UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE QUÍMICA

PROGRAMACIÓN DE MÉTODOS NUMÉRICOS EN PYTHON APLICADOS A PROBLEMAS DE INGENIERÍA QUÍMICA

TESIS

QUE PARA OBTENER EL TÍTULO DE

INGENIERO QUÍMICO

PRESENTA

CARLOS DANIEL RODRIGUEZ SOTELO

MÉXICO, D.F. **2015**

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

JURADO ASIGNADO:

PRESIDENTE: **Profesor: Juan Carlos Jiménez Bedolla**

VOCAL: **Profesora: Aida Gutiérrez Alejandre**

SECRETARIO: **Profesor: Juan Pablo Aguayo Vallejo**

1er. SUPLENTE: **Profesor: Ernesto José Calderón Castillo**

2° SUPLENTE: **Profesor: Carlos Álvarez Maciel**

SITIO DONDE SE DESARROLLÓ EL TEMA:

Centro de Ciencias Aplicadas y Desarrollo Tecnológico

(CCADET – UNAM)

ASESOR DEL TEMA:

Juan Pablo Aguayo Vallejo

SUSTENTANTE:

Carlos Daniel Rodríguez Sotelo

Agradecimientos

Este trabajo se lo dedicó:

A mi Madre y a mí Tía Aida Gracias por el trabajo y el tiempo que invirtieron en mí pero lo más importante gracias por todo el amor y el cariño que me dieron.

A mi hermano eres una parte muy importante de mi vida, espero que podamos estar juntos por el resto de nuestras vidas.

A mi Tía Carmen, Tío Fernando, Padrino Carlos y a mi familia por estar presente siempre en mi vida dándome su apoyo.

A mis Amigos Joab, Rodrigo, Hippo, Chucho, Alan, y Debby Gracias por haberme brindado su compañía, enseñarme que lo más bonito de la vida es compartir el tiempo con personas como ustedes.

Al Doctor Aguayo gracias por brindarme la confianza para realizar esta tesis y el apoyo que me dio en el transcurso de esta. Sin usted no habría podido hacer la tesis. Muchas Gracias.

A mis Profesores de la carrera por la formación académica y personal recibida, especialmente al Ingeniero Ortiz y al Ingeniero Sergio por haberme dado la oportunidad de hacer el servicio con ustedes y al Doctor Chávez por ayudarme en el proyecto de estancia.

Al colegio de Profesores de la facultad de química y a la sección 24 de la AAPAUNAM por el apoyo otorgado mediante la cátedra Alberto Urbina del Raso.

Y especialmente agradezco a la Universidad Nacional Autónoma de México por haberme permitido estudiar en esta gran universidad.

ÍNDICE

CAPÍTULO I 1

Introducción 1

Comparación lenguajes de programación 3

Planteamiento del problema 4

Objetivos 5

CAPÍTULO II 6

Programando en Python 6

Entorno de programación 6

Documentación en Python 6

Comentarios 6

Declarando Variables 7

Importando Objetos 8

Números en Python 10

Cadena de texto 11

Colecciones de Datos 12

Indentación 14

Controles de flujo 15

Ciclos 17

Funciones 19

CAPÍTULO III 21

Ecuación algebraica de una variable 21

Método de Newton-Raphson 21

Sistemas de ecuaciones algebraicas lineales 25

Método Gauss - Jordan 25

Sistemas de ecuaciones algebraicas no lineales 28
Método de Newton multivariable 28
Integración numérica 34
Evaluación numérica de integrales 34
Regla de Simpson Compuesta 37
Cuadraturas Gaussianas 39
Ecuaciones Diferenciales Ordinarias con condiciones iniciales 43
Método de Euler 43
Método de Runge Kutta 44
CAPÍTULO IV 49
Problema de ecuación algebraica de una variable 49
Problema de sistema de ecuaciones algebraicas lineales 51
Problema de un sistema de ecuaciones algebraicas no lineales 53
Problemas de Integración numérica 55
Problema de evaluación numérica 55
Problema de integral numérica 56
Problema de ecuaciones diferenciales ordinarias con condiciones iniciales 57
CAPÍTULO V 60
Conclusiones 60
Comentarios 61
APÉNDICE 62
ÍNDICE DE TABLAS E ILUSTRACIONES 63
BIBLIOGRAFÍA 64

CAPÍTULO I

Introducción

Esta tesis surge ante la necesidad de demostrar la importancia de programar. En los últimos años se ha desarrollado la idea que no saber programar será el 'analfabetismo del futuro' y varios países han empezado a enfocar recursos para que los niños aprendan a programar (Vizcarra, 2014).

La importancia de Programar también está en la ingeniería Química. En 1997 la CACHE (Es una organización sin fines de lucro que su propósito es promover la cooperación entre industrias, gobierno y universidades en el desarrollo y distribución de apoyo educacional relacionado con las computadoras para los programas de Ingeniería Química) informo que el 95.4 % de los programas de ingeniería química tenían un curso de programación (David, s.f). La programación en la ingeniería química se ocupa en los siguientes análisis:

- Balances de materia
- Balances de energía
- Flujo de Fluidos
- Transferencia de masa
- Ingeniería de reactores

La CACHE también sugiere que los ingenieros químicos tengan la capacidad de comprender e implementar algoritmos elementales para la solución numérica de problemas. Los algoritmos deben de incluir la solución de ecuaciones lineales algebraicas y la solución de ecuaciones diferenciales.

Pero ¿Cómo demostrar la importancia de programar en la Ingeniería Química? y ¿Qué programar? , esas dos preguntas se hicieron al inicio de este trabajo y se llegó a la respuesta de que la mejor forma de demostrar el uso de la programación en la ingeniería química era la programación de métodos numéricos.

Los métodos numéricos son 'técnicas mediante las cuales es posible formular problemas matemáticos de tal forma que puedan resolverse usando operaciones

aritméticas' (Chapra & Canale, 2010). Los métodos numéricos son un excelente ejemplo de programación debido a la gran cantidad de operaciones aritméticas requeridas, las cuales pueden realizarse de manera rápida en la computadora, además de que resuelven una gran cantidad de problemas.

El lenguaje de programación que se ocupó fue Python, lenguaje que se escogió principalmente por:

- Es un lenguaje multiplataforma (compatible con cualquier dispositivo que tenga Python instalado como dispositivos móviles, teléfonos celulares, computadoras con Linux, Windows y OS).
- La sintaxis del lenguaje hace que sea fácil de aprender.
- Es **gratuito** y de fácil accesibilidad.
- Es un lenguaje ampliamente utilizado en diferentes industrias (entretenimiento, finanzas, computación científica, entre otras). Algunas de las empresas que ocupan Python para uso comercial se muestran en la Ilustración 1 Empresas que ocupan Python
- Contiene una gran cantidad de librerías disponibles de forma gratuita que facilitan aún más el desarrollo de nuevas aplicaciones.
- En Python se pueden crear funciones de alto orden; esto quiere decir que puede tomar funciones como argumentos y devolver funciones.
- En el lenguaje de programación Python se pueden manejar números de gran magnitud de forma automática
- Python tiene asignación múltiple de variables; esta propiedad de Python permite realizar más programas en una menor cantidad de líneas.

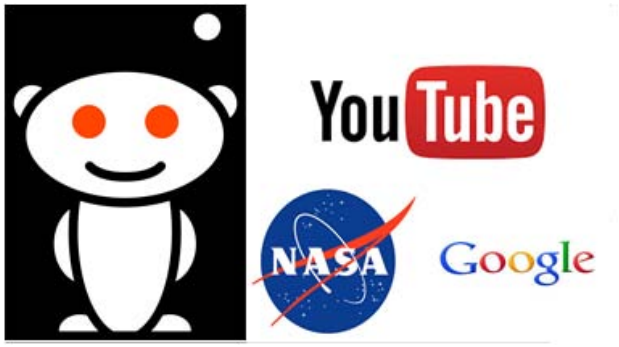

Ilustración 1 Empresas que ocupan Python

Comparación lenguajes de programación

En esta sección se quiere realizar una pequeña comparación entre diferentes lenguajes de programación según dos criterios: popularidad y desempeño. Algo muy importante a destacar es que sólo pocos lenguajes se pueden ocupar como lenguajes de computación científica (Boragan & Fernández-Villaverde, 2014).

En la tabla Tabla 1 Desempeño de funciones a comparación de C, se comparan diferentes lenguajes en ciertas funciones. Está tabla muestra como es el desempeño en funciones comúnmente ocupadas a comparación del lenguaje C.

Tabla 1 Desempeño de funciones a comparación de C

Función / Lenguaje	Fortran gcc 4.8.2	Julia 0.3.7	Python 2.7.9	R 3.1.3	Matlab R2014a	Mathematica 10	Java 1.70_75
fib	0.57	2.14	95.45	528.85	4258.12	166.64	0.96
parse_int	4.67	1.57	20.48	54.30	1525.88	17.70	5.43
quicksort	1.10	1.21	46.70	248.28	55.87	48.47	1.65
mandel	0.87	0.87	18.83	58.97	60.08	6.12	0.68
pi_sum	0.83	1.00	21.07	14.45	1.28	1.27	1.00

Fuente (http://julialang.org/)

Se puede observar que Python es lento a comparación de los lenguajes compiladores como Java o C sin embargo es más rápido que los lenguajes mathematica y Matlab en algunas funciones.

El indicador TIOBE es un índice que muestra la popularidad del lenguaje de programación. El indicador se realiza mensualmente y los resultados están basados en la cantidad de ingenieros que utilizan el lenguaje, la cantidad de cursos y la cantidad de vendedores. La tabla muestra la popularidad de los lenguajes según el indicador TIOBE.

Tabla 2 Indicador TIOBE 2015

Lenguaje	Lugar en el índice TIOBE
Java	1
C	2
Python	5
Matlab	15
Fortran	27
Mathematica	49

Fuente (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html)

Con esta información y debido a que Python es un lenguaje de código abierto, se puede decir que Python tiene una gran popularidad entre los lenguajes con usos reales en la industria y que puede ser utilizado para realizar cálculos de alta complejidad.

Planteamiento del problema

El problema que se quiere resolver con este trabajo es la realización de un archivo con extensión *.py* que contenga métodos numéricos programados en el lenguaje de programación Python. Con los métodos programados en este archivo se desean resolver diferentes problemas de ingeniería química y que sea una alternativa a programas como Polymath para problemas numéricos que se puedan resolver con los métodos descritos en la Tabla 3 Principales funciones programadas en metodosTesis.py.

Tabla 3 Principales funciones programadas en metodosTesis.py

Tema	***Problema***	***Método***	***Nombre en Python***
Ecuación algebraica de una variable	Encontrar la solución de una ecuación de una variable	Método Newton - Raphson	Newton (funcion , estimadoinicial)
Sistema algebraico de ecuaciones lineales	Encontrar la solución de un sistema de ecuaciones lineales	Método Gauss - Jordan	Gauss (matrix , vector)
Sistema algebraico de ecuaciones no lineales	Encontrar la solución de un sistema no lineal	Método Newton multivariable	newtonSistemas (funciones , puntos)
Integración Numérica	Encontrar la integral de una serie de puntos	Evaluación numérica de la integral	integracionNumerica (x , fx)
	Encontrar el valor de una integral numérica	Regla Simpson Compuesta	compositeSimpson (funcion , a ,b , n)
		Cuadraturas Gaussianas	cuadraturasGausianas (f , a , b , n)
Ecuaciones Diferenciales	Encontrar la solución de ecuaciones diferenciales con condiciones iniciales	Método de Euler	Euler(nP , Inicial , Final , funciones , valoresIniciales)
		Método de Runge Kutta	Runge (nP , Inicial , Final , funciones , valoresIniciales)

Objetivos

Los objetivos de esta tesis son los siguientes:

- Escribir un manual de referencia en Python. En este manual se describirán los comandos que se ocuparon para crear el archivo metodosTesis.py e ilustrar la facilidad de uso de estos comandos en Python.
- Recopilar los algoritmos de los métodos numéricos para programarlos en el archivo metodosTesis.py con los conceptos vistos en el manual de referencia.
- Hacer uso del archivo metodosTesis.py en problemas de Ingeniería Química para demostrar la importancia de los métodos programados.

CAPÍTULO II

Programando en Python

En este capítulo se ilustrarán los conceptos que se ocuparon para programar el archivo metodosTesis.py.

Entorno de programación

El entorno de desarrollo integrado (IDE) es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. El IDE que se ocupará en esta tesis es ENTHOUGHT CANOPY, el cual se puede descargar en la siguiente liga https://store.enthought.com/downloads/.

Las ventajas principales de ocupar un IDE son:

- La habilidad de escribir código y ver los resultados desde la consola de Python.
- Ayuda visual para la programación.
- Se pueden identificar los errores fácilmente y eliminarlos.
- Contienen una gran cantidad de librerías precargadas.

Documentación en Python

Para encontrar los manuales de Python usando Windows[1].

Inicio -> Todos los programas -> Python 2.7 -> Python Manuals

Comentarios

Los comentarios en Python se escriben con #.Después del símbolo de # la computadora ignorará esos caracteres. Los comentarios se ocupan para que el programa sea más fácil de entender.

[1] El sistema operativo que se ocupó en este trabajo es Windows 7 Home Premium.

Declarando Variables

En la mayoría de los lenguajes de programación (incluyendo java y C+) es necesario declarar cada variable especificando su tipo antes de usarla. Esto es llamado un tipado estático, porque el compilador conoce cada tipo de variable. Python como otros lenguajes de alto nivel ocupa un enfoque diferente. Las variables no tienen restricción de tipo y no se necesitan declarar. La sintaxis para asignar una variable en Python es:

```
<nombre> = <valor de la variable>
```

A continuación se presentaran algunos ejemplos que permiten ver las diferencias en cuanto a la asignación de variables en Python con respecto a un tipado estático como el usado en Java.

