Grupo 3
Miguel Angel Avila Santos
Juan Andres Martinez Amado
Jorge Luis Esposito Albornoz
Juan Sebastian Herrera Guaitero

Asignatura
Análisis numérico

Profesora
Eddy Herrera Daza

Taller 4

Integración y Ecuaciones Diferenciales

7 de Noviembre de 2021

1. Integración

1. c. Dados los siguientes puntos:

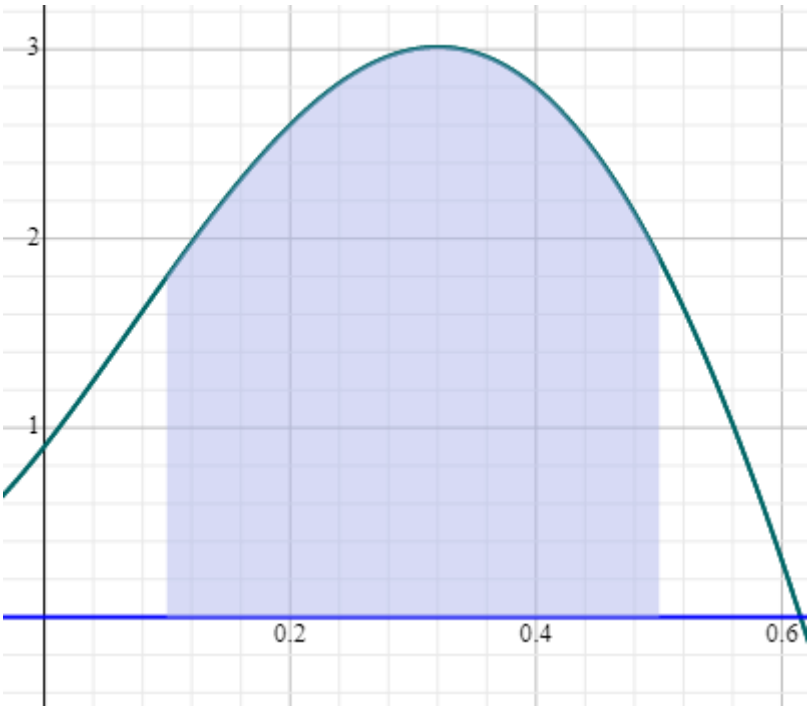
(0.1,1.8), (0.2,2.6),(0.3,3.0),(0.4,2.8),(0.5,1.9)

Utilice la fórmula de Simpson para encontrar una aproximación del área bajo la curva y calcule su error de truncamiento. Qué resultado se obtendría si primero interpola con Lagrange y luego calcula la integral compare los resultados con respecto al área.

Para el desarrollo del punto se hizo uso de la librería scipy de Python, la cual incorpora funciones de integración e interpolación, las cuales incluyen la fórmula de Simpson al igual que la interpolación de Lagrange, inicialmente se aproximó el área bajo con la función simpson() enviando como parámetro los valores de x y y de los puntos dados, posteriormente, se obtuvo un polinomio resultado de la interpolación de Lagrange y con esta se repite el proceso con la fórmula de simpson, pero esta vez enviado como parámetro el resultado de los puntos x evaluados en el polinomio obtenido y los valores de x originales, finalmente se calculó el error de truncamiento.

Resultados:

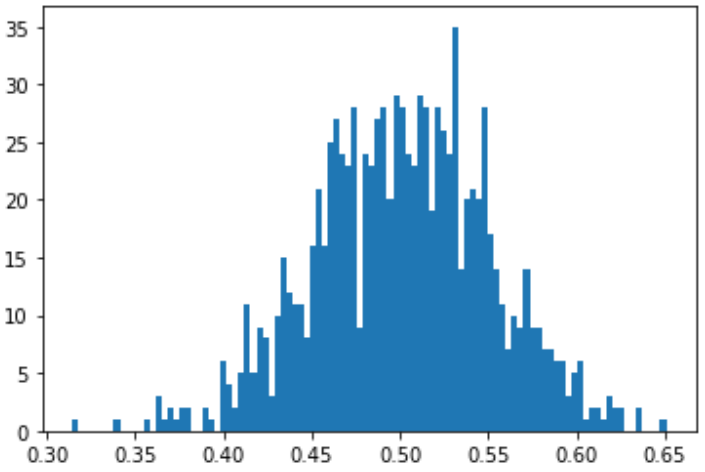
```
Area bajo la curva(simpson): 1.0433333333333334
-----------------------------------------
|     POLINOMIO INTERPOLADO(LAGRANGE)     |
-----------------------------------------
       4      3         2
41.67 x - 75 x + 14.58 x + 8.25 x + 0.9
-----------------------------------------
Area bajo la curva(simpson) con la interpolacion lagrange: 1.0433333333333057
Error Truncamiento: 2.7755575615628914e-14
```



Gráfica con la interpolación de Lagrange

1.l. Genere una tabla, donde se puede aproximar los valores de la distribución binomial a una normal, con un corrección por continuidad de 0.05 y para cuando p=0.5 y n=1000. Compare los valores aproximados con los valores exactos de la binomial.

