Presentación del Reto 1

Stefania García, Santiago Torres, Esteban Villa, Julian Tarazona

12 de septiembre de 2020

Este documento tiene como fin resolver y analizar los ejercicios presentados en el reto 1 utilizando métodos como el de Brent, Laguerre, modificaciones del método de Horner y Newton además de la aproximación de Remez y Taylor con los cuales a través de las herramientas Phyton y R darán cuenta a diversas conclusiones acerca de su eficiencia y precisión a la hora de encontrar raíces.

1.Evaluación de las raíces de un Polinomio

Evaluar raíces de un polinomio implica varios desafíos. Para poder evaluar las raíces posibles de un polinomio, se deben tener en cuenta las derivadas evaluadas en valores $[x_0]$ por lo que se introdujo al método Horner la posibilidad de calcular $f'(x_0)$ y $f''(x_0)$.

Para resolver a primera parte de este reto se usó el polinomio $4x^3+2x^2+3x+1$, se calcula la derivada en este polinomio la cual es $12x^2 + 4x + 3$, se calcula la segunda derivada que sería $12x + 4$. La primera derivada se almacenó en un arreglo llamado coeficientesDerivada y la segunda derivada se almacenó en un arreglo llamado coeficientesSegundaDerivada.

Con el primer arreglo se realiza un ciclo donde se tiene un contador inicializado en 1, ese ciclo usa dicho contador hasta que recorra el grado del polinomio, en dicho ciclo se tiene en cuenta el valor de x=3 y se utiliza la formula: $res = res * x + coeficienteDerivadas[j]$

Con el segundo arreglo que es la segunda derivada de la función se realiza un procedimiento igual al mencionado anteriormente con un contador distinto, ese ciclo usa dicho contador hasta que recorra el grado del polinomio, en dicho ciclo se tiene en cuenta el valor de x=3 y se utiliza la formula: $res = res * x + coeficienteSegundaDerivadas[j]$
Al realizar estas dos operaciones se obtuvieron los siguientes resultados:

	resultado2
1	24
2	76

En esta tabla se tienen en cuenta las iteraciones que hace el algoritmo donde podemos observar que el último resultado es 76 que es la última iteración que realiza al generar la segunda derivada de la función.

En la segunda parte cambiando de escenario y teniendo en cuenta que el vector ahora puede tener coeficientes con números complejos se usó el polinomio $2x^4+3x^3+2x^2+1$, donde se hace un ciclo parecido a los anteriores pero tomando el valor de $x = 2 + i$. Usando la formula: $res = res * x + coeficientesI[n$.
Donde se muestran los resultados de sus iteraciones en la siguiente tabla:

	resultado
1	2+0i
2	7+2i
3	14+11i
4	18+36i

En esta tabla se tienen en cuenta los resultados que hace el algoritmo por cada iteración donde se observa que el último resultado es 18+36i que es la última iteración que realiza el algortimo.

Posteriormente se pide integrar el método anteriormente mencionado al método de Newton para crear un método de Newton-Horner para evaluar si el método funciona se hicieron pruebas con los siguientes polinomios:

Las raíces del polinomio $5x^2 + 3x - 2$ halladas por el método Newton-Horner con una tolerancia de $1 * e^{-70}$ son:

```
Ingrese el grado n del polinomio: 2
Ingrese el coeficiente 1

5
Ingrese el coeficiente 2

3
Ingrese el coeficiente 3

-2

Ingrese el valor x0 para la primera aproximacion 3
La raiz  1 es (0.4+0j)
Las iteraciones fueron:  8

La raiz  2 es (-1+0j)
Las iteraciones fueron:  3
```

Las raíces del polinomio $3x^4 + 5x^3 - 2x^2 + x - 7$ halladas por el método Newton-Horner con una tolerancia de $1 * e^{-70}$ son:

```
Ingrese el grado n del polinomio: 4
Ingrese el coeficiente 1

3
Ingrese el coeficiente 2

5
Ingrese el coeficiente 3

-2
Ingrese el coeficiente 4

1
Ingrese el coeficiente 5

-7

Ingrese el valor x0 para la primera aproximacion 3
La raiz  1 es (1+0j)
Las iteraciones fueron:  10

La raiz  2 es (-2.238858376611623+0j)
Las iteraciones fueron:  40
```

Las raíces con parte imaginaria del polinomio $x^4 - 5x^3 + 4x^2 - 3x + 2$ halladas por el método Newton-Horner con una tolerancia de $1 * e^{-20}$ son:

```
Ingrese el grado n del polinomio: 4
Ingrese el coeficiente 1

1
Ingrese el coeficiente 2

-5
Ingrese el coeficiente 3

4
Ingrese el coeficiente 4

-3
Ingrese el coeficiente 5

2

Ingrese el valor x0 para la primera aproximacion 1+j
La raiz  1 es (0.8023068018257805+0j)
Las iteraciones fueron:  9

La raiz  2 es (4.188847029536467+5.126778006725572e-26j)
Las iteraciones fueron:  17

La raiz  3 es (0.0044230843188876391+0.7714190717314996j)
Las iteraciones fueron:  90

La raiz  4 es (0.004423084318876391-0.7714190717314996j)
Las iteraciones fueron:  2
```