En Python

```
In [1]: x = 6

In [2]: x
Out[2]: 6
```

En Java

```
int x = 6;
x
6  (int)
```

En la Tabla 4 Palabras clave en Python presentan una lista con las palabras clave de Python. Las palabras clave son palabras reservadas, que exclusivamente las puede ocupar el lenguaje por esta razón no pueden ser ocupadas como variables.

Tabla 4 Palabras clave en Python

and	class	elif	finally	if	lambda	print	while
as	continue	else	for	import	not	raise	with
assert	def	except	from	in	or	return	yield
break	del	exe	global	is	pass	try	

Fuente (Summerfield, 2007)

Importando Objetos

Python tiene una gran librería de módulos que proveen una gran funcionalidad mediante funciones auxiliares. En Python se puede hacer uso de esta característica importando constantes, variables, funciones y clases.

La sintaxis general para importar es:

```
import <nombreDelModulo>
```

Ejemplo

Editor

```
1 import math #Se importa el modulo math
2 pi = math.pi #Se obtiene la constante pi del modulo math
3 print pi # Se imprime el valor de pi
```

Consola

```
In [1]: %run "c:\users\dany\appdata\local\temp\tmpnbq95t.py"
3.14159265359
```

Nota: La nomenclatura %run “ubicación del archivo’” muestra los resultados de la consola. El comando *print* sirve para mostrar los resultados en la consola. Mientras que en el editor se muestra que código se escribió. Esta forma de mostrar los programas se respetará en toda la tesis.

En este trabajo los principales módulos que se importaron son:

- math
- numpy
- matplotlib

El módulo math contiene funciones matemáticas precargadas. La Tabla 5 Algunas funciones del módulo math muestra las funciones más representativas del módulo math.

Tabla 5 Algunas funciones del módulo math

math.exp(x)	Devuelve e ** x
math.log(x [, base])	Devuelve el logaritmo natural de x. Si se agrega otro argumento devuelve el logaritmo de x de la "base" del segundo argumento
math.log10(x)	Devuelve el logaritmo base 10
math.pow(x,y)	Devuelve el valor de x elevada a la y
math.cos(x)	Devuelve el coseno de x en radianes
math.sin(x)	Devuelve el seno de x en radianes
math.tan(x)	Devuelve la tangente de x en radianes

Fuente (Python Software Foundation, 2015)

Nota: Las funciones trigonométricas en el módulo math están en radianes para cambiar de radianes a grados está la función *math.degrees(x)*.

El módulo numpy se ocupó en la programación en el método de Runge para manejar listas de gran tamaño. Mientras que la principal función del módulo matplotlib es graficar datos de Python. Estos módulos son necesarios para correr los programas que se realizaron en este trabajo[2].

También se ocupó el concepto de importar funciones en el archivo metodosTesis.py para poder ocupar las funciones programadas.

Ejemplo:

```
import metodosTesis
x = [1.0 , 1.1 ,1.2 , 1.3, 1.4 , 1.5, 1.6, 1.7]
fx = [1.543 , 1.669 , 1.811 , 1.971 , 2.151 , 2.352 , 2.577 , 2.828]
integral = metodosTesis.integracionNumerica (x , fx)
```

En este ejemplo se importa el archivo m*etodosTesis.py* para poder hacer uso de la función *integracionNumerica (x, fx)* es muy importante que si se quiere ocupar una función programada en metodosTesis.py en un nuevo archivo el nuevo archivo este en la misma carpeta que metodosTesis.py

[2] ***Nota:*** El IDE elegido para esta tesis tiene estos módulos incluidos

Números en Python

Números enteros son aquellos números que no tienen decimales, se denotan con el tipo int.

Consola

```
In [12]: x = 3

In [13]: type(x)
Out[13]: int
```

Números decimales son aquellos que tiene el punto decimal, se denotan con el tipo float.

Consola

```
In [14]: x = 3.14

In [15]: type(x)
Out[15]: float
```

La Tabla 6 Principales operaciones con números en Python muestra las principales operaciones con números en Python.

Tabla 6 Principales operaciones con números en Python

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
* *	Exponente
/	División
%	Módulo

Fuente (González, s.f)

Nota: Se puede observar que no es necesario declarar el tipo de número en el lenguaje programación, el lenguaje identifica que tipo de número es.

Cadena de texto

Las cadenas de texto en Python son creadas encerrando el texto entre comillas (“” o ‘’).

Consola

```
In [2]: texto =  "Hola"

In [3]: texto
Out[3]: 'Hola'
```

Cadenas de texto con formato

Se ocupa el operador % para dar formato a las cadenas de texto, del lado izquierdo se escribe la cadena de texto y del lado derecho se escribe una secuencia con los objetos a insertar en la cadena de texto.

Ejemplo

Consola

```
In [11]: 'El resultado de la %s es: %.2f' % ('V/F' , 0.490989)
Out[11]: 'El resultado de la V/F es: 0.49'
```

En el anterior ejemplo %s es para formatear una cadena de texto remplaza %s con V/F. Mientras que %.2f sirve para formatear el número decimal 0.490989 redondeando a dos decimales 0.49.

Para cadenas que tienen múltiples líneas se ocupa tres comillas (‘’’ ‘’’). Este tipo de formato se ocupa si se requiere imprimir una cadena de texto en múltiples líneas en la consola.

Es importante saber manejar las cadenas de texto para mostrar de una forma adecuada los resultados obtenidos de un programa.

Consola

```
In [6]: texto = ''' Este
   ...: texto tiene
   ...: multiples
   ...: lineas '''

In [7]: texto
Out[7]: ' Este\ntexto tiene \nmultiples \nlineas '
```

Colecciones de Datos

Tuplas

Las tuplas son una secuencia ordenada de cero o más objetos. La característica principal de las tuplas es que son inmutables, esto quiere decir que no se puede remplazar o eliminar ninguno de sus elementos. La sintaxis para asignar una nueva tupla es la siguiente:

```
tupla = (elemento 0, elemento 1, … , elemento n-1 )
```

Ejemplo

Consola

```
In [51]: tupla = ('Juan' , 'Carlos', 'Alberto')

In [52]: tupla
Out[52]: ('Juan', 'Carlos', 'Alberto')
```

Nota: Si se quiere acceder a un elemento de una colección de datos se ocupan los corchetes. Por ejemplo si se quiere acceder a 'Juan' en la tupla es necesario escribir *tupla[0]*. Los índices en Python empiezan con el índice cero.

Las tuplas se ocuparon en la siguiente en el archivo metodosTesis.py en la siguiente línea de código:

```
DicionarioQuadraturasGauss = {
2 : ((1.0000 , 1.0000) , (-0.5773502691896257 , 0.5773502691896257))}
```

En esta línea se ocupan las tuplas por facilidad. En el método de cuadraturas gaussianas se tienen pares de datos estos pares de datos son fijos por lo tanto no

cambian a lo largo del programa. Por esta razón se ocuparon las tuplas para representar los coeficientes de cuadraturas gaussianas.

Listas

Las listas al igual que las tuplas son una secuencia ordenada de objetos. A diferencia de las tuplas las listas son mutables y tienen métodos que las modifican La sintaxis para crear una nueva lista es la siguiente:

```
lista = [elemento 0, elemento 1, … , elemento n-1]
```

Ejemplo

Consola

```
In [53]: lista = ['Juan' , 'Carlos', 'Alberto']

In [54]: lista
Out[54]: ['Juan', 'Carlos', 'Alberto']
```

Las listas se ocuparon en metodosTesis.py para representar las matrices. Se ocupan listas por que las matrices son mutables. Por ejemplo en la función *Gauss (matrix, vector)* está la siguiente línea de código:

```
matrix[i] , matrix[j] = matrix[j] , matrix[i]
```

En esta línea de código se cambian los renglones de las matrices por lo tanto se ocupan dos conceptos. El primero concepto se muestra la mutabilidad de las listas debido a que se están cambiando los elementos de las listas. El segundo concepto es una de las razones por las que se escogió Python para esta tesis que es la asignación múltiple, en una línea de código se realizan dos asignaciones.

Diccionarios

Los diccionarios son colecciones de datos que relacionan una clave y un valor. Las claves en el diccionario necesitan ser únicas y ser elementos inmutables, mientras que los valores asociados pueden ser de cualquier tipo. La sintaxis para crear un nuevo diccionario es la siguiente:

```
Diccionario = {clave 0: valor 0, clave 1: valor 1,…, clave n-1: valor n-1}
```

Ejemplo

Editor

```
1 Diccionario = {'Hidrogeno' : [1 ,1.0008]  ,
2 'Helio' : [2 , 4.003] ,'Litio' : [3 , 6.941] ,'Berilio' : [4 , 9.012]
3 }
```

En este ejemplo se ilustra cómo declarar un nuevo diccionario. Las claves en este ejemplo son los elemento de la tabla periódica mientras que el valor asociado a la clave es una lista de dos valores primer valor de la lista es el número atómico mientras que el segundo valor es el peso atómico.

El concepto de diccionario en metodosTesis.py se ocupó en la siguiente línea:

```
DicionarioQuadraturasGauss = {
2 : ((1.0000 , 1.0000) , (-0.5773502691896257 , 0.5773502691896257))}
```

Se ocupó un diccionario para representar los valores de cada aproximación esto ayuda de mucho debido a que la clave del diccionario es el grado con el que se quiere aproximar mientras que el valor asociado a la clave son el conjunto de coeficientes de Gauss. Los diccionarios en Python se ocupan para representar bases de datos.

Indentación

Un concepto importante en Python es el uso de la indentación. En Python la cantidad de espacios en blanco es la forma de separar los bloques de código. En Python el comando que crea una nueva indentación es (dos puntos :).La Ilustración 2 Indentación en Python muestra la separación de código mediante la cantidad de espacios en blanco.

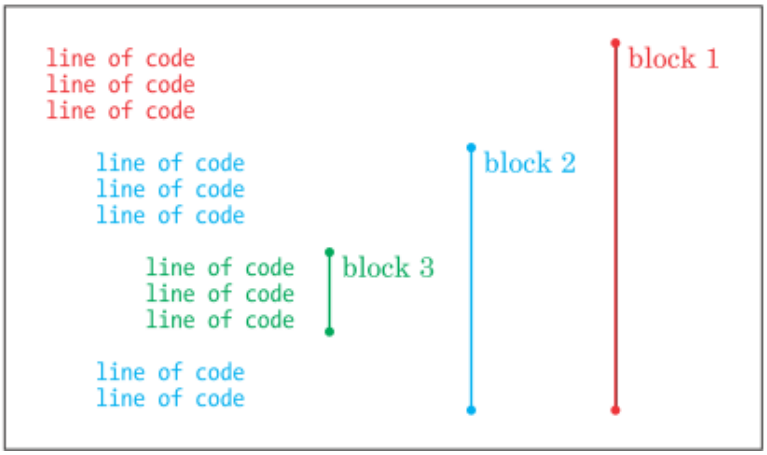

Ilustración 2 Indentación en Python (Briggs, 2013)

Controles de flujo

Condicional if

La forma de tomar una decisión en un programa es con un condicional *if*. Sólo si la condición es verdadera se ejecutará el código del bloque. La sintaxis de la cláusula *if* se muestra a continuación:

```
if <condición>:
        bloque
```

La Tabla 7 Condiciones más comunes en Python muestra las comparaciones más usadas en Python.

Tabla 7 Condiciones más comunes en Python

Operador	Nombre
= =	Igualdad
! =	Desigualdad
<	Menor que
>	Mayor que
< =	Menor o igual que
> =	Mayor o igual que

Fuente (González, s.f)

Ejemplo

Consola

```
In [61]: T = 10

In [62]: if T < 15:
    ...:     print 'Hoy es un dia frio'
    ...:
Hoy es un dia frio
```

En ejemplo anterior se puede ver que el valor de la condición es verdadero, por lo tanto se ejecutó el código del bloque if.

Los condicionales se ocuparon en el archivo metodosTesis.py para realizar decisiones un ejemplo de ello se muestra en la función *integracionNumerica (x , fx)* se tienen las siguientes líneas de código:

```
if len(x) == 2:
   return reglaTrapezoidal(x , fx)
if len(x) >= 3 and (len(x) - 1) % 2 == 0:
   return simpsonUnTercio (x , fx)
if len(x) >= 3 and (len(x) - 1) % 3 == 0:
   return simpsonTresOctavos (x , fx)
if len(x) > 4 and not (len(x) - 1) % 3 ==0 and not (len(x) - 1) % 2 == 0 :
   return  simpsonMixto (x , fx) fx)
```

En este ejemplo dependiendo de la cantidad de puntos que se conozcan, la función calculará el valor de la integral con su respectiva regla. El programa realiza la toma de decisión debido al condicional *if*.

Condicional if / elif / else

Se pueden tener más condicionales si se ocupa la estructura *if / elif / else*. En este tipo de cláusulas se ejecuta el código donde la condición sea verdadera, si ninguna de las condiciones anteriores es verdadera se ejecuta el código de la cláusula else.

```
if <condicion1>:
      bloque1
elif <condicion2>:
      bloque2
```

```
else:
      bloque3
```

```
In [63]: T = 16

In [64]: if T < 15:
    ...:     print 'Hoy es un dia frio'
    ...: elif T > 30:
    ...:     print 'Hoy es un dia caliente'
    ...: else:
    ...:     print 'Hoy es un dia templado'
    ...:
Hoy es un dia templado
```

En el ejemplo anterior se ejecutó el código de la cláusula *else* por que ninguna de la clausulas anteriores es verdadera.

Estos condicionales aumentan la cantidad de decisiones que puede realizar el programa sin embargo en el archivo metodosTesis.py no se ocupó este tipo de estructuras.