La distribución normal es una forma de representar datos organizando la distribución de probabilidad de cada uno de los valores . La mayoría de los valores permanecen alrededor del valor medio, lo que hace que la disposición sea simétrica.



El modelo de distribución binomial trata de encontrar la probabilidad de éxito de un evento que tiene solo dos resultados posibles en una serie de experimentos. Por ejemplo, lanzar una moneda siempre da cara o cruz. La probabilidad de encontrar exactamente 3 caras al lanzar una moneda repetidamente durante 10 veces se estima durante la distribución binomial.

Usamos la biblioteca de python seaborn que tiene funciones integradas para crear tales gráficos de distribución de probabilidad. Además, el paquete scipy ayuda a crear la distribución binomial.

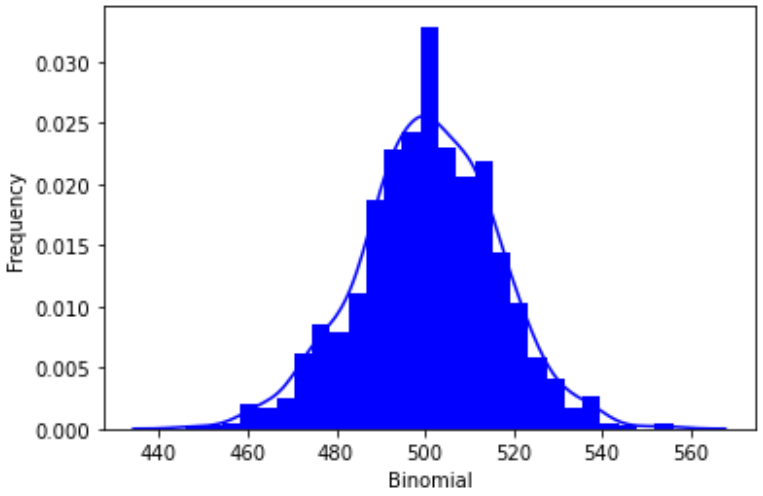


Tabla de comparación de los datos dados por la distribución binomial y la normal con sus rangos.

Frecuencia	Binomial/1000	Normal
5	0,46 y 0,53	0,40 y 0,60
10	0,48 y 0,52	0,43 y 0,58
15	0,49 y 0,515	0,45 y 0,55
20	0,495 y 0,51	0,46 y 0,53
25	0,5	0,48 y 0,52
30	0,5	0,51

Comparando las dos gráficas con los valores que dieron implementando la distribución binomial y normal los valores no cuadran con sus respectivos datos, en la binomial se ubicó una línea que representa como seria la grafica de la distribución normal pero con las escalas de la gráfica se nota que están muy distorsionados los datos de uno al otro.Pero por otro lado al cambiar la escala de valores de la gráfica binomial las Y por 1000 y las X en 1000 ya los datos serían más parecidos y podemos ver que el valor medio en ambas distribuciones es de 50.

2. Ecuaciones diferenciales

2.b. Utilice Taylor de orden cuatro para aproximar las soluciones en t=0.4;0.01;1.55 y estime el error de truncamiento. Finalmente compare numérica y gráficamente, la solución aproximada con la solución exacta en cada punto y en general osea un error total promedio.

