humedad y oxidación.	Apelmazamiento, rancidez y cambios de sabor.
	Mantequilla	Oxidación	Rancidez
	Queso fresco y semicurado	Oxidación, cristalización lactosa y crecimiento microbiano.	Rancidez, textura arenosa, producción mohos.
	Yogurt	Sinéresis y oxidación.	Separación del suero y rancidez.
DETERIORO EN PRODUCTOS DE LA PESCA FRESCOS Generalmente puede observarse crecimiento bacteriano, oxidación, reblandecimiento de la carne, cambios de coloración.	Pescado fresco	Crecimiento microbiano y reacciones químicas	Pérdida de firmeza de la carne, opacidad de la córnea, olores y sabores desagradables (sulfuro, amoniaco), pérdida de piel o escamas, mucus lechoso-opaco superficial.
	Marisco fresco	Crecimiento microbiano y reacciones químicas	Pérdida de firmeza de la carne, olores y sabores desagradables (sulfuro, amoniaco), mucus lechoso-opaco superficial, cambios de coloración (melanosis).

ANEXO 6. Identificación única de los objetos en su correspondiente ejecución. Este valor cambia por cada ejecución del programa. Si bien el nombre de la variable se llama matriz, estas corresponden a listas vacías.

```
>>> matriz = []
>>> matriz_2 = matriz
>>> id(matriz)
2720558111616
>>> id(matriz_2)
2720558111616
```

ANEXO 7. Salida de los datos ya en forma de matriz luego de la ejecución del subprograma "conversor". Para este ejemplo existen 6 columnas las cuales corresponden a 6 sales ingresadas.

```
matrices con humedades calculadas  g de agua/ 100 g de solido seco

[ 12.61670770 16.41271794 18.78353484 21.69703918 23.97247010 25.66264826
[ 13.84774104 17.79735890 19.84036751 22.62173017 24.86415897 26.35934924
[ 14.88585104 19.26431106 21.90294486 26.02635749 29.72456436 32.39263827
[ 15.39109537 19.13280568 21.54334164 25.03178373 28.21785687 30.40012638
[ 12.70229104 17.19254512 19.89278895 23.29965500 26.18467089 28.32896825
[ 13.38653598 17.79856747 20.16857998 23.51546919 26.22064045 28.16442731
[ 14.33582331 18.58646033 21.11325250 25.06489094 28.40624786 30.94353903
[ 13.62692041 17.53949235 19.57394606 22.65272263 25.12295389 26.92852154
[ 13.29959366 17.60731120 20.15306492 23.61022615 26.61226533 28.75585327
[ 12.81798273 16.64189774 18.82413669 21.64727835 23.89368791 25.44333458
[ 15.55209496 19.64935516 22.50799055 26.61584043 30.09765774 32.85804714
[ 14.09533483 17.52280392 19.54439037 22.43973736 24.92384335 26.71960960
[ 13.30480726 18.35741008 21.51130940 26.13004955 30.22165971 33.18047912
[ 14.50530648 18.52722107 21.06855092 24.84681162 28.22438735 30.60311509
```

ANEXO 8. Resumen el análisis estadístico realizado por el programa estadística para cada columna que asocia la humedad a las sales involucradas en la simulación.

```
los valores maximos de cada sal son: [15.86516178 19.87766858 22.50799055 26.61584043 30.44440
 46.42856089 56.31467101 58.13495302 66.19451799]

los valores minimos de cada sal son: [ 8.2944377  13.79335781 16.69463757 20.88059909 23.42670
 30.36869622 33.50818156 33.93221764 36.05119954]

el promedio de las sales es:  [13.71918671 18.05361163 20.57616397 24.21407036 27.31027934 29.
 37.77883261 43.23469898 44.27387582 48.68212741] g de agua/ 100 g de solido seco

el promedio de las sales en g de agua/ g de solido seco es :  [0.13719187 0.18053612 0.2057616
 0.37778833 0.43234699 0.44273876 0.48682127] estos valores corresponden al eje X de la regres

la desviacion de cada sal es:  [1.22743514 1.03053075 1.07109197 1.32035597 1.68862641 2.01908
 3.69082833 5.15715796 5.46609322 6.86601808]

se tienen los siguientes valores

el promedio de las sales en g de agua100 g de solido seco es :  [13.71918671 18.05361163 20.57
 37.77883261 43.23469898 44.27387582 48.68212741] estos valores corresponden al eje y de la re

los valores promedio de cada sal son: [0.11293892 0.23809005 0.32202084 0.4319199  0.51001096
 0.69126171 0.75288667 0.76292456 0.80089606] estos valores corresponden al eje x del grafico
```