Ya teniendo en cuenta que el método desarrollado es funcional se procede a evaluar el polinomio $x^4 - 5x^3 - 9x^2 + 155x - 250$ con las siguientes tolerancias:

Resultado con tolerancia de $1 * e^{-4}$:

```
Ingrese el grado n del polinomio: 4
Ingrese el coeficiente 1

1
Ingrese el coeficiente 2

-5
Ingrese el coeficiente 3

-9
Ingrese el coeficiente 4

155
Ingrese el coeficiente 5

-250

Ingrese el valor x0 para la primera aproximacion 1
La raiz  1 es (1.9999999999994171+0j)
Las iteraciones fueron:  4

La raiz  2 es (-4.9999997284011 92+0j)
Las iteraciones fueron:  16
```

Resultado con tolerancia de $1 * e^{-8}$:

```
Ingrese el grado n del polinomio: 4
Ingrese el coeficiente 1

1
Ingrese el coeficiente 2

-5
Ingrese el coeficiente 3

-9
Ingrese el coeficiente 4

155
Ingrese el coeficiente 5

-250

Ingrese el valor x0 para la primera aproximacion 1
La raiz  1 es (2+0j)
Las iteraciones fueron:  5

La raiz  2 es (-4.999999999999916+0j)
Las iteraciones fueron:  17
```

Resultado con tolerancia de $1 * e^{-32}$:

```
Ingrese el grado n del polinomio: 4
Ingrese el coeficiente 1

1
Ingrese el coeficiente 2

-5
Ingrese el coeficiente 3

-9
Ingrese el coeficiente 4

155
Ingrese el coeficiente 5

-250

Ingrese el valor x0 para la primera aproximacion 1
La raiz  1 es (2+0j)
Las iteraciones fueron:  6

La raiz  2 es (-5+0j)
Las iteraciones fueron:  17
```

Resultado con tolerancia de $1 * e^{-90}$:

```
Ingrese el grado n del polinomio: 4
Ingrese el coeficiente 1

1
Ingrese el coeficiente 2

-5
Ingrese el coeficiente 3

-9
Ingrese el coeficiente 4

155
Ingrese el coeficiente 5

-250

Ingrese el valor x0 para la primera aproximacion 1
La raiz  1 es (2+0j)
Las iteraciones fueron:  6

La raiz  2 es (-5+0j)
Las iteraciones fueron:  17
```

Ahora bien, teniendo los resultados del método de Newton-Horner con sus respectivas pruebas de tolerancia, es necesario hacer una comparación entre este ultimo y el método de Laguerre para el cual se evaluarán los mismos polinomios de prueba en comparación con Newton-Horner.

Las raíces del polinomio $5x^2 + 3x - 2$ halladas por el método Laguerre con una tolerancia de $1 * e^{-70}$ son:

	x	la1
1	3.3	0.4

	x	la2
1	-2	-1

Las raíces con parte imaginaria del polinomio $x^4 - 5x^3 + 4x^2 - 3x + 2$ halladas por el método Laguerre con una tolerancia de $1 * e^{-90}$ son:

	x	la
1	-3	-2.067174

En este caso el método de Laguerre es incapaz de determinar la raíz real = 0 y las 2 raíces imaginarias que tiene este polinomio.

Teniendo presente lo anterior se prosigue a evaluar de nuevo el polinomio $x^4 - 5x^3 - 9x^2 + 155x - 250$ en las siguientes tolerancias con el método Laguerre:

Resultado con tolerancia de $1 * e^{-8}$:

```
> ################################################
> ####  TOLERANCIA 1e-8  ######
> start_time <- Sys.time()
> poli1 <- c(1.0, -5.0, -9.0, 155.0, -250.0)
> laguerre(poli1, 3.3,25,.Machine$double.eps^(1/2))
[1] 2
> end_time <- Sys.time()
>
> cat("Tiempo de ejecucion")
Tiempo de ejecucion> print(end_time - start_time)
Time difference of 0.01595712 secs
> cat("tolerancia")
tolerancia> print(.Machine$double.eps^(1/2))
[1] 1.490116e-08
>
> start_time <- Sys.time()
> poli2 <- c(1.0, -5.0, -9.0, 155.0, -250.0)
> laguerre(poli2, -6.2,25,.Machine$double.eps^(1/2))
[1] -5
> end_time <- Sys.time()
>
>
> cat("Tiempo de ejecucion")
Tiempo de ejecucion> print(end_time - start_time)
Time difference of 0.01695704 secs
> cat("tolerancia")
tolerancia> print(.Machine$double.eps^(1/2))
[1] 1.490116e-08
```