Ciclos

Los ciclos se utilizan para ejecutar fragmentos de códigos de forma repetida, hasta que se cumplen ciertas condiciones.

Los dos principales tipos de ciclos en Python son:

- *while*
- *for*

Ciclo while

Los ciclos *while* se ejecutan mientras se cumpla la condición del ciclo. La sintaxis se muestra a continuación:

```
while <condición>:
   bloque
```

Ejemplo

Consola

```
In [5]: edad = 15

In [6]: while edad < 18:
   ...:     print 'La edad es ' + str(edad) + ' es menor de edad'
   ...:     edad = edad + 1
   ...: print 'La edad es ' + str(edad) + ' es mayor de edad'
   ...:
La edad es 15 es menor de edad
La edad es 16 es menor de edad
La edad es 17 es menor de edad
La edad es 18 es mayor de edad
```

En este ejemplo muestra que mientras la variable edad < 18 se ejecuta el código del bucle *while*, una vez que la variable edad sea igual 18 se ejecuta la línea de código externo al ciclo *while* debido a que la condición es falsa.

Los ciclos while se ocuparon en metodosTesis.py se ocuparon en la función *Newton (funcion , estimadoinicial)* en la siguiente línea:

```
while i < imax:
```

En esta línea mientras que contador *i* sea menor que la cantidad máxima de iteraciones se ejecutará el código dentro del ciclo haciendo el paso de iteración del método de Newton. Una vez que la cantidad de iteraciones sea mayor se ejecutará el código fuera del ciclo. También es importante destacar las siguientes líneas de código dentro la misma función.

```
if tol < 0.001:
   condition = True
   break
```

En Python el comando *break* hace que los ciclos se paren. Esto significa, que se deja de ejecutar los comandos dentro del ciclo y se empezarán a ejecutar las líneas de código afuera del ciclo. En el método de Newton cuando se cumple la tolerancia ya no es necesario ejecutar el código dentro del ciclo por eso se ocupó un comando *break*.

Ciclo for

El ciclo *for* se ocupa para iterar sobre una secuencia. La sintaxis es la siguiente.

```
for <identificador>  in <secuencia>:
   bloque
```

Ejemplo

Consola

```
In [67]: for i in range(5):
    ...:     print i
    ...:
0
1
2
3
4
```

En este ejemplo se ejecuta el código hasta que la secuencia haya acabado.

Funciones

Una función es un fragmento de código con un nombre asociado que realiza una serie de tareas. Las funciones sirven para dividir el programar y reutilizar el código. La sintaxis para escribir una función en Python es la siguiente:

```
def < nombre de la función > (argumento 0, arg 1,…., arg n):
    bloque
```

Ejemplo

Editor

```
1 def cuadradoDeUnNumero (numero):
2     return numero ** 2
3 print cuadradoDeUnNumero(2)
4 print cuadradoDeUnNumero(4)
```

Consola

```
In [1]: %run "c:\users\dany\appdata\local\temp\tmpcmgcod.py"
4
16
```

Este ejemplo ilustra la utilidad de las funciones al ocupar una función definida para realizar dos la operación de elevar al cuadrado.

Los métodos se programaron como funciones un concepto importante que se destaca es el uso de funciones de alto orden. Esto es debido a que los métodos requieren funciones como argumentos.

CAPÍTULO III

En este capítulo se explica cómo se programaron los métodos numéricos, además de proporcionar ejemplos ilustrando como se ocupan estos métodos. La mayoría de los algoritmos se obtuvieron del libro Numerical Analysis (Burden & Faires, 2005).

Los ejemplos están programados en el archivo ejemplo.py es muy importante que el archivo este en la misma carpeta en la computadora que el archivo metodosTesis.py debido a que importa los métodos programados.

Para encontrar la ayuda sobre el uso archivo metodosTesis.py es necesario importar el archivo metodosTesis.py en un nuevo archivo de Python y escribir el comando *help*(metodosTesis) en la consola para obtener información de la funciones programadas en metodosTesis.py

Ecuación algebraica de una variable

Método de Newton-Raphson

El método de Newton (también conocido como Newton-Raphson) es uno de los métodos más poderosos y más conocidos para el problema de encontrar raíces. Su convergencia es cuadrática; sin embargo, para que funcione adecuadamente se tienen que proporcionar buenos estimados iniciales.

Para encontrar el algoritmo del método de Newton se tiene que proporcionar un estimado inicial $x^{[0]}$ que se acerca al verdadero valor de x_s para el cual $f(x_s)$ = 0. Ocupamos las series de Taylor para aproximar el valor de $f(x)$ en el valor de $x^{[0]}$.

$$f(x) = f\left(x^{[0]}\right) + \frac{df}{dx}\left(x - x^{[0]}\right) + \frac{1}{2!}\frac{d^2f}{dx^2}\left(x - x^{[0]}\right)^2 + \ldots.$$

En donde las derivadas están evaluadas en $x^{[0]}$. Si el valor de $f(x) = 0$ entonces:

$$0 = f\left(x^{[0]}\right) + \frac{df}{dx}\left(x - x^{[0]}\right) + \frac{1}{2!}\frac{d^2f}{dx^2}\left(x - x^{[0]}\right)^2 + \ldots.$$

Si $x^{[0]}$ está cerca de la solución x_s:

$$\left|\left(x - x^{[0]}\right)\right| \gg \left|\left(x - x^{[0]}\right)\right|^2 \gg \left|\left(x - x^{[0]}\right)\right|^3$$

Por lo tanto la solución se aproxima mediante:

$$0 = f\left(x^{[0]}\right) + \frac{df}{dx}\left(x - x^{[0]}\right)$$

Despejando x y realizando este paso de forma sucesiva, se obtiene el método de Newton para una ecuación no lineal de una variable.

$$x^{[k+1]} = x^{[k]} - \frac{f(x^{[k]})}{f^{(1)}(x^{[k]})}$$

Algoritmo Newton-Raphson

Datos de entrada: aproximación inicial, tolerancia, número máximo de iteraciones.

Datos de salida: solución aproximada de solución o mensaje de error.

Paso 1: Establece la iteración $i = 0$

Paso 2: Mientras $i \leq imax$ (número máximo de iteraciones) realizar pasos 3 – 6

Paso 3: Calcular $x = x_0 - f(x_0)/f'(x_0)$

Paso 4: Si $|x - x_0| < Tol$

Resultado es x

Parar el ciclo

Paso 5: $i = i + 1$

Paso 6: $x_0 = x$

Paso 7: Devuelve el resultado

Código de Python método Newton-Raphson

```
def derivada( funcion , punto , delta = 0.0001):
    ''' Devuelve  el valor de la derivada en un punto
    Los argumentos necesarios son:
```

```
    funcion: La funcion a la cual se requiere sacar la derivada
    punto: valor en el que se quiere calcular la derivada
    Ejemplo:
        Primero se define la funcion
        def f(x): # Asi se define la funcion en este caso fx = x^2
            return x**2
        derivada (f , 1) ---> 2.0
    '''
    der = (funcion(punto*1.0 + delta*1.)-funcion(punto*1.0-
delta*1.0))/(2.0*delta*1.0)
    return der
#------------------------------------------------------------------------
def Newton ( funcion , estimadoinicial , imax = 20 , delta = 0.00001 , tol =
0.0001):
    ''' Devuelve la raiz cuando f = 0 con un estimado inicial
    Los argumentos necesarios son:
    funcion: es la funcion igualada a 0
    estimadoinicial: es el estimado inicial propuesto
    argumentos opcinales
    imax default 20 como tercer argumento
    delta default 0.00001
    tol por default 0.0001
    Ejemplo:
        Primero se define la funcion
        def f(x): # Asi se define la funcion en este caso fx = x^2 - 4 = 0
            return x**2 - 4
        Newton(f , 1) ---> 2
    '''
    i = 0
    xvieja =  estimadoinicial
    condition = True
    while   i < imax:
        der = derivada( funcion , xvieja , delta)
        xnueva = xvieja*1.0 - (funcion(xvieja*1.0))/(der)
        i = i + 1
        error = abs((xnueva*1.0-xvieja*1.0))
        if error < tol: # Se puede cambiar el valor de la tol
            condition = False
            break
        xvieja = xnueva
    if not condition:
         return xvieja
    else:
       print 'No se cumplio con la tolerancia '
```

El código está organizado de la siguiente forma:

Se puede observar que lo primero que contiene el código es la creación de una función derivada. Esta función lo que permite es calcular la derivada numérica de una función. La aproximación de las derivadas que se ocupó a lo largo de esta tesis fue la aproximación denominada diferencias finitas centradas.

Ejemplo método Newton -Raphson

Se desea resolver $f(x) = cosx - x = 0$ con un estimado inicial de 0.78

Editor

```
89 print 'La funcion que se quiere resolver es f(x) = cosx - x = 0'
90 print ''
91 def f(x):
92             return math.cos(x)-x
93 x = Newton(f , .78)
94 print ''
95 print ' La solucion de x es: ' , x
```

Consola

```
La funcion que se quiere resolver es f(x) = cosx - x = 0

El estimado inicial es:  0.78

| i | valor(i)   |  valor(i+1) |   error     |
-------------------------------------------
| 0 |   ND       |   0.7800    |   ND        |
| 1 |   0.7800   |   0.7394    |   0.040561  |
| 2 |   0.7394   |   0.7391    |   0.000354  |
| 3 |   0.7391   |   0.7391    |   0.000000  |

 La solucion de x es:  0.739085160878
```

En este ejemplo se modificó el código de Python para que imprimiera en cada iteración los valores de x. Se puede observar que se llama a la función Newton asignando el valor de x. Al final se imprime el valor de x siendo el resultado 0.739085 el valor es igual al reportado (Burden & Faires, 2005).

Sistemas de ecuaciones algebraicas lineales

Método Gauss - Jordan

El método de Gauss Jordan resuelve sistemas de ecuaciones algebraicas lineales y es considerado un método de eliminación. Estos métodos de eliminación son fáciles de implementar y generalmente resuelven cualquier sistema que tenga una solución. El punto negativo es que ocupan demasiados recursos si el sistema a resolver consiste en un número elevado de ecuaciones.

La estrategia de este método es realizar una secuencia de operaciones que convierte el sistema original en uno que se pueda resolver fácilmente. Esto se logra convirtiendo la matriz de coeficientes en la matriz identidad con una serie de operaciones.

Algoritmo Gauss – Jordan

Paso 1: Agregar la matriz de constantes a la matriz de coeficientes obteniendo la matriz aumentada.

Paso 2: Localizar la columna de la izquierda que tenga al menos un elemento diferente a cero.

Paso 3: Si el elemento del primer renglón, en la columna del paso 1, es un cero, se debe de intercambiar el renglón superior por otro.

Paso 4: Si el elemento de primer reglón, en la columna del paso 1, no es un uno, dividir el renglón por este elemento; de esta manera se obtiene un uno principal.

Paso 5: Sumar múltiplos adecuados del renglón superior a los renglones inferiores para que todos los elementos debajo del ‘uno’ , obtenido en el paso 3 se hagan ceros.

Paso 6: Repetir los pasos 3 y 4 para los renglones posteriores en el paso 4 sumar los múltiplos adecuado en los renglones superiores y los renglones inferiores.

Código de Python método Gauss - Jordan

```
def Gauss (matrix , vector):
    ''' Devuelve el vector incognita del sistema de matrix de coeficientes
    vector de solucion
    Los argumentos necesarios son:
    matrix: Es la matrix de coeficientes del sistema
    vector: es el vector solucion
    Ejemplo Gauss ([[1 ,1] , [3,5]] , [[3] , [5]]) ------>  [[5.0], [-2.0]]
    '''
    def ordenarmatrix(matrix , vector):
        for i in range(len(matrix)):
            pivote = matrix[i][i]
            if pivote == 0:
                for j in range(i + 1 , len(matrix)):
                    nuevo = matrix[j][i]
                    if nuevo != 0 :
                        matrix[i] , matrix[j] = matrix[j]  , matrix[i]
                        vector[i] , vector[j] = vector[j]  , vector[i]
                        pass
        return matrix , vector
    copia = copy.deepcopy(matrix)
    copiav = copy.deepcopy(vector)
    copia , copiav =  ordenarmatrix(copia , copiav)
    for i in range(len(copia)):
        pivote = copia[i][i]
        for j in range(len(matrix)):
            copia[i][j] = copia[i][j]/(pivote * 1.0)
            if i == j:
                copiav[i][0] = copiav[i][0]/(pivote * 1.0 )
        for k in range(len(copia)):
            if k == i:
                pass
            else:
                current = copia[k][i] * 1.0
                for l in range(len(matrix)):
                    copia[k][l] = copia[k][l] - copia[i][l] * current
                    if k == l:
                        copiav[k][0] = copiav[k][0] - copiav[i][0] * current
    return copiav
```

Ejemplo método Gauss –Jordan

En la notación de esta tesis para crear una matriz se ocuparán los corchetes ([]), como puede verse en la Tabla 8 Ejemplos de matriz.

Tabla 8 Ejemplos de matriz

Matriz (Renglón | Columna)	Sintaxis Python	Matriz
1 X 2	[[1 ,2]]	$\begin{matrix}1 & 2\end{matrix}$
2 X 1	[[1] , [2]]	$\begin{matrix}1\\2\end{matrix}$
2 X 3	[[1,2,3] , [4,5,6]]	$\begin{matrix}1 & 2 & 3\\4 & 5 & 6\end{matrix}$
3 X 2	[[1,2],[3,4],[5,6]]	$\begin{matrix}1 & 2\\3 & 4\\5 & 6\end{matrix}$
3 X 3	[[1,2,3] , [4,5,6] , [7,8,9]]	$\begin{matrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{matrix}$

Los argumentos necesarios para realizar el método de eliminación Gauss – Jordan son una matriz con los coeficientes, y un vector de resultados. Al escribir estos dos argumentos se puede ejecutar el método de Gauss – Jordan.