ANEXO 9. Un arreglo en una dimensión “a” es transformado en una matriz de 3x5 según los parámetros entregados a la función np.reshape .

```
import numpy as np
a = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
alfa = np.reshape(a,(5,3))
print(alfa)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]]
```

ANEXO 10. Matriz promedio de Aw y g agua/g solido-asociados a cada valor de sal

```
ingrese la posicion desde la cual quiere empezar a trabajar, posicion desde la 1- 10

[['1' 'LICL' '0.11300038409508623' '0.13733224160824345']
 ['2' 'CH3COOK' '0.2379907449603656' '0.18096276756971613']
 ['3' 'MGCL2' '0.3220029919871858' '0.20661405493978413']
 ['4' 'K2CO3' '0.43200350811500177' '0.24356579399070466']
 ['5' 'MG(NO3)2' '0.5100045147601766' '0.2752023063233392']
 ['6' 'NABR' '0.5579970042631642' '0.29815739543747516']
 ['7' 'SRCL2' '0.691002471345609' '0.38435642921810126']
 ['8' 'NACL' '0.753005648609559' '0.4393688129927628']
 ['9' 'NH4CL' '0.7630053997982987' '0.45086594884735204']
 ['10' '(NH4)2SO4' '0.8009994889482374' '0.49669866028718934']]
```

ANEXO 11. Repetición del cálculo por Montecarlo para mayor variabilidad.

```
A continuacion, se calculara mediante MONTECARLO iteraci
interes asociadas a una actividad de agua Aw
ingrese la posicion desde la cual quiere empezar a traba

[['1' '' '0.11297901625144942' '0.13796100371927716']
 ['2' '' '0.23810560370443257' '0.18146290220330819']
 ['3' '' '0.3219978889418679' '0.20688106074035012']
 ['4' '' '0.43197125013142834' '0.24371474862445772']
 ['5' '' '0.5099716817750587' '0.275100285177068']
 ['6' '' '0.5580244827411146' '0.29777596386863303']
 ['7' '' '0.6909794369855028' '0.38183128832533186']
 ['8' '' '0.7530499690389322' '0.4384636712657604']
 ['9' '' '0.7629334379275952' '0.4490478360754946']
 ['10' '' '0.8010186170584891' '0.4950674500648104']]

ingrese su opcion desde la cual quiere trabajar:
```

ANEXO 12. Elección del ancho del intervalo a contar. Por ejemplo, si el ancho del intervalo es de 10, el programa los agrupara en grupos de diez días.

```
Este es el eje x [0.69116276 0.75368758 0.76336705 0.80174296]
este es el eje y [0.35052102 0.39197178 0.39923122 0.4307356 ]
Valor minimo de pendiente 0.40015253648010757 y el maximo de pendiente e
Valor minimo de interceptos -2.133184616387305 y el maximo de intercepto
elija los intervalos de dias por los que quiere contar:
```

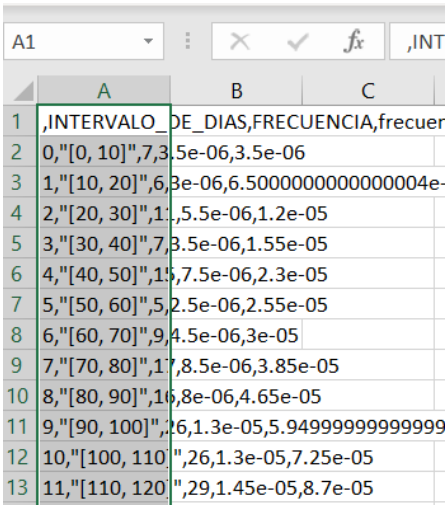
ANEXO 13.

1)Primero se deben seleccionar todos los datos con CTRL +SHIFT+flecha abajo del teclado.

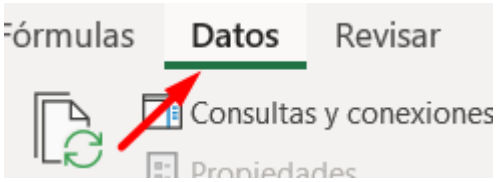
2)Luego pinchar en “datos” e ir a “texto en columnas”.