Resultado con tolerancia de $1 * e^{-16}$:

```
> ################################################
> ####  TOLERANCIA 1e-16  ######
> start_time <- Sys.time()
> poli1 <- c(1.0, -5.0, -9.0, 155.0, -250.0)
> laguerre(poli1, 3.3,25,1e-16)
[1] 2
> end_time <- Sys.time()
>
> cat("Tiempo de ejecucion")
Tiempo de ejecucion> print(end_time - start_time)
Time difference of 0.01795292 secs
> cat("tolerancia")
tolerancia> print(.Machine$double.eps^(1/2))
[1] 1.490116e-08
>
> poli2 <- c(1.0, -5.0, -9.0, 155.0, -250.0)
> laguerre(poli2, -6.2,25,1e-16)
[1] -5
```

Resultado con tolerancia de $1 * e^{-32}$:

```
> ##############################################
> ####  TOLERANCIA 1e-32  ######
> start_time <- Sys.time()
> poli1 <- c(1.0, -5.0, -9.0, 155.0, -250.0)
> laguerre(poli1, 3.3,25,1e-32)
[1] 2
> end_time <- Sys.time()
>
> cat("Tiempo de ejecucion")
Tiempo de ejecucion> print(end_time - start_time)
Time difference of 0.01695609 secs
> cat("tolerancia")
tolerancia> print(.Machine$double.eps^(1/2))
[1] 1.490116e-08
>
> poli2 <- c(1.0, -5.0, -9.0, 155.0, -250.0)
> laguerre(poli2, -6.2,25,1e-32)
[1] -5
```

Convergencia método Laguerre

Si x es una raíz simple del polinomio $p(x)$, entonces el método de Laguerre converge cúbicamente siempre que el valor inicial x_0 este lo suficientemente cerca de la raíz. Por otro lado, si x es una raíz múltiple, entonces la convergencia solo es lineal. Este método también puede converger a una raíz compleja del polinomio observando que en los ejemplos presentados anteriormente notamos que en $x^4 - 5x^3 + 4x^2 - 3x + 2$ que tiene raíces imaginarias, el método muestra una solución de raíces y aún así converge por que debido a la raíz cuadrada de ser tomado en el cálculo de un anterior puede ser un número negativo. También se demuestra según el método que la falla de convergencia es muy rara, lo que convierte a este algoritmo en un buen candidato para la busqueda de raíces polinomiales.

Para finalizar y teniendo en cuenta todo lo visto durante las pruebas con diferentes polinomios y tolerancias se puede destacar que el metodo de Newton-Horner es mucho mas útil a la hora de encontrar raíces con parte imaginaria pero de igual manera también se debe tener en cuenta que la complejidad algorítmica para el algoritmo de Laguerre como para Newton-Horner es

$O(logn)$, además de observar que los resultados tanto de las pruebas como del polinomio del reto fueron iguales excepto para el polinomio $x^4 - 5x^3 + 4x^2 - 3x + 2$ el cual tenia cuatro raices (2 reales y 2 con parte imaginaria) de las cuales el método de Laguerre solo fue capaz de hallar una, entonces se puede determinar que ambos métodos son similares en cuanto a eficiencia y precisión pero que el método de Laguerre flaquea a la hora de encontrar raíces con parte imaginaria.

2. Algoritmo Brent

El método de Brent es un algoritmo de búsqueda de raíces el cual combina algunos métodos numéricos como: método de la bisección , el método de la secante y la interpolación cuadrática inversa, Así mismo este método es reconocido por tener la alta confiabilidad del método de bisección pero con una capacidad de convergería tan rápida como la de otros métodos menos confiables.

Este algoritmo se encarga de encontrar una raíz en un intervalo dado, y devuelve un flotante entre a y b por lo que esta función debe ser continua en dicho intervalo [a, b] a partir de un cambio de signo. Teniendo como ejemplo la función definida por: $f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27}$ en los intervalos $[0, 2]$ en el cual tomamos = $[0, 2]$

- $f(a_0)$=$-\frac{8}{27}$
- $f(b_0)$=2,37037037

Entonces las condiciones $f(a_0)f(b_0) < 0$ y $|f(b_0)||f(a_0)|$ están satisfechas. Por lo que como alternativa a la solución de este tipo de problemas decidimos usar una librearía de python muy robusta [from scipy import optimize] el cual nos permite acceder a un modulo [optimize.brentq] que nos permite

satisfacer la necesidad de encontrar una raíz a través del método ya mencionado.

Parámetros de la función: f: función que debe ser continua, en la cual $f(a)$ y $f(b)$ deben tener signos opuestos. Esta también recibe a y b [a,b] que son ambos escalares los cuales se encuentran en el intervalo de la función. Y por ultimo le mandamos como parámetro xtol es la tolerancia decimal que tendra la funcion al momento de devolver el valor aproximado.

Se hicieron múltiples pruebas con el método de Brent para verificar su funcionalidad utilizando diferentes polinomios y los resultados obtenidos se compararon con las raíces obtenidas en Wolfram Alpha

Algoritmo de Brent con el polinomio $f(x) = 6x^3 + 8x^2 - 5x + 2$ en el intervalo [-2,2]

```
Raiz:  -1.8732014759485645
```

resultado de Wolfram Alpha:

$$x \approx -1.87320147594856$$

en el polinomio $f(x) = 2x^4 + x^3 - 5x^2 + 9x$ en el intervalo [-3,-1]

```
Raiz:  -2.363415300218559
```

resultado de Wolfram Alpha:

$$x \approx -2.36341530021856$$

Convergencia método de Brent

Brent demostró que su método requiere como máximo de n^2 iteraciones.