Ejemplo, se desea resolver el siguiente sistema de ecuaciones

$$\begin{matrix}x & y & z\\2x & 3y & 5z\\4x & 0y & 5z\end{matrix} = \begin{matrix}5\\8\\2\end{matrix}$$

Editor

```
51 coeficientes = [[1 ,1 ,1] , [2,3,5] , [4,0,5]]
52 aumentada = [[5], [8] , [2]]
53 variables = ['x' , 'y' , 'z']
54 x = Gauss (coeficientes , aumentada)
55 print 'La solucion del sistema es el vector ' , x
```

Nota: La asignación “variables” no es necesaria sólo se agregó como ayuda visual.

Consola

```
[1, 1, 1]x = [5]
[2, 3, 5]y = [8]
[4, 0, 5]z = [2]

[1.0, 1.0, 1.0]x = [5.0]
[0.0, 1.0, 3.0]y = [-2.0]
[0.0, -4.0, 1.0]z = [-18.0]

[1.0, 0.0, -2.0]x = [7.0]
[0.0, 1.0, 3.0]y = [-2.0]
[0.0, 0.0, 13.0]z = [-26.0]

[1.0, 0.0, 0.0]x = [3.0]
[0.0, 1.0, 0.0]y = [4.0]
[0.0, 0.0, 1.0]z = [-2.0]

La solucion del sistema es el vector  [[3.0], [4.0], [-2.0]]
```

Se puede observar la reducción del sistema de ecuaciones a la matriz identidad, resolviendo el sistema de ecuaciones encontrado así, la solución del sistema.

Sistemas de ecuaciones algebraicas no lineales

Método de Newton multivariable

El método de Newton multivariable sirve para resolver sistemas de ecuaciones no lineales manteniendo una convergencia cuadrática. Para que funcione adecuadamente se tienen que proporcionar buenos estimados iniciales para lograr la convergencia.

Nuevamente se ocupan series de Taylor para aproximar las funciones y así obtener el método de Newton multivariable.

$$f_i(x) = f_i\left(x^{[k]}\right) + \sum_{m=1}^{N} \frac{\partial f_i}{\partial x_m}\left(x_m - {x_m}^{[k]}\right) + \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\left(x_m - {x_m}^{[k]}\right)\frac{\partial^2 f_i}{\partial x_m\,\partial x_n}\left(x_m - {x_m}^{[k]}\right)$$

Al igual que en el método de Newton para una ecuación, se puede aproximar la solución mediante:

$$f_i(x_s) = 0 \approx f_i\left(x^{[k]}\right) + \sum_{m=1}^{N} \frac{\partial f_i}{\partial x_m} \left(x_m - x_m{}^{[k]}\right)$$

El jacobiano que es un arreglo de derivadas parciales, tiene definidos sus elementos de la siguiente forma:

$$J_{im}{}^{[k]} = \frac{\partial f_i}{\partial x_m}$$

Se escribe el método de Newton multivariable como la resolución de un sistema de ecuaciones algebraicas:

$$J^{[k]}\Delta x^{[k]} = -f(x^{[k]})$$

En donde:

$$\Delta x^{[k]} = x^{[k+1]} - x^{[k]}$$

Algoritmo Método de Newton

Datos de entrada: Número de ecuaciones, estimados iniciales, tolerancia y máximo número de iteraciones.

Datos de salida: Solución aproximada o mensaje de error.

Paso 1: Establece la iteración $i = 1$

Paso 2: Mientras $k \leq imax$ Realizar pasos del 3- 7

Paso 3: Calcular $F(x)$ y $Jacobiano(x)_{i,j}$ Para $1 \leq i, j \leq n$ $n\ es\ el\ número\ de\ incógnitas$

Paso 4: Resolver el sistema de ecuaciones algebraico lineal $J(x)y = -F(x)$ y es la solución del sistema.

Paso 5: $x\ nueva = x\ vieja + y$

Paso 6: Si el $error < Tol$ La función error

$error = \sum_{i=0}^{n} abs(xnueva[i] - xvieja[i])$

$El\ resultado\ es\ xnueva$ Parar el ciclo

Paso 7: $xnueva = xvieja$

Paso 8: $k = k + 1$

Paso 9 Devuelve el resultado

Código de Python método Newton

```
def evalaucionF (puntos , funciones ):
      copia =  copy.deepcopy(puntos)
      rm1 , cm1   =  len(copia) , len(copia[0])
      listacolumna = []
      for r in range (len(copia[0])):
         columna = []
         for c2 in range(len(copia)):
            columna.append(copia[c2][r])
         listacolumna.append(columna)
      listacolumna = listacolumna[0]
      matrix = []
      for r in range (rm1):
         matrix.append([])
         for c in range(cm1):
            matrix[r].append(-1.0 * funciones[r](listacolumna) )
      return matrix
def norma(matrixA  , matrixB):
   rm1 , rm2 , cm1 , cm2  =  len(matrixA) , len(matrixB) , len(matrixA[0]) ,
len(matrixB[0])
   assert rm1 == rm2 and cm1 == cm2 , 'Las matrices deben ser iguales en
reglones y columnas'
    # Crea la matrix 0 de la suma
   norma = 0
   for r in range (rm1):
       for c in range(cm1):
          norma = norma + abs( matrixA[r][c] - matrixB[r][c] )
   return norma
def jacobiano (funciones , puntos , delta = 0.0001 ):
   ''' Se requiere una lista de funciones y una lista de puntos
   '''
   assert len(funciones) == len(puntos) , 'La listade funciones y la lista de puntos
debe de coresponder'
```

```
    matrix = []
    n = len(funciones)
    # Cambiar renglones por columnas de puntos me va a facilitar despues
    # Crea la matrix de ceros
    copia = copy.deepcopy(puntos)
    listacolumna = []
    for r in range (len(copia[0])):
        columna = []
        for c2 in range(len(copia)):
            columna.append(copia[c2][r])
        listacolumna.append(columna)
    listacolumna = listacolumna[0]
    for i in range (len(funciones)):
        matrix.append([])
        for j in range(n):
                matrix[i].append(0)
    for i in range (len(funciones)):
        for j in range(n):
                copiaAde = copy.deepcopy(listacolumna)
                copiaAtr = copy.deepcopy(listacolumna)
                copiaAde[j] = copiaAde[j] + delta
                copiaAtr[j] = copiaAtr[j] - delta
                denominador = 2 * delta * 1.0
                parcial  =  (funciones[i](copiaAde) - funciones[i](copiaAtr)  )/
denominador
                matrix[i][j] =  parcial
    return matrix
def newtonSistemas (funciones , puntos , imax = 10 , tol = 0.0001 , delta =
0.0001 ):
    ''' Devuelve un vector con la solucion  los argumentos necesarios son
          una lista de funciones una lista de
          puntos argumentos opcionales imax por default 10
          la tol por default 0,00001
          y el delta por default 0.0001'''
    xvieja = copy.deepcopy(puntos)
    i = 0
    condition = True
    while i < imax:
        jaco = jacobiano (funciones , xvieja , delta = 0.0001 )
        fx = evalaucionF (xvieja  , funciones)
        y =  Gauss ( jaco , fx)
        x = sumaMatrices (xvieja , y)
        if norma(xvieja , x) < tol:
            condition = False
            break
        xvieja = x
        i = i + 1
```

```
if not condition:
    return xvieja
else:
  print 'No se cumplio con la tolerancia '
```

Ejemplo método de Newton

Se desea resolver el siguiente sistema.

$$3\,x_1 - \cos x_2 x_3 - \frac{1}{2} = 0$$

$$x_1{}^2 - (x_2 + 0.1)^2 + sinx_3 + 1.06 = 0$$

$$e^{-x_1 x_2} + 20\,x_3 + \frac{10\,\pi - 3}{3} = 0$$

Con los siguientes estimados iniciales.

$$x^0 = \begin{matrix} 0.1 \\ 0.1 \\ -0.1 \end{matrix}$$

Se puede observar que para llamar al método Newton Sistemas se requieren dos argumentos. El primer argumento es la lista de funciones que se requieren resolver en este caso son las siguientes tres funciones:

$$f_1(x_1\,, x_2, x_3) = 3\,x_1 - \cos x_2 x_3 - \frac{1}{2} = 0$$

$$f_2(x_1\,, x_2, x_3) = x_1{}^2 - (x_2 + 0.1)^2 + sinx_3 + 1.06 = 0$$

$$f_3(x_1\,, x_2, x_3) = e^{-x_1 x_2} + 20\,x_3 + \frac{10\,\pi - 3}{3} = 0$$

Es muy importante que las funciones estén igualadas a cero si las funciones no están igualadas a cero no se encontrará la solución correcta debido a la deducción del método $0 \approx f_i\left(x^{[k]}\right) + \sum_{m=1}^{N} \frac{\partial f_i}{\partial x_m}\left(x_m - x_m{}^{[k]}\right)$.

Se puede observar que para resolver este tipo de sistemas, las funciones dependerán de varias variables por lo tanto para poder extender a *n* ecuaciones, los

argumentos se considerarán una lista de variables, por lo que se le asigna un índice a cada variable que se presentan a continuación.

$$x_1 = lista[0]$$

$$x_2 = lista[1]$$

$$x_3 = lista[2]$$

Las funciones quedan entonces de la siguiente forma.

$$f_1(lista) = 3\ lista[0] - \cos(\ lista[1]lista[2]) - \frac{1}{2} = 0$$

$$f_2(lista) = lista[0]^2 - \ (lista[1] + 0.1)^2 + \sin(lista[2]) + 1.06 = 0$$

$$f_3(lista) = e^{-lista[0]lista[1]} + 20\ lista[2] + \frac{10\ \pi - 3}{3} = 0$$

Es muy importante considerar que el índice empieza en cero esto es debido a que los índices en las listas de Python comienzan en cero.

Por último la lista de funciones se escribe de la siguiente forma respetando la sintaxis de Python.

$$lista = [\ f_1\ , f_2\ , f_3]$$

El segundo argumento es un vector con los estimados iniciales. Ocupando la sintaxis previamente explicada (ver el vector *puntos* del siguiente ejemplo).

Editor

```
def f1(lista):
    return 3* lista[0] - math.cos(lista[1] * lista[2]) - 0.5

def f2(lista):
    return  lista[0] ** 2 - 81 * (lista[1] + 0.1 )**2 + math.sin(lista[2]) + 1.06

def f3(lista):
    return  math.e ** (-lista[0] * lista[1]) + 20 * lista[2] + (10 * math.pi - 3)/3.0

puntos = [[0.1] ,[ 0.1] ,[-0.1]]
listaFunciones = [f1 , f2 , f3]
```

```
print 'Las funciones que se quieren resolver son: '
print ''
print 'f1(x1 , x2 , x3) = 3x1 - cos(x2 x3) - 1/2'
print 'f2(x1 , x2 , x3) = x1 ** 2 - 81(x2 + 0.1)**2 + sin x3 + 1.06'
print 'f3(x1 , x2 , x3) = e ** -x1x2 + 20 x3 + (10 * pi -3) /3 '
print ''

x = newtonSistemas (listaFunciones , puntos)
print ''
print 'La solucion es el vector: '
print ''
print x
```

Consola

```
Las funciones que se quieren resolver son:

f1(x1 , x2 , x3) = 3x1 - cos(x2 x3) - 1/2
f2(x1 , x2 , x3) = x1 ** 2 - 81(x2 + 0.1)**2 + sin x3 + 1.06
f3(x1 , x2 , x3) = e ** -x1x2 + 20 x3 + (10 * pi -3) /3

Los estimados inicales son: [[0.1], [0.1], [-0.1]]

| i |   x1 (i)     |   x2 (i)    |   x3 (i)    |   error    |
-------------------------------------------------------------
| 0 |   0.100000   |   0.100000  | -0.100000   | ND         |
| 1 |   0.499870   |   0.019467  | -0.521520   | 0.901923   |
| 2 |   0.500014   |   0.001589  | -0.523557   | 0.020059   |
| 3 |   0.500000   |   0.000012  | -0.523598   | 0.001632   |
| 4 |   0.500000   |   0.000000  | -0.523599   | 0.000013   |

La solucion es el vector:

[[0.5000001134678346], [1.2444783376805608e-05], [-0.5235984500729046]]
```

Integración numérica

Evaluación numérica de integrales

Estos métodos surgen ante la necesidad de calcular la integral de una función que no tengan una antiderivada explícita o que la antiderivada no sea fácil de obtener. La idea básica de estos métodos es aproximar el valor de la integral $\int_a^b f(x)dx$ usando un polinomio de la forma $\sum_{i=0}^{n} a_i f(x_i)$. Se necesitan seleccionar un conjunto de nodos $\{x_0 , \ldots . , x_n\}$ del intervalo $[a , b]$ para después integrarlos con el polinomio de Lagrange.

Al tener un mayor conjunto de nodos en el intervalo de la integral el grado del polinomio con el que se interpola es mayor teniendo como resultado un menor error

en la aproximación de la integral, siempre y cuando el orden del polinomio no sea excesivamente alto.

Algoritmo de Evaluación numérica de integrales

Paso 1: Ver cuántos pares de puntos $x\,, f(x)$ se conocen

Paso 2: Si el número de puntos $x\,, f(x)$ son dos se ocupa la regla trapezoidal $\int_{x0}^{x1} f(x)dx = \frac{h}{2}[f(x_0) + f(x_1)]$

Paso 3: Si el número de puntos es mayor a dos

Paso 4: Si el número de puntos menos uno es divisible entre dos se ocupa la regla de Simpson un tercio

$$\int_{X_O}^{X_N} f(x)dx = \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + 4f_{N-1} + f_N]$$

Paso 5: Si el número de puntos menos uno es divisible entre tres se ocupa la regla de Simpson tres octavos

$$\int_{x_0}^{X_N} f(x)dx = \frac{3}{8}h\,[f_0 + 3f_1 + 3f_2 + 2f_3 + 3f_4 + 3f_5 + 2f_6 + \cdots + f_{N-1} + f_N]$$

Paso 6: Si no se cumple el Paso 4 y el Paso 5 se divide la integral en dos intervalos para que una parte sea tres octavos y la otra parte un tercio (K.G., 2004) para conservar el error en todo el cálculo de la integral.