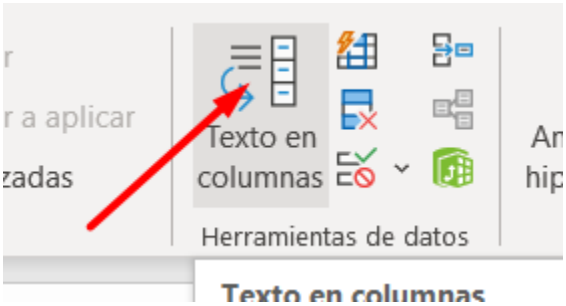
3)Se selecciona delimitado y se da clic en siguiente

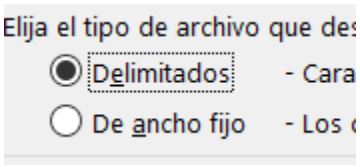
4)Se selecciona el separador por comas y se da clic siguiente ahora los datos queda listos para futuras operaciones.

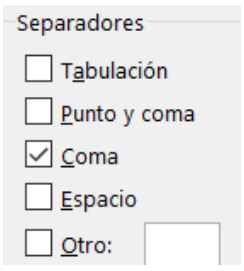


	A	B	C
1	,INTERVALO_	DE_DIAS,FRECUENCIA,frecuer	
2	0,"[0, 10]",7,3	.5e-06,3.5e-06	
3	1,"[10, 20]",6,	3e-06,6.5000000000000004e-	
4	2,"[20, 30]",11	,5.5e-06,1.2e-05	
5	3,"[30, 40]",7,	3.5e-06,1.55e-05	
6	4,"[40, 50]",15	,7.5e-06,2.3e-05	
7	5,"[50, 60]",5,	2.5e-06,2.55e-05	
8	6,"[60, 70]",9,	4.5e-06,3e-05	
9	7,"[70, 80]",17	,8.5e-06,3.85e-05	
10	8,"[80, 90]",16	,8e-06,4.65e-05	
11	9,"[90, 100]",	26,1.3e-05,5.94999999999999	
12	10,"[100, 110	",26,1.3e-05,7.25e-05	
13	11,"[110, 120	",29,1.45e-05,8.7e-05	









ANEXO 14. Regresión lineal para todos los elementos involucrados.

```
Esta matriz corresponde a las sales involucradas-Los valores de AW  de cada sal
[['1' 'LiCl' '0.11293892266826186' '0.13719186712144654']
 ['2' 'CH3CooK' '0.2380900464893941' '0.180536116312154']
 ['3' 'MgCl2' '0.3220208432199193' '0.2057616396938405']
 ['4' 'K2CO3' '0.43191990263261526' '0.24214070359055687']
 ['5' 'Mg(NO3)2' '0.5100109633075398' '0.27310279336445903']
 ['6' 'NABR' '0.5580993863673238' '0.2953935396680831']
 ['7' 'SRCL2' '0.6912617140662336' '0.37778832608488444']
 ['8' 'NACL' '0.7528866698367852' '0.43234698980687l7']
 ['9' 'NH4CL' '0.7629245628454767' '0.4427387582062166']
 ['10' '(NH4)2SO4' '0.8008960565040938' '0.4868212740704926']]

la pendiente es : 0.0886380123566309 y su intercepto es: 0.26145841166148376

la ecuacion de la recta es: y =  0.0886380123566309 X + 0.26145841166148376
```

ANEXO 15. Ecuación y ajuste de la recta.

```
la ecuacion de la recta es: y =  0.5053535759263187 X + 0.04785580991994154
el ajuste de la ecuacion es de:  0.9561622335181028
```