Por lo que en el mejor de los casos donde la función f tiene un buen comportamiento, el método de Brent generalmente convergerá de forma lineal o en el peor de los casos en forma cuadrática.

Resultados y comparación del método con Newton-Raphson: los resultados evaluando el polinomio $f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27}$ con el método de Brent y una tolerancia de $10 * e^{-90}$ fueron:

Utilizando diferentes tolerancias como $1e-8, 1e-16, 1e-90$ en el algoritmo de Brent descubrimos que el resultado no varió por lo que se atribuye este error a la librería de optimize, el resultado obtenido fue:

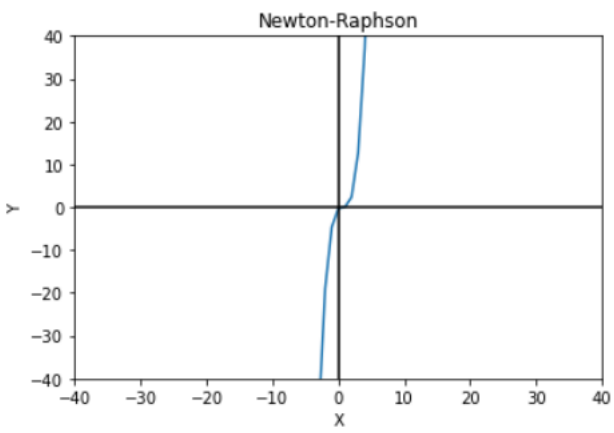
```
Raiz:  0.66666678545543223144420608150539919734001159667968 75
```

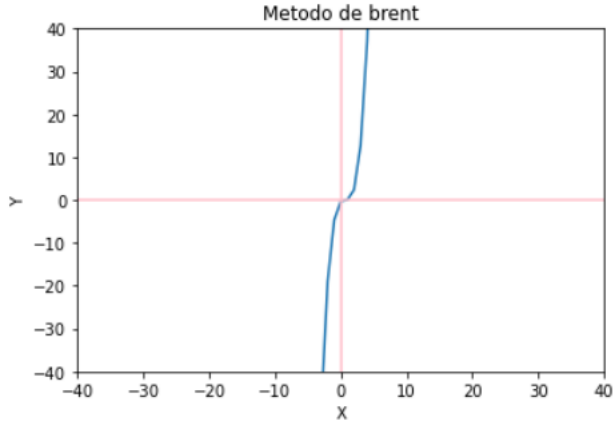
Mientras que para el método de Newton-Raphson el resultado fue:

```
Root = 0.6666707295628727
Number of iterations = 26
```

En comparación el método de Brent llegó a ser mas preciso y rápido que el método de Newton-raphson teniendo en cuenta que el resultado exacto de la raíz de este polinomio es de $\frac{2}{3}$ o $0,6$ periódico.

De igual forma, las gráficas sacadas a partir de cada método son comparadas a continuación:





Teniendo en cuenta esto y lo anteriormente planteado se puede determinar que el método de Brent en comparación al método de Newton-Raphson, es superior tanto en eficiencia como en precisión.

3. Óptima Aproximación Polinómica

El algoritmo de Remez es un algoritmo iterativo utilizado para encontrar aproximaciones simples a funciones. Se usa para producir un polinomio óptimo P(x) que se aproxima a una función dada en un intervalo dado. Este algoritmo converge a una función de error con n+2 niveles externos.

Este algoritmo permite que se construya un polinomio de grado N-ésimo que conduce a valores de error alternos, dados los n+2 puntos de referencia. Dados estos puntos de referencia $[x_1, x_2, ..., x_{n+2}]$, donde $[x_1 y x_{n+2}]$ son los puntos finales del intervalo de aproximación), estas ecuaciones deben resolverse:

$$P(x_1) - f(x_1) = +e$$

$$P(x_2) - f(x_2) = -e$$

.
.
.

$$P(x_{n+2}) - f(x_{n+2}) = e$$

Los lados de la derecha se alternan, es decir:

$$P_0 + P_1 x_1 + P_2 x_1^2 + P_3 x_1^3 + ... + P_n x_1^n - f(x_1 = +e$$

$$P_0 + P_1 x_2 + P_2 x_2^2 + P_3 x_2^3 + ... + P_n x_2^n - f(x_2 = -e$$

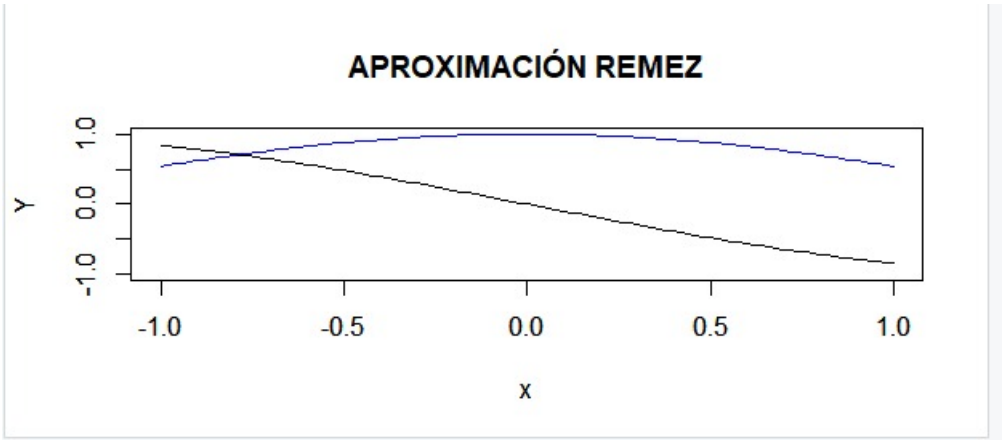
Dado que $x_1, ..., x_{n+2}$ son datos dados, todas sus potencias son conocidas, y $f(x_1), ..., f(x_{n+2})$ también son conocidos. Eso significa que las ecuaciones anteriores son solo $n+2$ ecuaciones lineales en las $n+2$ variables $P_0, P1, P_n$,y e. Dados los puntos de referencia $x_1, ..., x_{n+2}$, se puede resolver este sistema para obtener el polinomio P y el número e.

Ahora se realizaran una serie de pruebas con el método de Remez para verificar su precisión y funcionalidad.

El error para la aproximación de Remez con una tolerancia de $10 * e^{-90}$ para la funcion $cos(x)$ es:

	ErrorRelativo	ErroAbsoluto	ValorXpunto
1	2.376056e-192	2.375877e-104	0.01227185

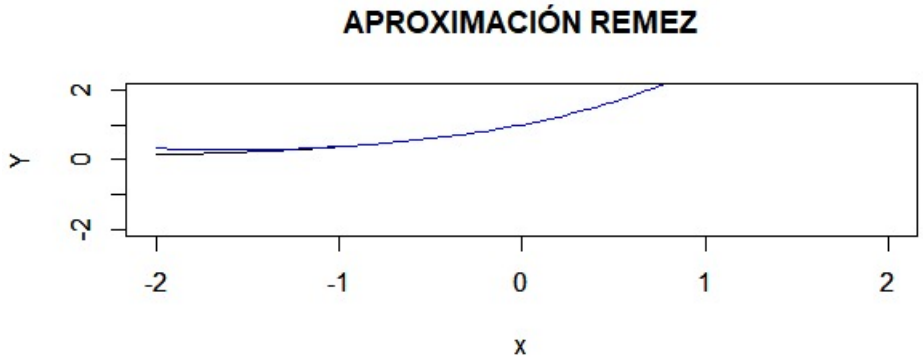
La aproximación de Remez gráficamente representada para la función $cos(x)$ es:



El error para la aproximación de Remez con una tolerancia de $10 * e^{-90}$ para la funcion e^x es:

	ErrorRelativo	ErroAbsoluto	ValorXpunto
1	2.321017e-190	2.349676e-102	0.01227185

La aproximación de Remez gráficamente representada para la función e^x es:



Ya teniendo en cuenta que el método desarrollado es funcional se procede a evaluar la función $f(x) = e^{sinx-cosx^2}$ para la cual los datos generados a

partir del método de Remez en el intervalo de $[-2^{-8}, 2^{-8}]$ con una precisión de $10 * e^{-90}$ fueron:

	x	y
1	-0.003906250	0.3664452
2	-0.001953125	0.3671616
3	0.000000000	0.3678794
4	0.001953125	0.3685987
5	0.003906250	0.3693193

Dichos datos permiten hallar el error relativo, absoluto y los valores por punto de la aproximación de Remez en comparación de la función los cuales dependiendo de la tolerancia permitirán saber cuanta precisión fue utilizada en el algoritmo y si esta aproximación es satisfactoria para la resolución del reto, las respectivas tolerancias son:

Errores con tolerancia de $1 * e^{-8}$:

	ErrorRelativo	ErroAbsoluto	ValorXpunto
1	1.198229e-24	4.462464e-19	0.01227185

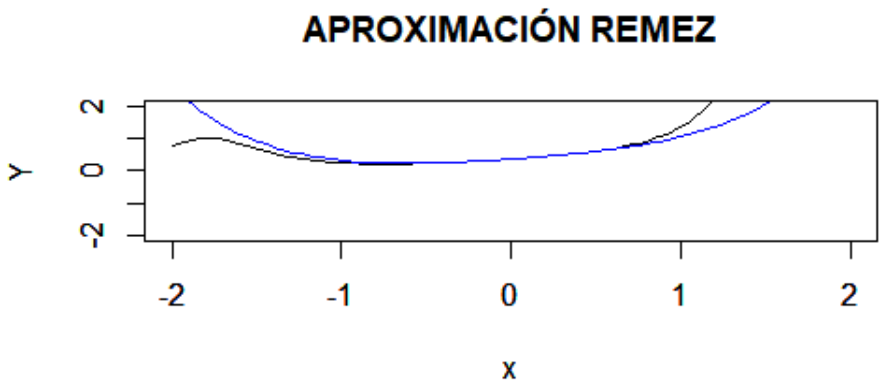
Errores con tolerancia de $1 * e^{-32}$:

	ErrorRelativo	ErroAbsoluto	ValorXpunto
1	1.198229e-72	4.462464e-43	0.01227185

Errores con tolerancia de $1 * e^{-90}$:

	ErrorRelativo	ErroAbsoluto	ValorXpunto
1	1.198229e-148	4.462464e-81	0.01227185

De igual manera se presenta la gráfica de la aproximación de Remez para la función $f(x) = e^{sinx-cosx^2}$:



Convergencia del algoritmo de Remez

Se observa que este algoritmo converge rápidamente, esto se da porque el algoritmo toma la precisión deseada y realiza una secuencia hasta que converja. Podemos notar que si los puntos de prueba están dentro del resultado correcto, tiene un mismo comportamiento que la función gráfica. Al tener una tolerancia de $1e^{-}90$ de acuerdo a la gráfica en la función $f(x) = e^{sinx-cosx^2}$, como esta función es derivable se dice que la convergencia es cuadrática;la aproximación que genera este algoritmo es muy parecida al comportamiento de la gráfica original, y de acuerdo a lo explicado anteriormente podemos notar que esta función se acerca a los puntos que están dentro del resultado correcto y permite así que converja rápidamente y cumpla con llegar a la presición deseada.

Aproximación de Taylor

Una serie de Taylor es una aproximación de funciones mediante una serie de potencias o funciones de potencias o suma de potencias enteras de polinomios como $(x-a)^n$ llamados términos de la serie, dicha suma se calcula a partir de las derivadas de la función para un determinado valor o punto a suficientemente derivable sobre la función y un entorno sobre el cual converja la serie.Si esta serie está centrada sobre el punto cero, $a = 0$.

Esta aproximación tiene como ventajas:
-La derivación de una de estas series se puede realizar término a término
-Calcula valores aproximados de funciones
-Es posible calcular la optimidad de la función

La serie de Taylor de una función f real o compleja $f(x)$ infinitamente diferenciable en el entorno de un número real o complejo a es la siguiente serie de potencias:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + ...$$

Escrito de una manera como la siguiente suma:

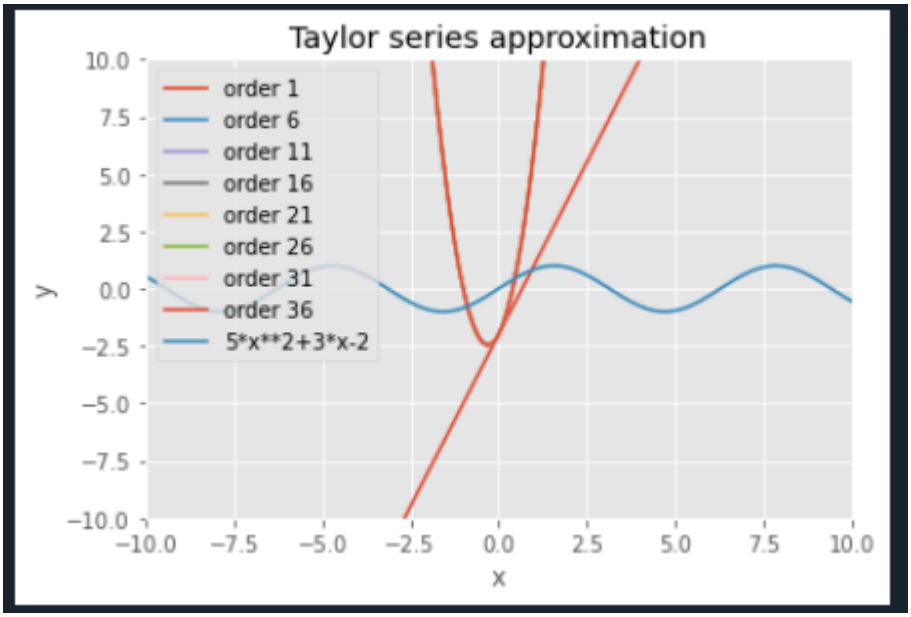
$\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$, tenemos qué:
-$n!$ es el factorial de n
-$f^n(a)$ denota la n-ésima derivada de f para el valor a de la variable respecto de la cual se deriva

Se hicieron pruebas para observar el comportamiento de este algoritmo en distintas funciones: Como primera función se usó $f(x) = 5x^2 + 3x - 2$

-Series de Taylor en esta función

```
Taylor expansion at n=1 3*x - 2
Taylor expansion at n=6 5*x**2 + 3*x - 2
Taylor expansion at n=11 5*x**2 + 3*x - 2
Taylor expansion at n=16 5*x**2 + 3*x - 2
Taylor expansion at n=21 5*x**2 + 3*x - 2
Taylor expansion at n=26 5*x**2 + 3*x - 2
Taylor expansion at n=31 5*x**2 + 3*x - 2
Taylor expansion at n=36 5*x**2 + 3*x - 2
```