Código de Python de evaluación numérica de integrales

```
def integracionNumerica (x , fx ):
    ''' Devuelve el valor de la integral numerica
    se requiere de dos argumentos
    x: una lista de los valores independientes
    fx: una lista con los los valores de x evaluados en f
    ejemplo integracionNumerica ([1,2,3,4] , [1 ,4,9,16] ) ---> 21.0'''
    def reglaTrapezoidal (x , fx):
        h = (x[len(x)-1]*1.0 - x[0]*1.0)/((len(x)-1)*1.0)
        return h/ 2.0 * (fx[0] + fx[1])
    def simpsonUnTercio (x , fx):
```

```
    integral = 0
    h = (x[len(x)-1]*1.0 - x[0]*1.0)/((len(x)-1)*1.0)
    lista = []
    for i in range(len(x)):
            if i == 0 or i == (len(x)-1):
                coef = 1
                integral = integral + fx[i] * coef
                lista.append(coef)
            if i % 2 == 0 and not i == 0 and not i == (len(x)-1)  :
                coef = 2
                lista.append(coef)
                integral = integral + fx[i] * coef
            if i % 2 == 1:
                coef = 4
                lista.append(coef)
                integral = integral + fx[i] * coef
    return h / 3.0 * integral
def simpsonTresOctavos (x , fx):
    integral = 0
    h = (x[len(x)-1]*1.0 - x[0]*1.0)/((len(x)-1)*1.0)
    lista = []
    for i in range(len(x)):
            if i == 0 or i == (len(x)-1):
                coef = 1
                integral = integral + fx[i] * coef
                lista.append(coef)
            if i % 3 == 0 and  i != 0 and i != (len(x)-1)  :
                coef = 2
                lista.append(coef)
                integral = integral + fx[i] * coef
            if i % 3  != 0 and  i != 0 and i != (len(x)-1) :
                coef = 3
                lista.append(coef)
                integral = integral + fx[i] * coef
    return 3.0*h / 8.0 * integral
def simpsonMixto (x , fx):
        split = len(x)/2
        x0 , fx0 = x[0 : split + 1] , fx[0 :split + 1]
        x1 , fx1 = x[split : len(x) ] ,  fx[split : len(x) ]
        return integracionNumerica (x0 , fx0 ) +  integracionNumerica (x1 , fx1)
if len(x) == 2:
    return reglaTrapezoidal(x , fx)
if len(x) >= 3 and (len(x) - 1) % 2 == 0:
    return simpsonUnTercio (x , fx)
if len(x) >= 3 and (len(x) - 1) % 3 == 0:
    return simpsonTresOctavos (x , fx)
if len(x) > 4 and not (len(x) - 1) % 3 ==0 and not (len(x) - 1) % 2 == 0 :
```

```
return  simpsonMixto (x , fx)
```

Ejemplo de evaluación numérica de integrales

El ejemplo se obtuvo de la siguiente fuente (K.G., 2004). En este ejemplo se desea resolver la integral desde 1.0 hasta 1.7

Tabla 9 Tabla de valores problema evaluación numérica de integrales

x	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7
f(x)	1.543	1.669	1.811	1.971	2.151	2.352	2.577	2.828

Fuente (K.G., 2004)

Editor

```
print 'La lista con valores de x = [1.0 , 1.1 ,1.2 , 1.3, 1.4 , 1.5, 1.6, 1.7]'
print 'Lista con valores de f(x) = [1.543 , 1.669 , 1.811 , 1.971 , 2.151 , 2.352 , 2.577 , 2.828]'
x = [1.0 , 1.1 ,1.2 , 1.3, 1.4 , 1.5, 1.6, 1.7]
fx = [1.543 , 1.669 , 1.811 , 1.971 , 2.151 , 2.352 , 2.577 , 2.828]
integral = metodosTesis.integracionNumerica (x , fx)
print 'El valor de la integral es: ' , integral
```

Consola

```
Problema evaluacion numerica de integrales

La lista con valores de x = [1.0 , 1.1 ,1.2 , 1.3, 1.4 , 1.5, 1.6, 1.7]
Lista con valores de f(x) = [1.543 , 1.669 , 1.811 , 1.971 , 2.151 , 2.352 , 2.577 , 2.828]
El valor de la integral es:  1.470425
```

Regla de Simpson Compuesta

Esta es la regla más usada para la evaluación de una integral mediante cuadraturas. El error de esta regla es de tercer grado $O(h^4)$ lo que es mayor que las reglas de Simpson estándar; sin embargo es importante destacar que este método se aplica cuando se desea integrar una función conocida, no un conjunto de datos provenientes de un experimento, esta diferencia hace que los errores no sean comparables debido a que el espacio entre nodos $h = \frac{b-a}{n}$ puede ser ahora escogido en función de *n*. Lo que permite reducir el espacio entre nodos resultando en una mejor aproximación. Los nodos en esta regla son igualmente espaciados.

Algoritmo de regla de Simpson Compuesta

Aproximar la integral $I = \int_a^b f(x)dx$

Datos de entrada: Función f(x), ***a*** , ***b*** y ***n*** que es el número de intervalos que se quiere aproximar la solución

Paso 1: Calcular el tamaño de cada subintervalo: $h = \frac{b-a}{n}$

Paso 2: Calcular $XI0 = f(a) + f(b)$

$XI1 = 0$

$XI2 = 0$

Paso 3: Para $i = 1, \ldots, n-1$ Realizar Pasos 4 y 5

Paso 4: Calcular $X = a + ih$

Paso 5: Si i es par entonces $XI2 = XI2 + f(x)$

en caso contrario entonces: $XI1 = XI1 + f(x)$

Paso 6: Calcular $XI = h(XI0 + 2\,XI2 + 4XI1)/3$

Paso 7: Devolver XI

Código de Python de regla de Simpson Compuesta

```
def compositeSimpson (funcion , a  ,b , n):
    ''' Devuelve el valor de la integral numerica por el
    metodo simpson compuesto los argumentos
    necesarios son:
    funcion: Es la funcion de la integral
    a: valor inicial
    b: valor final
    n: es el numero de pasos
    Ejemplo:
        Primero se define la funcion
        def f(x): # Asi se define la funcion en este caso fx = sin(x)
            return math.sin(x)
        compositeSimpson (f , 0  , 2 , 6) ---> 1.416245265
    '''
```

```
    h = (b - a)/(n *1.0)
    XI0 = funcion(a) + funcion(b)
    XI1 = 0
    XI2 = 0
    XI = 0
    i = 1
    for j in range(n-1):
        X = a + i*h
        if i % 2 == 0:
            XI2 = XI2 + funcion(X)
        else:
            XI1 = XI1 + funcion(X)
        j = j + 1
        i = i +1
    return (h* (XI0 + 2*XI2+ 4*XI1))/3.0
```

Ejemplo de regla de Simpson Compuesta

Se desea resolver la siguiente integral:

$$\int_0^{\pi} \sin x \, dx$$

Editor

```
161 print 'Se desea resolver la integral sin(x)  desde 0 hasta pi'
162 print 'El orden del polinomio con el cual se quiere aproximar es 18'
163
164 def f(x):
165     return math.sin(x)
166
167 print 'El resultado de la integral es: ' , metodosTesis.compositeSimpson (f , 0  , math.pi , 18)
```

Consola

```
Problema regla de Simpson Compuesta

Se desea resolver la integral sin(x)  desde 0 hasta pi
El orden del polinomio con el cual se quiere aproximar es 18
El resultado de la integral es:  2.00001034771
```

Para tener obtener un resultado con un error de 10^{-5} es necesaria una n de 18.

Cuadraturas Gaussianas

En este método los puntos para evaluar la integral son escogidos de una manera óptima en lugar de ser espaciados igualmente. Los nodos $x_1, x_2, \ldots x_n$ en el

intervalo $[a\,,b]$ y los coeficientes $c_1\,,c_2\,,\ldots,c_n$ son escogidos para minimizar el error en la aproximación. Los coeficientes de este método están reportados en el libro de Numerical Analysis (Burden & Faires, 2005).La forma de aproximar la integral se presenta a continuación.

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} c_i\, f(x_i)$$

Algoritmo Cuadraturas Gaussianas

Paso 1: Seleccionar el número de puntos con el que se quiere aproximar la integral.

Paso 2: Realizar el cálculo de la integral el cual se realiza de la siguiente forma

$$I \cong c_1 f(x_1) + c_2 f(x_2) + \cdots + c_n f(x_n)$$

Código de Cuadraturas Gaussianas

Nota: Los valores del diccionario se obtuvieron de (Kammermans, 2011). Los coeficientes que se obtuvieron fueron para valores de $2 \leq n \leq 15$ donde n es el grado del polinomio con el que se aproxima.

```
def cuadraturasGausianas (f , a , b , n):
    ''' Devuelve el valor de la integral numerica por el
    metodo de cuadraturas gausianas los argumentos
    necesarios son:
    funcion: Es la funcion de la integral
    a: valor inicial
    b: valor final
    n: es el numero de pasos  el valor de n tiene que estar dentro 2 <= n <= 15
    Ejemplo:
        Primero se define la funcion
        def f(x): # Asi se define la funcion en este caso fx = sin(x)
            return math.sin(x)
        cuadraturasGausianas (f , 0  , 2 , 6) ---> 1.41614683655
    '''
    if a ==  -1 and b ==  1:
        suma = 0
        for i in range(n):
            suma = suma + DicionarioQuadraturasGauss[n][0][i] *
f(DicionarioQuadraturasGauss[n][1][i])
        return suma
```

```
    else:
        suma = 0
        for i in range(n):
            suma = suma + DicionarioQuadraturasGauss[n][0][i] *
f((DicionarioQuadraturasGauss[n][1][i] * (b-a) + (a+b)) / (2.0))
        return  (b - a)/2.0 * suma
```

Ejemplo de Cuadraturas Gaussianas

Se desea resolver la integral:

$$\int_{-1}^{1} e^x \cos(x)\, dx$$

En este ejemplo se resolverá con cuadraturas gaussianas aproximando el polinomio desde un polinomio grado dos hasta un polinomio grado quince.

Editor

```
print 'Se desea resolver la integral e ** x * cos x desde -1 hasta 1'
print ''
def f(x):
    return math.e ** x * math.cos(x)
lista = [2 , 3 ,4 ,5 ,6 ,7 ,8 ,9 ,10,11,12,13,14,15]
a = -1
b = 1
integral = []
for i in range(2 , 16):
    integral.append(metodosTesis.cuadraturasGausianas (f , a , b , i))
error = []
valorIntegral = 1.9333904
n = 2
for i in range(len(integral)):
    error.append(abs(integral[i]) - valorIntegral)
    print 'El valor de la integral con el polinomio grado %i es %.6f ' % (n , integral[i])
    n = n +1
print ''
plt.plot(lista , error)

plt.xlabel('n')
plt.ylabel('Error')
plt.suptitle('Error de la integral '+ r'$\int_{-1}^1 e^x\cosx $' + ' en funcion del grado', fontsize=11)
```

Consola

```
Problema Cuadraturas Gaussianas
Se desea resolver la integral e ** x * cos x desde -1 hasta 1

El valor de la integral con el polinomio grado 2 es 1.962973
El valor de la integral con el polinomio grado 3 es 1.933390
El valor de la integral con el polinomio grado 4 es 1.933417
El valor de la integral con el polinomio grado 5 es 1.933421
El valor de la integral con el polinomio grado 6 es 1.933421
El valor de la integral con el polinomio grado 7 es 1.933421
El valor de la integral con el polinomio grado 8 es 1.933421
El valor de la integral con el polinomio grado 9 es 1.933421
El valor de la integral con el polinomio grado 10 es 1.933421
El valor de la integral con el polinomio grado 11 es 1.933421
El valor de la integral con el polinomio grado 12 es 1.933421
El valor de la integral con el polinomio grado 13 es 1.933421
El valor de la integral con el polinomio grado 14 es 1.933421
El valor de la integral con el polinomio grado 15 es 1.933421
```

En la Ilustración 3 Error de la integral en función del grado, se puede observar que el menor error se obtiene con una aproximación con un polinomio de orden tres. Sin embargo el error dependerá de la función que se requiera resolver. Por lo tanto no siempre un mayor orden significará mayor precisión en el cálculo de la integral.