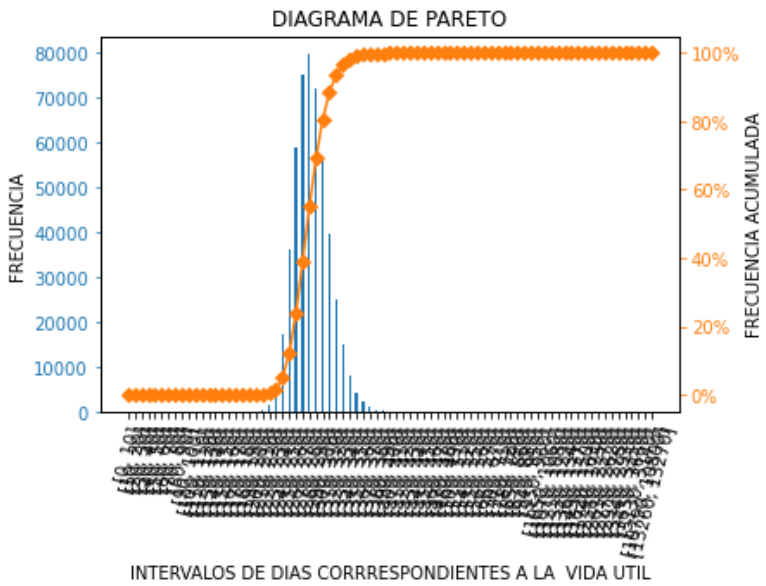
ANEXO 16. Ajuste de la regresión lineal para las ultimas 4 sales de interés que presentan una mayor Aw asociadas

```
09794369855028, 0.7530499690389322, 0.7629334379275952, 0.8010186170584891] su coordenada
83253318, 0.4384636712657604, 0.449, 0.495]
: y = 1.015262244658422 X+ -0.3223987626013418 y su ajuste es 0.9925654666535715
```

ANEXO 17. Duración del producto mediante convergencia de todas las iteraciones.

```
0.001603731201163345
la duracion del producto es 242.05421633988865 dias, lo que es lo mismo que decir 8.068473877996288 meses de
duracion
```

ANEXO 18. Obtencion de una distribucion de 500000 valores de vida util ordenados en interalos de 10 dias



ANEXO 19. Análisis estadísticos y parámetros encontrados mediante en ajuste curve_fit.

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 184
    # data points      = 39
    # variables        = 3
    chi-square         = 3.1827e-18
    reduced chi-square = 8.8409e-20
    Akaike info crit   = -1708.14203
    Bayesian info crit = -1703.15135
    R-squared          = 1.00000000
[[Variables]]
    k:  5.9613e-04 +/- 1.7507e-12 (0.00%) (init = 1e-06)
    b: -1.6211e-05 +/- 1.4491e-13 (0.00%) (init = -1e-05)
    n:  1.15710000 +/- 4.9824e-10 (0.00%) (init = 1e-06)
[[Correlations]] (unreported correlations are < 0.100)
    C(k, n) = -0.994
    C(b, n) = 0.749
    C(k, b) = -0.684
```

ANEXO 20. Reporte estadistico asociado a la regresion no lineal de difusion anomala.

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 189
    # data points      = 9
    # variables        = 2
    chi-square         = 0.15609138
    reduced chi-square = 0.02229877
    Akaike info crit   = -32.4908443
    Bayesian info crit = -32.0963951
    R-squared          = 0.99683362
[[Variables]]
    difusividad2:  4.1916e-17 +/- 4.5089e-17 (107.57%) (init = 1e-13)
    alfa:          2.64342025 +/- 0.10986980 (4.16%) (init = 1)
[[Correlations]] (unreported correlations are < 0.100)
    C(difusividad2, alfa) = -1.000
```

ANEXO 21. Resultados informados por Hernández (2017) para obtener los coeficientes de difusividad.

	Fruta fresca			**Fruta pretratada**		
	$\alpha = 2{,}231$			$\alpha = 1{,}452$		
	$D_{eff} \cdot 10^{14}$ m²/sᵅ	R^2	SEE	$D_{eff} \cdot 10^{11}$ m²/sᵅ	R^2	SEE
VR 55 °C	0,11	0,9877	1,2E-02	0,36	0,9891	2,9E-03
VR 75 °C	0,81	0,9811	7,5E-03	1,06	0,9878	8,8E-03
VR 95 °C	3,00	0,9927	8,0E-04	2,52	0,9927	8,0E-03

ANEXO 22. Datos extraídos de Hernández (2017) del secado a 55°C por ventana de refractancia de manzana para llevar a cabo la modelación de difusión anómala.

Tiempo (minutos)	Contenido de Humedad
0	1
20	0.972
40	0.933
60	0.881
80	0.808
100	0.721
120	0.612
180	0.238
240	0.054
360	0.0

Enlaces de los programas

1)https://github.com/Alpacawithglasses/Montecarlo

2)https://github.com/Alpacawithglasses/modelamiento

3) https://github.com/Alpacawithglasses/Manual.git

Apéndice

Definiciones y funciones básicas en Python

INPUT: Cuando se ejecuta esta función en alguna línea de código, esta espera a recibir un texto por medio del teclado. Por defecto, el texto ingresado sea numérico o no, tiene carácter de "STRING". "Input" almacena el texto ingresado en la o las variables asignadas para ello.