-Serie de Taylor de la función de forma gráfica

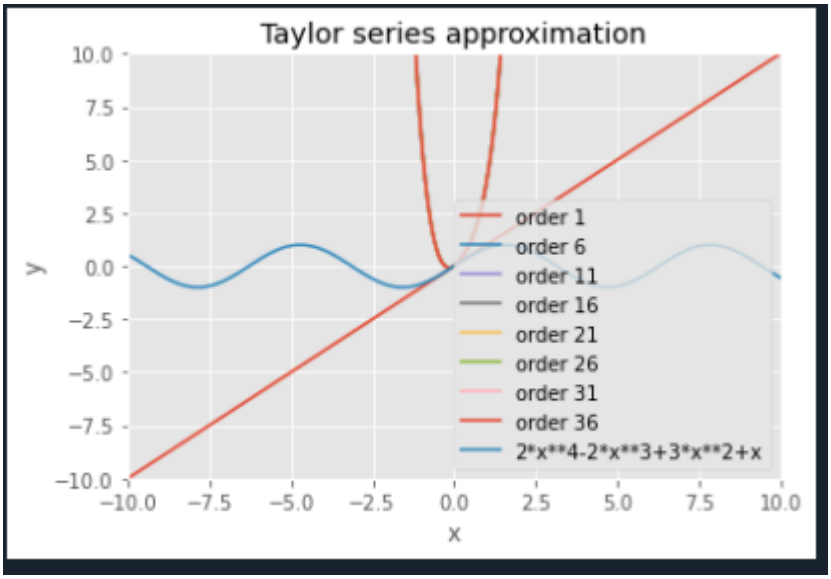


Como segunda función se usó $f(x) = 2x^4 - 2x^3 + 3x^2 + x$

-Series de Taylor en esta función

```
Taylor expansion at n=1 x
Taylor expansion at n=6 2*x**4 - 2*x**3 + 3*x**2 + x
Taylor expansion at n=11 2*x**4 - 2*x**3 + 3*x**2 + x
Taylor expansion at n=16 2*x**4 - 2*x**3 + 3*x**2 + x
Taylor expansion at n=21 2*x**4 - 2*x**3 + 3*x**2 + x
Taylor expansion at n=26 2*x**4 - 2*x**3 + 3*x**2 + x
Taylor expansion at n=31 2*x**4 - 2*x**3 + 3*x**2 + x
Taylor expansion at n=36 2*x**4 - 2*x**3 + 3*x**2 + x
```

-Serie de Taylor de la función de forma gráfica

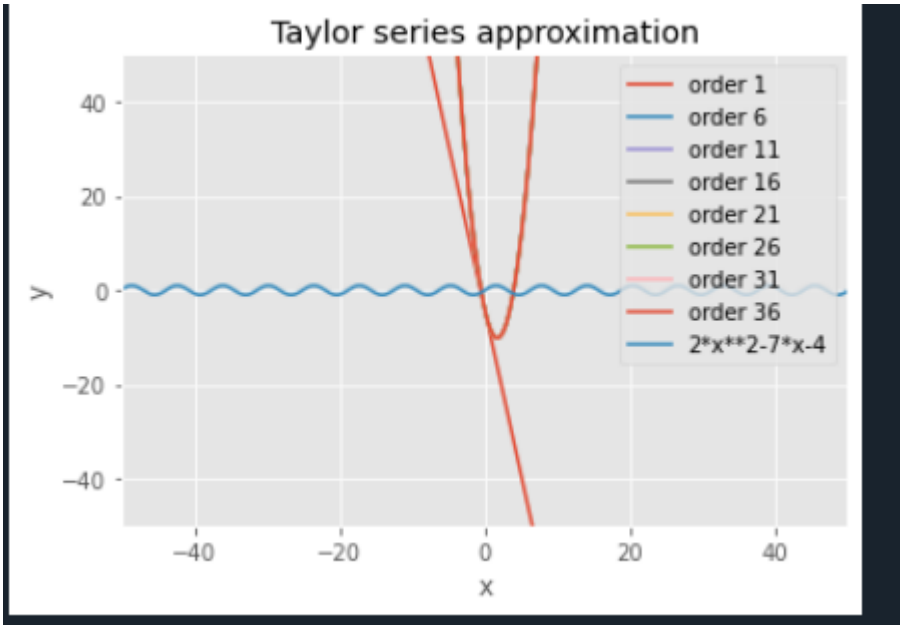


Como tercera función se usó $f(x) = 2x^2 - 7x - 4$

-Series de Taylor en esta función

```
Taylor expansion at n=1 -7*x - 4
Taylor expansion at n=6 2*x**2 - 7*x - 4
Taylor expansion at n=11 2*x**2 - 7*x - 4
Taylor expansion at n=16 2*x**2 - 7*x - 4
Taylor expansion at n=21 2*x**2 - 7*x - 4
Taylor expansion at n=26 2*x**2 - 7*x - 4
Taylor expansion at n=31 2*x**2 - 7*x - 4
Taylor expansion at n=36 2*x**2 - 7*x - 4
```