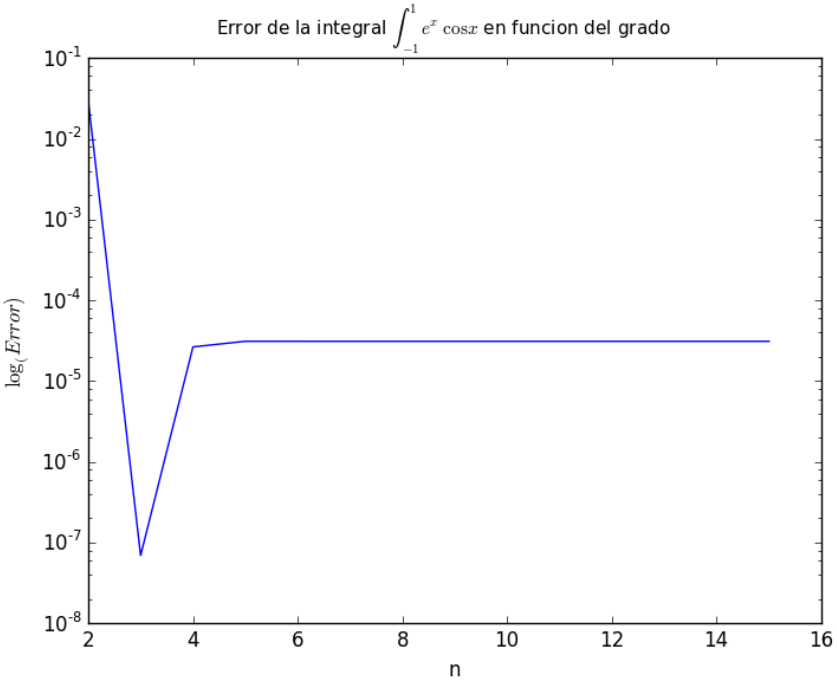

Ilustración 3 Error de la integral en función del grado

Ecuaciones Diferenciales Ordinarias con condiciones iniciales

Método de Euler

El método de Euler es la técnica más elemental para resolver problemas con condición inicial. El objetivo del método de Euler es obtener aproximaciones bien posicionadas de los problemas de condición inicial.

$$\frac{dy}{dt} = f\,(x\,, y), \qquad a \le t \le b\,, \qquad y(a) = \alpha$$

En este método se obtiene un conjunto de puntos llamado puntos de malla en el intervalo $[a\,, b]$ para obtener la solución continua de método se pueden interpolar los puntos de malla.

Para la deducción del método de Euler estipulamos que los puntos de malla son distribuidos igualmente en el intervalo $[a\,, b]$.

La distancia entre los puntos es conocida como el tamaño de paso $h = (b - a)/N$ donde N es la cantidad de puntos. Usamos el Teorema de Taylor para establecer el método de Euler.

Algoritmo del método de Euler

Datos de Entrada: Puntos iniciales (a) y finales (b), cantidad de puntos (N), condiciones iniciales (α) y las funciones que se quieren resolver. La letra t es la variable independiente.

Datos de Salida: Solución aproximada de las ecuaciones.

Paso 1: Calcular $h = \frac{b-a}{N}$

$t = a \quad y \quad w = \alpha$

Paso 2: Para $i = 1\,, 2\,, \dots, N$ realizar pasos 3 y 4

Paso 3: Calcular $w = w + h\,f\,(t, w)$ y $t = a + ih$

Paso 4: Salida valores de $w\,, t$

Paso 5: Parar devuelve los resultados de w y t.

Código de Python de método de Euler

```
def Euler( nP , Inicial ,  Final ,
   funciones , valoresIniciales ):
   ''' Resuelve un Euler con una lista de funciones
   para cualquier número de funciones '''
   h  = (Final - Inicial)/( nP*1.0) #Tamaño de Paso
   ind = numpy.zeros(nP + 1)
   listaDep = []
   for i in range(len(funciones)):
      nuevaLista = numpy.zeros(nP + 1)
      listaDep.append(nuevaLista)
   for j in range(nP + 1):
      if j == 0:
         ind[j] = Inicial
         listaAnterior = numpy.zeros(len(funciones))
         for i in range(len(listaDep)):
            listaDep[i][j] = valoresIniciales[i]
            listaAnterior[i] = valoresIniciales[i]
      else:
         ind[j] = ind[j-1] + h
         for i in range(len(listaDep)):
            listaDep[i][j] = listaDep[i][j-1] + h * funciones[i](ind[j-1] ,
listaAnterior)
            listaAnterior[i] = listaDep[i][j-1] + h * funciones[i](ind[j-1] ,
listaAnterior)
   return ind , listaDep
```

Método de Runge Kutta

Los métodos de Runge permiten obtener un error de truncamientos pequeño sin la necesidad de evaluar la derivada. La deducción del método de Runge Kutta se obtiene del Teorema de Taylor de dos variables. El método que se programó en esta de tesis fue el método de Runge Kutta de cuarto orden. Una diferencia entre este método y el método de Euler son las evaluaciones que se tienen que hacer en cada paso en el método de Euler solo se requiere hacer una evaluación mientras que en el método de Runge Kutta de cuarto orden se requieren realizar cuatro evaluaciones por cada paso.

Algoritmo del método de Runge Kutta

En esta tesis se programó el método de Runge Kutta de cuarto orden, ya que en general, es el más utilizado. Se presenta a continuación el algoritmo.

Datos de Entrada: Puntos iniciales (*a*) y finales (*b*), cantidad de puntos (*N*), condiciones iniciales (α) y las funciones que se quieren resolver. La letra *t* es la variable independiente.

Datos de Salida: Solución aproximada de las ecuaciones en cada intervalo dividido.

En cada paso se tienen que calcular las siguientes constantes:

Paso 1: Calcular $h = \frac{b-a}{N}$

$t = a \quad y \quad w = \alpha$

Paso 2: Para $i = 1, 2, \ldots N$ Realizar del paso 3 al 5

Paso 3: Calcular $k_1 = h\, f\,(t\,,\ w)$

$$k_2 = h\, f\left(t + \frac{h}{2}\,, w + \frac{1}{2}\,k_1\right)$$

$$k_3 = h\, f\left(t + \frac{h}{2}\,, w + \frac{1}{2}\,k_2\right)$$

$$k_4 = h\, f(t\,, w + \ k_3)$$

Paso 4: Calcular $w = w + (k_1 + \ k_2 + \ k_3 + \ k_4)/6$ y $t = a + ih$

Paso 5: Salida $(t\,, w)$

Paso 6: Parar devuelve los resultados de t y w.

Código de Python del método de Runge Kutta

```
def Runge ( nP , Inicial ,  Final ,
   funciones , valoresIniciales ):
   ''' Resuelve un Runge con una lista de funciones
   para cualquier número de funciones '''
```

```
def constantes(funciones , listaAnterior , current):
        lista = []
        for i in range(4):
            for j in range(len(funciones)):
                lista.append([])
                if i == 0:
                    elemento = h * funciones[j](current , listaAnterior)
                    lista[i].append(elemento)
                if i == 1:
                    constantes = lista[i-1]
                    argumento = []
                    for k in range(len(funciones)):
                        argumento.append(listaAnterior[k] + 0.5 * constantes[k])
                    elemento = h * funciones[j](current + 0.5* h , argumento)
                    lista[i].append(elemento)
                if i == 2:
                    constantes = lista[i-1]
                    argumento = []
                    for k in range(len(funciones)):
                        argumento.append(listaAnterior[k] + 0.5 * constantes[k])
                    elemento = h * funciones[j](current + 0.5* h , argumento)
                    lista[i].append(elemento)
                if i == 3:
                    constantes = lista[i-1]
                    argumento = []
                    for k in range(len(funciones)):
                        argumento.append(listaAnterior[k] + constantes[k])
                    elemento = h * funciones[j](current + h , argumento)
                    lista[i].append(elemento)
        return lista
h  = (Final - Inicial)/( nP*1.0) #Tamaño de Paso
ind = numpy.zeros(nP + 1)
listaDep = []
for i in range(len(funciones)):
    nuevaLista = numpy.zeros(nP + 1)
    listaDep.append(nuevaLista)
for j in range(nP + 1):
    if j == 0:
        ind[j] = Inicial
        listaAnterior = numpy.zeros(len(funciones))
        for i in range(len(listaDep)):
            listaDep[i][j] = valoresIniciales[i]
            listaAnterior[i] = valoresIniciales[i]
    else:
        ind[j] = ind[j-1] + h
        current = ind[j-1]
        lista = constantes(funciones , listaAnterior , current)
```

```
        for i in range(len(listaDep)):
            valor = 0
            for k in range(4):
                if k == 0:
                    valor = valor + lista[k][i]
                if k == 1:
                    valor = valor + 2 * lista[k][i]
                if k == 2:
                    valor = valor + 2 * lista[k][i]
                if k == 3:
                    valor = valor + lista[k][i]
            valor = valor / 6.0
            #print valor
            listaDep[i][j] = listaDep[i][j-1] + valor
            listaAnterior[i] = listaDep[i][j-1] + valor
    return ind , listaDep
```

Ejemplo de resolución de Ecuaciones Diferenciales Ordinarias

Se desea resolver la siguiente ecuación diferencial ordinaria

$$y' = y - t^2 + 1\,,\ 0 \le t \le 2\,,\ \ y(0) = 0.5$$

Con los métodos de Euler y Runge Kutta.

Editor

```
print '''La ecuacion que se quiere resolver es y' = y - t**2 +1
en el intervalo 0 <= t <= 2 con una condicion inicial y(0) = 0.5 '''
def funcion (ind , listaDep):
    return listaDep[0] - ind ** 2 + 1
runge =  metodosTesis.Runge(20 , 0 , 2 , [funcion] , [.5])
euler = metodosTesis.Euler(20 , 0 , 2 , [funcion] , [.5])
print ''
def exact(ind):
    return (ind + 1)**2 - 0.5* math.e ** ind
print '  t(i) |  Exacta | Euler  | Runge     '
print '-------------------------------------'
exacta = []
for i in range(len(runge[0])):
    print '  %.2f |  %.4f | %.4f | %.4f' % (euler[0][i] , exact(euler[0][i]), euler[1][0][i] , runge[1][0][i])
    exacta.append(exact(euler[0][i]))
```

Consola

t(i)	Exacta	Euler	Runge
0.00	0.5000	0.5000	0.5000
0.10	0.6574	0.6500	0.6574
0.20	0.8293	0.8140	0.8293
0.30	1.0151	0.9914	1.0151
0.40	1.2141	1.1815	1.2141
0.50	1.4256	1.3837	1.4256
0.60	1.6489	1.5971	1.6489
0.70	1.8831	1.8208	1.8831
0.80	2.1272	2.0538	2.1272
0.90	2.3802	2.2952	2.3802
1.00	2.6409	2.5438	2.6409
1.10	2.9079	2.7981	2.9079
1.20	3.1799	3.0569	3.1799
1.30	3.4554	3.3186	3.4553
1.40	3.7324	3.5815	3.7324
1.50	4.0092	3.8437	4.0092
1.60	4.2835	4.1030	4.2835
1.70	4.5530	4.3573	4.5530
1.80	4.8152	4.6040	4.8152
1.90	5.0671	4.8405	5.0670
2.00	5.3055	5.0635	5.3055

En la Ilustración 4 Solución de la ecuación diferencia se puede observar que con el mismo número de pasos la solución que da el método de Runge Kutta tiene un menor error que la solución que proporciona el método de Euler debido a que el método de Runge Kutta es de un orden mayor en este caso de cuarto orden sin embargo se requieren hacer una mayor cantidad de cálculos por cada paso.

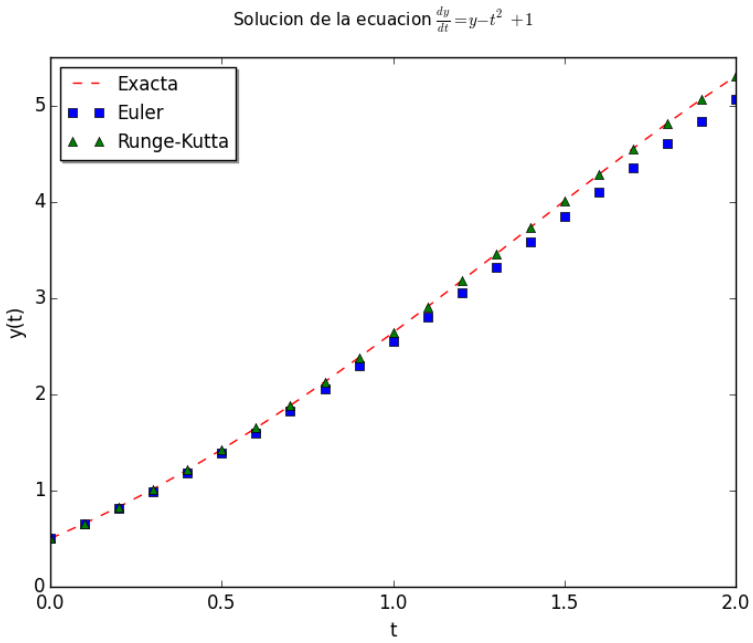

Ilustración 4 Solución de la ecuación diferencial

CAPÍTULO IV

En este capítulo se presentan problemas pertinentes a ingeniería química y cómo se pueden resolver con el archivo con los métodos descritos en el capítulo anterior. Todos los problemas se encuentran en, el archivo AplicacionesIngenieria.py para que este programa corra sin problemas es necesario tener el archivo metodosTesis.py en la misma carpeta de la computadora.

Problema de ecuación algebraica de una variable

Para el problema de ecuación de una variable se escogió el problema de un tanque flash del libro Operaciones de separación por etapas de equilibrio en la ingeniería química (Henley & Seader, 2000).

El problema es el siguiente:

Una alimentación de 100 kmol/h que contiene 10, 20, 30, 40 moles % de propano (3), n-butano (4), n-pentano (5) y n-hexano (6), respectivamente, entra en una columna de destilación a 100 psia y 200 °F. Suponiendo que existe el equilibrio ¿qué fracción de la alimentación entra como líquido y cuáles son las composiciones del líquido y el vapor?

Para las condiciones del flash $K_3 = 4.2,\ K_4 = 1.75$, $K_5 = 4.2$ y, $K_6 = 0.34$

En este problema es necesario resolver la ecuación de Rachford-Rice .

$$f(\varphi) = \sum_{i=1}^{c} \frac{z_i(1 - K_i)}{1 + \varphi\,(K_i - 1)} = 0$$

A continuación se muestra el código de Python utilizado para resolver este problema:

```
def calculoFraccion(  alim  , listaKi):
    def FuncionConvergencia(x):
        funcion = 0
        for i in range(len(listaKi)):
            funcion = funcion + (alim[i] * (1 - listaKi[i])) / ((1 + (listaKi[i] - 1 ) * x ))
        return funcion
    return metodosTesis.Newton ( FuncionConvergencia , 0.5)
```

Como se puede observar se está llamando a la función Newton localizada en el archivo metodosTesis.py. De argumentos toma la alimentación del tanque y una lista de constantes.