If : Condicional que toma una acción según una condición booleana, es decir, si la condición es verdadera o falsa.

STRING: Cadena de datos formados por caracteres iguales o distintos. Son datos no numéricos.

INT: Cuando se coloca esta función antes del "input" la variable que se espera a recibir en Python es transformada a un carácter numérico de tipo "entero de precisión fija", esta función no es válida para datos de tipo letra o de caracteres. Si la variable recibe por parte del usuario un numero de tipo decimal (de punto flotante en Python), la función "int" redondeara al entero de menor valor posible (ANEXO 4).

Def: Se ocupa para definir y crear un programa. Se definen el nombre de la función junto a los parámetros que necesita para funcionar.

Expresiones regulares(re.): Sirven para operar específicamente cadena de textos con distintas funciones, por ejemplo, la función de expresión regular "re.match", corresponde a patrones de búsqueda dentro de un texto. La función "match" de los objetos regulares, hace la búsqueda y

devuelve un valor verdadero en caso de que el valor ingresado se encuentre dentro de los posibles valores ya definidos.

Print: Esta función es la más común en los lenguajes de programación, si bien cambia la sintaxis dependiendo el lenguaje en el que se esté implementando el programa, su función es la misma, mostrar un mensaje por pantalla. Puede servir para mostrar el resultado final de unas operaciones lógicas, operaciones aritméticas, o como una serie de instrucciones para facilitar el uso de los programas.

Scipy: Desde la página oficial "docs.scipy.org" se informa que en particular su modulo "optimize", permite maximizar o minimizar funciones objetivo que posiblemente se encuentren sujetas a restricciones. Por otro lado, "curve_fit" utiliza mínimos cuadrados no lineales para ajustar una función a los datos de prueba. Existen otras funciones de interés dentro de esta biblioteca que permiten solucionar problemas que no se pueden resolver de la forma analítica, y estas son: La función "Newton" (que aplica el algoritmo de Newton-Raphson) o el algoritmo de "Tom", por nombrar algunas.

Lmfit: Se basa en los métodos de optimización de "Scipy", pero mejorados. Proporciona una interfaz de alto nivel para la optimización no lineal y problemas de ajuste de curvas de Python. El uso de "Parameters" como objetos, le da más robustes y atributos a la hora de realizar optimizaciones, ya que estos parámetros son modificables y se pueden acotar a intervalos definidos por el programador.

Sum (): Suma todos los elementos numéricos de una lista, tupla o un objeto de clase Numpy.ndarray. Devuelve un valor numérico, por lo que los datos de entradas no deben ser alfanuméricos o números en forma de Strings.

Np.array(lista).reshape(len(lista),z): Se utiliza para redimensionar vectores. Necesita como parámetros el arreglo o vector, el cual está indicado como "lista". "Len(lista)" obtiene la dimensión del vector lista y se toma como parámetro y lo convierte en un vector de "len (lista)" dimensiones. Finalmente el valor de "z" indica el número de columnas, mientras que len(lista) se asocia con el número de filas. (ANEXO 9)

Lista_de_sales: Provino del programa "cortador" y es una lista que solo contiene el nombre de las sales.

Posición: Es una lista con números de 1 a n, donde “n” corresponde a la cantidad de sales ingresadas.

Promediow, promedio y promedio2: Estas tres variables son del tipo ‘numpy. Ndarray’, la cuales fueron creadas en subprogramas anteriores. Un objeto de clase “numpy.ndarray”, es la estructura de datos central de la librería Numpy, y los objetos que tengan esta clase (características), van a poder ser operados directamente con esta librería y utilizar todos sus módulos ya definidos.

Penultimamatriz = np.concatenate((matriz_1, matriz2, matriz3, matriz4),1): Se utilizo esta función para juntar todas las matrices en una sola mediante la concatenación. Se entregan entre paréntesis las matrices a unir seguido de una “,” como separador para luego agregarle un número. Si no se agrega este último número (o si es que se coloca un “0”) la concatenación se llevara a cabo por filas. El numero “1” por otra parte, indica que la concatenación se llevara a cabo por columnas.