-Serie de Taylor de la función de forma gráfica

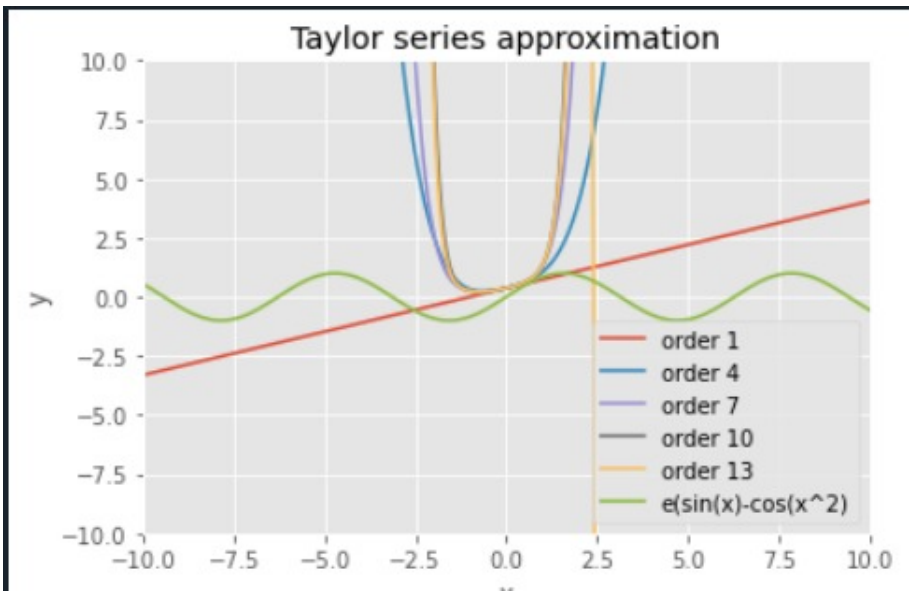


Esta introducción a las series de Tayor se realizó para mostrar de acuerdo a la definición como actúa el método en la función $f(x) = e^{sinx-cosx^2}$:

-En esta primera imagen se observa como actúa la serie de Taylor en la función

```
In [1]: runfile('C:/Users/esteb/Downloads/Taylor.py', wdir='C:/Users/esteb/Downloads')
Taylor expansion at n=1 0.367879441171442*x + 0.367879441171442
Taylor expansion at n=4 0.137954790439291*x**4 + 0.183939720585721*x**2 + 0.367879441171442*x
+ 0.367879441171442
Taylor expansion at n=7 0.00408754934634936*x**7 + 0.0904370292879796*x**6 +
0.159414424507625*x**5 + 0.137954790439291*x**4 + 0.183939720585721*x**2 +
0.367879441171442*x + 0.367879441171442
Taylor expansion at n=10 0.0142627289183667*x**10 + 0.0184588537942285*x**9 +
0.00964406173904302*x**8 + 0.00408754934634936*x**7 + 0.0904370292879796*x**6 +
0.159414424507625*x**5 + 0.137954790439291*x**4 + 0.183939720585721*x**2 +
0.367879441171442*x + 0.367879441171442
Taylor expansion at n=13 -0.00148211284625082*x**13 - 0.00232659314512354*x**12 +
0.00192698754899327*x**11 + 0.0142627289183667*x**10 + 0.0184588537942285*x**9 +
0.00964406173904302*x**8 + 0.00408754934634936*x**7 + 0.0904370292879796*x**6 +
0.159414424507625*x**5 + 0.137954790439291*x**4 + 0.183939720585721*x**2 +
0.367879441171442*x + 0.367879441171442
```

-En la segunda imagen se observa como actúa la serie de Taylor de forma gráfica



adiferenciable noindeninfinitt

Referencias

- Teoría de la aproximación - es.LinkFang.org. (2020). Retrieved 11 September 2020, from https://es.linkfang.org/wiki/Teor
- numpy.polynomial.laguerre.Laguerre — NumPy v1.19 Manual. (2020). Retrieved 11 September 2020, from numpy.org/doc/stable/reference/generated/numpy.polynomial.laguerre.Laguerre.html
- Kiusalaas, J. (2013). Numerical methods in engineering with Python 3. Cambridge: Cambridge University Press.
- (2020). Retrieved 11 September 2020, from https://es.khanacademy.org/computing/computer science/algorithms/asymptotic-notation/a/big-o-notation
- (2020). Retrieved 11 September 2020, from https://www.ugr.es/ acanada/docencia/matematicas/definitivoAlejandraTorresMartinezTFG.pdf
- Burden, R., Faires, J. (2011). Analisis numerico. Mexico: Cengage Learning.

- El método de Laguerre - Laguerre's method - qwe.wiki. (2020). Retrieved 12 September 2020, from https://es.qwe.wiki/wiki/Laguerre
- El método de Brent - Brent's method - qwe.wiki. (2020). Retrieved 12 September 2020, from https://es.qwe.wiki/wiki/Brent's$_{m}ethod$