Para correr el programa se ocupó la función run_flash :

```
def run_flash ():
    ''' Crea un reporte con los resultados del programa'''
    alim = [0.10 , 0.20 , 0.30 , 0.40]
    listaKi =[4.2 , 1.75 , 0.74 , 0.34]
    fraccion = calculoFraccion(alim , listaKi)
    xi = calculodeXi ( alim , fraccion , listaKi)
    yi =  calculodeYi (xi ,  listaKi)
    NCOMPONENTES = ['Propano' , 'nButano' , 'nPentano' , 'nHexano']

    print 'La V/F calculada es de          : ' , round(fraccion,4)
    print '\n'
    print 'El reporte de las composiciones se muestra a continuación'
    print '\n'
    print 'NOMBRE DEL COMPONENTE|    zi    |    xi    |    yi    |'
    print '-----------------------------------------------------'
    i = 0
    for i in range(len(alim)):
        print '   ' , NCOMPONENTES[i] , ' '* (23 -
        len(NCOMPONENTES[i])) , str(alim[i]) , '   ' , str(
        round(xi[i] , 4)) , '   ' , str(round(yi[i] , 4))
run_flash()
```

Esta función imprime los siguientes resultados:

```
La V/F calculada es de          :  0.1211


El reporte de las composiciones se muestra a continuación


NOMBRE DEL COMPONENTE|    zi    |    xi    |    yi    |
-----------------------------------------------------
   Propano                  0.1    0.0721    0.3027
   nButano                  0.2    0.1833    0.3209
   nPentano                 0.3    0.3098    0.2292
   nHexano                  0.4    0.4347    0.1478
```

Como se puede ver el valor obtenido por el programa es similar obtenido en el libro que es de 0.1219. Otro punto a destacar es que se pueden cambiar las condiciones

de la alimentación o las constantes para que el programa realice otro cálculo de tanque flash.

Problema de sistema de ecuaciones algebraicas lineales

Para ejemplificar el problema de sistemas de ecuaciones lineales. Se escogió un sistema de separación, este problema se tomó en el libro Numerical Methods for Chemical Engineering (Beers, 2007) .

Para demostrar la importancia de los sistemas algebraicos lineales se considera un balance de masa. El sistema de separación está representado en la Ilustración 5 Sistema de separación , en este sistema se conoce el flujo de entrada (kg/h) y las fracciones másicas en cada corriente. Se desea calcular los flujos de salida para todas las corrientes de salida.

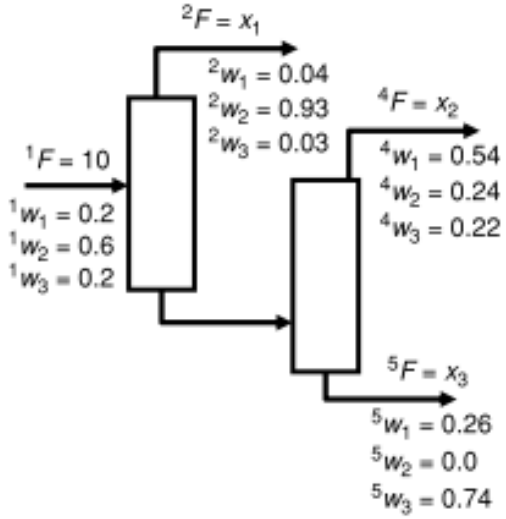

Ilustración 5 Sistema de separación (Beers, 2007)

Para este problema se tienen tres componentes. El problema pide calcular x_1 , x_2 , x_3 . Se plantean los balances de materia.

Balance total de materia.

$$x_1 + x_2 + x_3 = 10$$

Balance por componente.

El componente w_1

$$(0.04)x_1 + (0.54)x_2 + (0.26)x_3 = 10\ (0.2) = 2$$

El componente w_2

$$(0.93)x_1 + (0.24)x_2 + (0.0)x_3 = 10\ (0.6) = 6$$

Se obtiene un sistema de ecuaciones lineales de tres ecuaciones con tres incógnitas. El código para resolver este problema en Python es el siguiente:

Editor

```
def run_balances():
    coef = [[1,1,1] , [0.04 , 0.54 , 0.26] , [0.93 , 0.24 , 0.0]]
    resultados = [[10] , [2] ,[6]]
    resultado = metodosTesis.Gauss(coef , resultados)
    variables = ['x1' , 'x2' , 'x3']
    metodosTesis.sistemasPrint(coef, resultados, variables )
    print ''
    print 'La solución es:'
    print ''
    for i in range(len(variables)):
        print (variables[i] , resultado[i][0] )
run_balances()
```

Consola

```
[1, 1, 1]x1 = [10]
[0.04, 0.54, 0.26]x2 = [2]
[0.93, 0.24, 0.0]x3 = [6]

La solución es:

('x1', 5.82375478927203)
('x2', 2.432950191570881)
('x3', 1.7432950191570888)
```

Problema de un sistema de ecuaciones algebraicas no lineales

El problema se recopiló del libro de Numerical Methods for Chemical Engineering (Beers, 2007). El problema es el siguiente:

Como un ejemplo de los sistemas algebraicos no lineales, se considera un reactor CSTR operado de forma isotérmica, con un flujo constante de fluido. En el reactor se llevan a cabo las siguientes dos reacciones químicas. Se desean conocer las concentraciones en el estado estacionario.

$$A + B \rightarrow C \qquad con\ una\ velocidad\ de\ reacción \qquad r_{R_1} = k_1 C_A\, C_B$$

$$C + B \rightarrow D \qquad con\ una\ velocidad\ de\ reacción \qquad r_{R_2} = k_2 C_c C_B$$

Las concentraciones en estado estacionario son:

$$x_1 = C_A \qquad x_2 = C_B \qquad x_3 = C_C \qquad x_4 = C_D$$

Los balances de masa se escriben de la siguiente forma:

$$\frac{d}{dt}(VC_A) = \mathrm{v}\left(C_{A,in} - x_1\right) + V(-k_1 x_1 x_2)$$

$$\frac{d}{dt}(VC_B) = \mathrm{v}\left(C_{B,in} - x_2\right) + V(-k_1 x_1 x_2 - k_2 x_3 x_2)$$

$$\frac{d}{dt}(VC_C) = \mathrm{v}\left(C_{c,in} - x_3\right) + V(k_1 x_1 x_2 - k_2 x_3 x_2)$$

$$\frac{d}{dt}(VC_D) = \mathrm{v}\left(C_{D,in} - x_4\right) + V(k_2 x_3 x_2)$$

En donde $V\ es\ el\ volumen\ del\ reactor$ y $\mathrm{v}\ es\ el\ flujo\ de\ entrada\ del\ reactor$

En estado estacionario se obtienen las siguientes ecuaciones no lineales:

$$\mathrm{v}\left(C_{A,in} - x_1\right) + V(-k_1 x_1 x_2) = 0$$

$$\mathrm{v}\left(C_{B,in} - x_2\right) + V(-k_1 x_1 x_2 - k_2 x_3 x_2) = 0$$

$$\mathrm{v}\left(C_{c,in} - x_3\right) + V(k_1 x_1 x_2 - k_2 x_3 x_2) = 0$$

$$\mathrm{v}\left(C_{D,in} - x_4\right) + V(k_2 x_3 x_2) = 0$$

Los parámetros y los estimados iniciales son los siguientes:

Parámetros:

$$\mathrm{v} = 1 \qquad V = 1 \qquad k_1 = 1 \qquad k_2 = 1$$

Estimados iniciales:

$$C_{A,in} = 1 \qquad C_{B,in} = 2 \qquad C_{c,in} = 0 \qquad C_{D,in} = 0$$

El código de Python es:

Editor

```
def run_cstr():
    def  f1 (lista):
        return (1  - lista[0]) + 100 * (-lista[0] * lista[1])
    def f2 (lista):
        return (2 -lista[1]) + 100 * (-lista[0] * lista[1] -lista[1] * lista[2])
    def f3(lista):
        return (-lista[2]) + 100 * (lista[0] * lista[1] -lista[1] * lista[2])
    def f4(lista):
        return (-lista[3]) + 100 * (lista[1] * lista[2])
    funciones = [f1,f2,f3,f4]
    estimados = [[0.5] , [1] , [1] , [1]]
    var = ['CA' , 'CB' ,'CC' , 'CD']
    print ''
    print 'La solucion es: '
    sol =  metodosTesis.newtonSistemas(funciones , estimados)
    print ''
    for i in range(len(var)):
        print (var[i] , sol[i][0] )

run_cstr()
```

Los resultados son:

Consola

```
Problema de sistemas de ecuaciones no lineales


La solucion es:

('CA', 0.05661362831676501)
('CB', 0.16663586858184706)
('CC', 0.053408611948316996)
('CD', 0.889977759734918)
```

Los resultados son los valores de las concentraciones en estado estacionario. Los valores reportados en el libro son $C_A = 0.056614\,, C_B = 0.16664\,, C_C = 0.053409\; y\; C_D = 0.88998.$

Problemas de Integración numérica

Problema de evaluación numérica

Para ilustrar el uso de evaluación numérica se eligió un problema del libro Elementos de ingeniería de las reacciones química (Fogler, 2008) es el ejemplo 2-3.

Determinación de un tamaño de un PFR

La reacción que describen los datos de la Tabla 10 Datos Procesados Reactor PFR debe correrse en un PFR. La velocidad de flujo alimentada A es 0.4 mol/s. Use una de la reglas de integración numérica para determinar el volumen del reactor si se desea una conversión del 80%.

Tabla 10 Datos Procesados Reactor PFR

X	0.0	0.1	0.2	0.4	0.6	0.7	0.8
$\frac{F_{A0}}{-r_A}$ (m^3)	0.89	1.08	1.33	2.05	3.54	5.06	8.0

La integral que se requiere resolver para este problema es la siguiente:

$$V = \int_0^{0.8} \frac{F_{A0}}{-r_A}\, dX$$

Editor

```
104 def run_pfr():
105     conversion = [0.0 , 0.2 , 0.4 , 0.6 , 0.8]
106     fa0_ra =[0.89 , 1.33 , 2.05 ,3.54 , 8.0]
107     sol =  metodosTesis.integracionNumerica(conversion, fa0_ra)
108     print 'La solución de la integral es: ' ,sol
109
```

Los resultados se presentan a continuación:

Consola

```
Problema de evaluacion numerica de la integral
La solución de la integral es:  2.16466666667
```

El volumen del PFR debe de ser de 2.165 m^3. El valor de la integral es el mismo reportado en el libro.

Problema de integral numérica

Para el problema de integral numérica se escogió un problema de una página web de la (University of South Florida). Este problema es el sugerido para ilustra el uso de cuadraturas gaussianas en la ingeniería química.

El problema dice así:

En un intento de comprender el mecanismo de despolarización en una celda de combustible, se desarrolló un modelo de electrocinética para una mezcla oxigeno metanol en platino de en laboratorio FAMU. Este modelo es una relación con una forma integral. Para encontrar el tiempo en segundos requerido para que el 50% de oxigeno sea consumido se necesita resolver la siguiente integral.

$$T = -\int_{1.22* \ 10^{-6}}^{0.61* \ 10^{-6}} \frac{6.73x + 4.3025 * \ 10^{-7}}{2.316 * \ 10^{-11}x} \, dx$$

Encontrar el valor de la integral.

El código para encontrar el valor de la integral es el siguiente:

Editor

```
116 print 'Problema de integral numerica '
117 print ''
118 def run_integral():
119     def f(x):
120         return -1.0 * (6.73 * x + 4.3025 * 10 ** -7 ) / (2.316 * 10 ** -11 * x)
121     a = 1.22 * 10 ** -6
122     b = 0.61 * 10 ** -6
123     n = 14
124     cuadraturas =  metodosTesis.cuadraturasGausianas(f, a, b, n)
125     composite = metodosTesis.compositeSimpson(f,a,b,n)
126     print 'El valor de la integral obtenido con cuadraturas es: ' , cuadraturas
127     print 'El valor de la integral con el metodo compuesto de simpson es' ,  composite
128     print ''
129 run_integral()
```

La solución es la siguiente:

Consola

```
Problema de integral numerica

El valor de la integral obtenido con cuadraturas es:  190134.998896
El valor de la integral con el metodo compuesto de simpson es 190135.01382
```

El valor exacto de la integral es de 190140 segundos se puede observar que el error en los dos métodos es muy pequeño.

Problema de ecuaciones diferenciales ordinarias con condiciones iniciales

Para el problema de sistemas de ecuaciones diferenciales se escogió el ejemplo 8-4 del libro Elementos de ingeniería de las reacciones química (Fogler, 2008). En este problema se pide calcular un reactor PFR en estado estacionario con intercambiador de calor de la reacción de la isomerización del n-butano. En este problema se pide graficar la X, X_e y la temperatura a lo largo del reactor y se pregunta si el reactor alcanzo una temperatura mayor a los 325 K.

El balance de materia en base molar:

$$\frac{dX}{dV} = \frac{-r_A}{F_{A0}}$$

Ley de velocidad y estequiometria

$$r_A = -kC_{A0}\left[1 + \left(1 + \frac{1}{Kc}\right)X\right]$$

Con:

$$k = 31.1\exp\left[7906\left(\frac{T-360}{360T}\right)\right]\ h^{-1}$$

$$K_C = 3.03\exp-\left[830.3\left(\frac{T-333}{333T}\right)\right]$$

Con los siguientes parámetros $C_{A0} = 9.3$, $F_{A0} = 14.7$, $\Delta H = -6900$, $U_A = 5000$, $T_A = 310\ y\ Cp_A = 159$.