Serie de Taylor

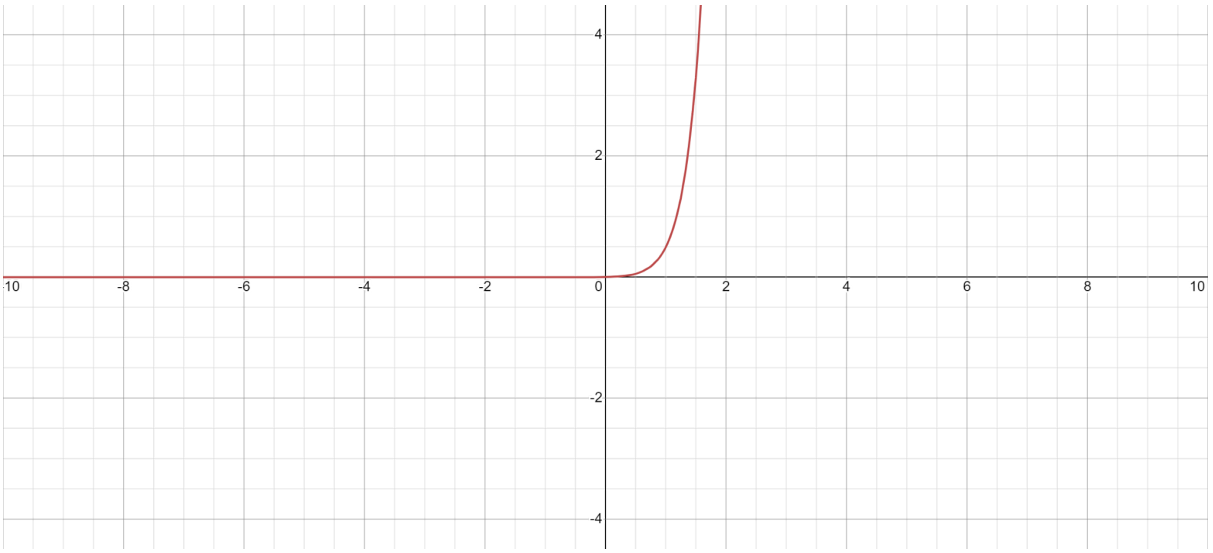
$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Para el desarrollo del punto se hizo uso de la librería sympy de Python, la cual incorpora funciones de integración,derivación,etc. Gracias a esta librería, se logró implementar el algoritmo de Taylor con la que se derivan n-ésima veces la función para así poder obtener la aproximación que se busca.

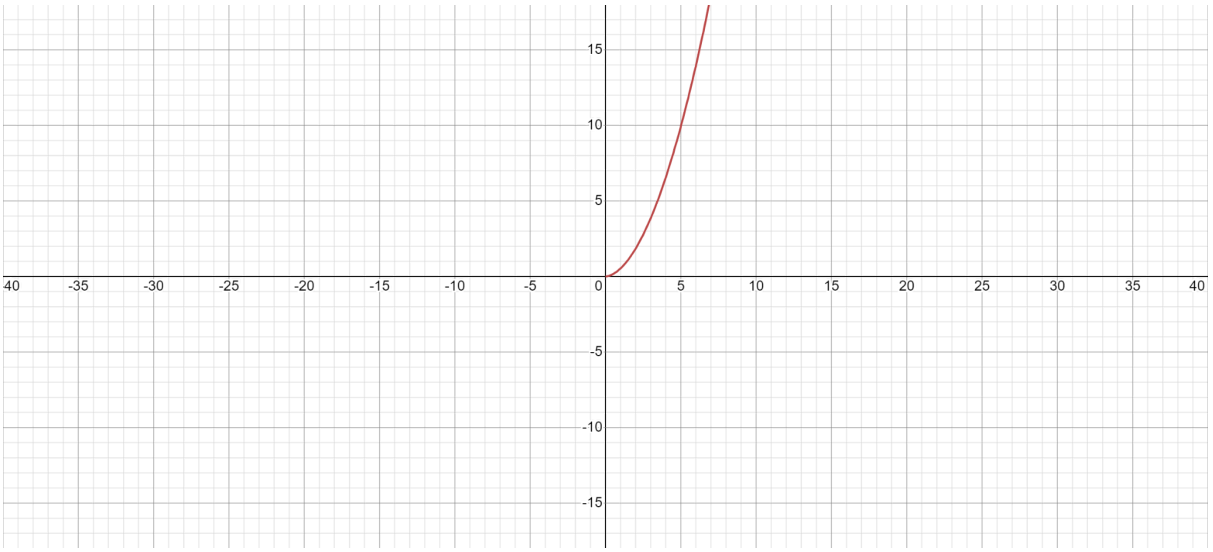
Resultados obtenidos:

```
POL TAYLOR: 9*x**4/2 + 9*x**3/2 + 3*x**2 + x - 40
{'Real': -38.6719532309054, 'Aprox': -38.7168000000000, 'Error': 0.0448467690946259}
{'Real': -39.9896954546605, 'Aprox': -39.9896954550000, 'Error': 3.39532846282964e-10}
{'Real': 122.106727644527, 'Aprox': 11.4889656250000, 'Error': 110.617762019527}
```

Gráfica original:



Gráfica aproximación taylor:



	Real	Aproximación	Error
0.4	-38.6719532309054	-38.7168000000000	0.0448467690946259
0.01	-39.9896954546605	-39.9896954550000	3.39532846282964e-10
1.55	122.106727644527	11.4889656250000	110.617762019527

Las series de potencia de Taylor son muy importantes en la representación o aproximación de funciones mediante series de suma y potencias enteras de polinomios como también la reducir los cálculos de las mismas, sin embargo, como podemos ver en los resultados obtenidos una vez realizada la aproximación para el valor 1.55 el error es muy grande y su aproximación no es lo suficientemente acertada cuando se compara con el valor real. Para los valores 0.4 y 0.01, el error entre la aproximación y el real es menor a 1 con lo que podemos decir que la aproximación es buena para dichos valores.