El balance de energía:

$$\frac{dT}{dV} = \frac{r_A\,\Delta H_{RX} - Ua\,(T - T_a)}{F_{A0}\,Cp_A}$$

El código que resuelve estas ecuaciones es:

```
def run_pfr_adiabatico():
    Ca0 = 9.3 ; Fa0 = 0.9 * 163 * 0.1 ;deltaH = -6900 ;Ua = 5000 ; Ta = 310 ; Cpo = 159
    Inicial = 0 ; Final = 5.0 ; nP = 10000 ; valoresIniciales = [310 , 0.0 ]
    def calculoKc(lista_dep):
        return  3.03 * math.exp(-830.3 * (lista_dep[0]-333.0)/(lista_dep[0]*333.0))
    def calculok(lista_dep):
        T = lista_dep[0]
        return   31.1 * math.exp(7906 * (T - 360) / ( T * 360))
    def ra(lista_dep):
        return -calculok(lista_dep) * Ca0 * (1-(1 + 1/calculoKc(lista_dep))* lista_dep[1])
    def f2 ( ind , lista_dep):
        return -ra(lista_dep)/Fa0
    def f1 ( ind , lista_dep ):
        return ((ra(lista_dep)*1.0 * deltaH) -Ua*(lista_dep[0] - Ta))/(Cpo * Fa0)
    sol = metodosTesis.Runge(nP , Inicial , Final , [f1,f2] , valoresIniciales)
    perfil_volumen = sol[0] ; perfil_temp = sol[1][0] ;perfil_conver = sol[1][1]
    print perfil_temp
    def calculoKc(T):
        return  3.03 * math.exp(-830.3 * (T -333.0)/(T*333.0))
    equilibrio = [calculoKc(T) / (1 + calculoKc(T)) for T in perfil_temp]
    def perfil_conversiones():
        lines = plt.plot(perfil_volumen  , perfil_conver  , perfil_volumen , equilibrio )
        plt.setp(lines, color='k', linewidth=1.4) ; plt.title('Perfil de conversiones')
        plt.xlabel('Volumen [m$^{3}$]') ; plt.ylabel('Conversion') ; plt.axis([0, 5.0 , 0, 1.0])
        plt.text(.96, .48, 'X')  ;plt.text(.96, .80, 'Xe') ;  plt.show()
    def perfil_temp():
        lines = plt.plot(perfil_volumen , sol[1][0])
        plt.setp(lines, color='k', linewidth=1.4) ; plt.title('Perfil de temperaturas')
        plt.xlabel('Volumen [m$^{3}$]') ; plt.ylabel('T[K]') ; plt.axis([0, 5.0 , 310, 325])
        plt.show()
    #perfil_conversiones()
    perfil_temp()
run_pfr_adiabatico()
```

La Ilustración 6 Perfil de temperaturas y conversiones muestra los perfiles de Temperatura y conversión que pide el problema. Los perfiles son similares obtenidos al programa de Polymath se observan en la Ilustración 7 Perfil de temperaturas y conversiones obtenidos en Polymath.son muy similares.

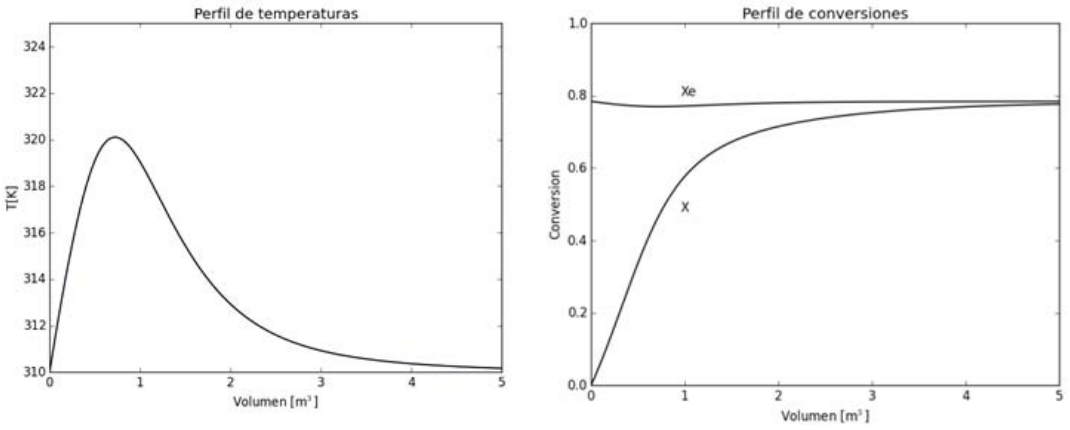

Ilustración 6 Perfil de temperaturas y conversiones obtenidos en Python

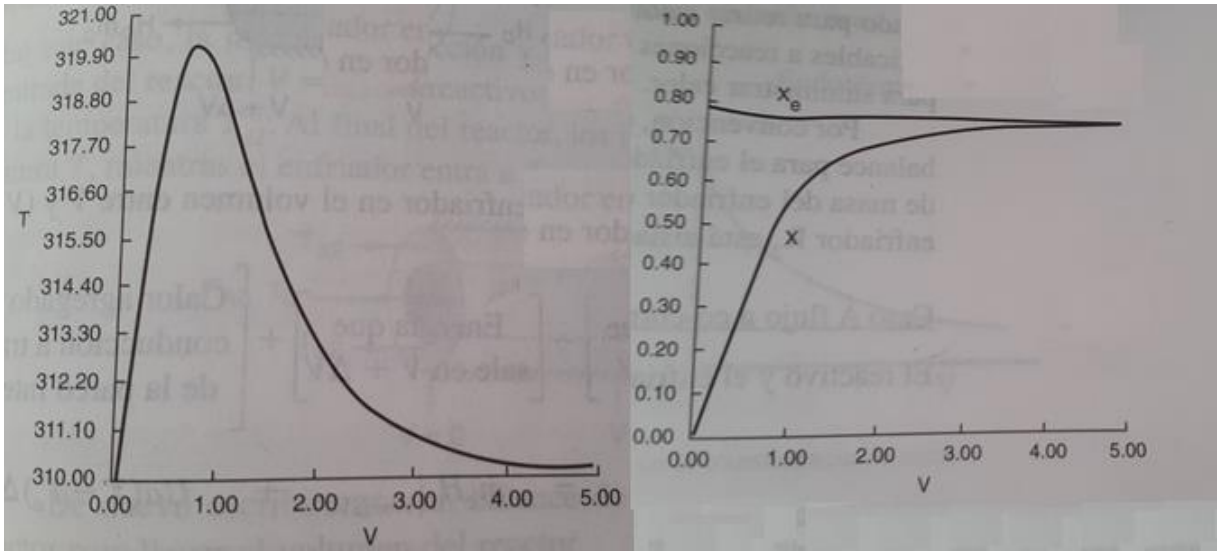

Ilustración 7 Perfil de temperaturas y conversiones obtenidos en Polymath

La temperatura del reactor no alcanza los 325 K.

CAPÍTULO V

Conclusiones

En este trabajo se logró realizar la creación de un archivo con extensión .py. Con este archivo creado se logró resolver una variedad de problemas de Ingeniería Química problemas clásicos como la vaporización de un tanque flash hasta la solución de un reactor PFR en estado estacionario con intercambiador de calor. Con lo anterior se demuestra la importancia de programar los métodos numéricos ya que resuelven una amplia variedad de problemas de ingeniería química. Una aspecto a destacar de programar, es la versatilidad de los programas ya que el autor puede cambiar diferentes parámetros (ecuaciones, iteraciones, forma de calcular la derivada, etc.). Debido a esto el programador puede ajustar los programas a sus necesidades.

En esta tesis también se muestran cómo se implementan los algoritmos a código de computadora viendo como la CACHE, tiene la razón en que los graduados de Ingeniería Química necesitan está habilidad. La principal razón por la que se ocupó Python fue su facilidad para programar, esto se muestra en que los algoritmos y el código escrito en Python son muy similares.

Otro punto importante a resaltar es que como vimos en las aplicaciones de ingeniería química, con este archivo se pueden resolver problemas que normalmente son resueltos por programas de alto costo como Polymath, por lo que el código presentado aquí es una alternativa que de acceso libre; además el IDE que se ocupó en esta tesis es fácil de descargar e instalar y también es gratuito. Es conveniente recordar que Python es un lenguaje multiplataforma.

Por último se pueden destacar algunas de las características de Python que se ocuparon a lo largo de este trabajo como:

- Sintaxis simplificada
- Creación de módulos auxiliares
- El uso de librerías externas
- Funciones de alto orden

Demostrando que Python es un lenguaje muy útil que se recomienda ampliamente para la creación de nueva herramientas.

Comentarios

El objetivo principal de esta tesis fue el demostrar la importancia de programar. Durante el trascurso de este trabajo se creó un módulo que cualquier persona lo puede obtener. Con este módulo se resolvieron diferentes problemas de Ingeniería Química. Se espera que este módulo sea de utilidad a la comunidad de la Facultad de Química

APÉNDICE

Los archivos:

- metodosTesis.py
- Ejemplos.py
- AplicacionesIngenieria.py

Se pueden descargar en la siguiente liga

https://www.dropbox.com/sh/0c5ogfeou8g9twj/AAAezpLTUaxD7n3Jb6TfEdcFa?dl=0

Para abrir estos archivos es necesario instalar Enthoughy Canopy. En la siguiente liga explica detalladamente como instalarlo.

https://courses.edx.org/courses/MITx/6.00.1x_5/1T2015/courseware/Week_1/Problem_Set_1/

Una vez instalado se abre el Programa para encontrarlo se sigue la siguiente ruta:

Inicio -> Todos los programas -> Enthoughy Canopy -> Canopy

Una vez abierto el programa se selecciona la pestaña File -> Open y se selecciona la dirección en donde se descargaron los archivos anteriores se escoge el archivo que se desea abrir y ya se puede hacer uso de él.

ÍNDICE DE TABLAS E ILUSTRACIONES

Tabla 1 Desempeño de funciones a comparación de C 3
Tabla 2 Indicador TIOBE 2015 4
Tabla 3 Principales funciones programadas en metodosTesis.py 5
Tabla 4 Palabras clave en Python 7
Tabla 5 Algunas funciones del módulo math 9
Tabla 6 Principales operaciones con números en Python 10
Tabla 7 Condiciones más comunes en Python 15
Tabla 8 Ejemplos de matriz 27
Tabla 9 Tabla de valores problema evaluación numérica de integrales 37
Tabla 10 Datos Procesados Reactor PFR 55

Ilustración 1 Empresas que ocupan Python 3
Ilustración 2 Indentación en Python (Briggs, 2013) 15
Ilustración 3 Error de la integral en función del grado 42
Ilustración 4 Solución de la ecuación diferencial 48
Ilustración 5 Sistema de separación (Beers, 2007) 51
Ilustración 6 Perfil de temperaturas y conversiones obtenidos en Python 59
Ilustración 7 Perfil de temperaturas y conversiones obtenidos en Polymath 59

BIBLIOGRAFÍA

University of South Florida. (s.f.). *Holistic Numerical Methods*. Recuperado el 10 de Junio de 2015, de Holistic Numerical Methods: http://nm.mathforcollege.com/topics/gauss_quadrature.html

Beers, K. J. (2007). *Numerical Methods for Chemical Engineering.* New York: Cambride University Press.

Boragan, A., & Fernández-Villaverde, J. (5 de Agosto de 2014). *Penn Economics.* Recuperado el 2015 de Julio de 17 , de http://economics.sas.upenn.edu/~jesusfv/comparison_languages.pdf

Briggs, J. (2013). *Python For Kids.* San Francisco: No Starch Press, Inc.

Burden, R., & Faires, D. (2005). *Numerical Analysis.* Boston: Brooks/Cole.

Chapra, S. C., & Canale, R. P. (2010). *Numerical Methods for Engineers.* New York: Mc Graw Hill.

David, S. (s.f). Recuperado el 2015 de Agosto de 12, de http://www.engr.uky.edu/~aseeched/papers/2001/a1024.pdf

Fogler, H. S. (2008). *Elementos de ingeniería de las reacciones química.* México: Pearson.

González, R. (s.f). Python para todos. En R. González. España: Creative Commons.

Henley, E., & Seader, J. (2000). *Operaciones de separación por etapas de equilibrio en la ingeniería química.* México: Reverté.

julia. (s.f.). *julia*. Recuperado el 2015 de Agosto de 17, de http://julialang.org/

K.G., P. (5 de Junio de 2004). *Kettering University.* Recuperado el 20 de Marzo de 2015, de http://paws.kettering.edu//~ktebeest/math305/simp38b.pdf

Kammermans, M. (5 de Junio de 2011). *GitHub*. Recuperado el 12 de 03 de 2015, de http://pomax.github.io/bezierinfo/legendre-gauss.html

Lamber, J. (2009). *Department of Mathematics.* Recuperado el 2015 de Agosto de 17, de http://www.math.usm.edu/lambers/mat610/sum10/lecture4.pdf

Python Software Foundation. (03 de Mayo de 2015). *Python 2.7.10rc0 documentation*. Recuperado el 03 de Mayo de 2015, de https://docs.python.org/2/library/math.html

Summerfield, M. (2007). *Rapid GUI programming with Python and Qt.* USA: Prentice Hall.

Vizcarra, P. (12 de Diciembre de 2014). *Yo programador: #LaHoradelCódigo*. Recuperado el 20 de Abril de 2015, de Diario Correo: http://diariocorreo.pe/miscelanea/yo-programador-lahoradelcodigo-550765/

Wikipedia. (20 de Abril de 2015). *Wikipedia, La enciclopedia libre.* Recuperado el 20 de Abril de 2015, de http://es.wikipedia.org/wiki/Python