UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales
Departamento de Computación
Métodos Numéricos

TRABAJO PRÁCTICO NÚMERO 1

Efectividad de métodos numéricos para hallar ceros de funciones.

Alumnos:

Izcovich, Sabrina | sizcovich@gmail.com
Otero, Fernando | fergabot@gmail.com
Vita, Sebastián | sebastian_vita@yahoo.com.ar

Palabras Clave:

Newton - Secante - Bisección - Estimación

Resumen:

Este trabajo consiste en un análisis de métodos numéricos utilizados para hallar ceros de funciones. El objetivo del análisis es estimar los parámetros necesarios de una función DΓG para estudiar datos de manera certera y eficiente. Para realizar nuestra experimentación, incluimos la utilización de los Métodos de Bisección, Newton y Secante. Luego de diversas pruebas en las que combinamos distintos paquetes de datos, criterios de parada y márgenes de error, hemos concluido que los métodos numéricos son muy acertados siempre y cuando los parámetros sean los correctos para su buen funcionamiento.

Índice

1. Introducción teórica

A lo largo del trabajo práctico, hemos experimentado con distintas herramientas necesarias para obtener resultados precisos que nos permitieran sacar conclusiones significativas y sólidas. Este tipo de herramientas consistieron, generalmente, en métodos numéricos, fórmulas de muestreo probabilístico y datos aleatorios. En esta sección, enunciaremos los utensilios utilizados.

- **Método de Newton:** Esta técnica numérica para resolver un problema de búsqueda de raíces f(x) = 0 es una de las más poderosas y conocidas pues permite lograr una convergencia más rápida (cuadrática) que la que ofrecen otros tipos de iteración funcional. Dicho método, basado en los polinomios de Taylor, genera la sucesión P_n definida por $P_n = P_{n-1}$ - $\frac{f(P_{n-1})}{f'(P_{n-1})}$, para n $\leq$ 1.
- **Método de Bisección:** Este método consiste en hallar un intervalo [a,b] en el que la función f resulte continua y f(a).f(b)<0. Basado en el teorema del valor intermedio, la idea del mismo es realizar una búsqueda binaria para hallar el punto p perteneciente a (a,b) en el que f(p)=0. Su convergencia está asegurada y la misma es lineal.
- **Método de Secante:** El método de secante altera el método de Newton aproximando la derivada de la función afectada utilizando la idea del teorema del valor medio. De esta manera, no resulta necesaria una derivada de la función. Esto último afecta a la convergencia que resulta superlineal en dicho método.
- **Criterios de parada:** Los criterios de parada utilizados fueron los siguientes (para bisección, se remplazaron X_n por b, X_{n-1} por a y f(X_n) por c):
  - $|X_n - X_{n-1}| < \epsilon$
  - $\frac{|X_n - X_{n-1}|}{|X_{n-1}|} < \epsilon$
  - $|f(X_n)| < \epsilon$
  - $|f(X_n) - f(X_{n-1})| < \epsilon$
  - $\frac{|f(X_n) - f(X_{n-1})|}{|f(X_{n-1})|} < \epsilon$
- **Cambio de base de exponenciales:**

$$y = a^x \Leftrightarrow y = e^{mx} \text{ (con } y = (e^m)^x).$$

$$\text{Luego, } a = e^m \Leftrightarrow ln(a) = ln(e^m) \Leftrightarrow ln(a) = mln(e).$$

$$\text{Como } ln(e) = 1, \text{ queda } ln(a) = m \Rightarrow \begin{cases} \text{si a } > 1 \Rightarrow \text{m } > 0 \\ \text{si } 0 < \text{a } < 1 \Rightarrow \text{m } < 0 \end{cases}$$

2. Desarrollo

Análisis previo:

El objetivo del trabajo práctico es hallar las combinaciones de métodos numéricos de mayor eficiencia que, junto con los parámetros más acordes, otorguen los resultado más cercanos posibles a β dado un paquete de datos determinado. Para cumplir con dicho objetivo, utilizamos los métodos:

- Newton: El uso de este método era requerido por la cátedra. Dicho método tiene su importancia en la convergencia cuadrática ya que es la mejor entre los métodos analizados en clase. Sin embargo, dicha convergencia no está garantizada por un teorema de convergencia global como podría estarlo el método de bisección. Por lo tanto, es necesario partir de una aproximación inicial próxima a la raíz buscada para que el método converja debidamente. Este método resulta muy interesante pero a la vez muy costoso dado que requiere de la derivada de la función a utilizar, que en nuestro caso resultó ser compleja.

- Secante: Debido a que la convergencia de secante es superlineal, elegimos el método para comparar el tiempo de ejecución del mismo con el de los demás. Por otro lado, nos pareció interesante observar el margen de error que mantiene con Newton al aproximar la derivada con el Polinomio de Taylor para lograr obtener resultados relevantes y encontrar una justificación para ellos.
- Bisección: Utilizamos este método dado que nos resultó importante asegurarse convergencia en todo momento a pesar de que ésta sea lineal. También, lo utilizamos para aproximar, tanto el P_0 en Newton, como el P_0 y P_1 en Secante.

El procedimiento de nuestro trabajo práctico fue el siguiente:

Elegimos, en primer lugar, la función que nos resultó más valorable para analizar. Ésta resultó ser la (5) debido a que no contiene la función *log*, por lo tanto, se deben realizar menos operaciones para resolverla. Ésto logra que se acarree menos error y nos otorga mayor precisión en los resultados.

En segundo lugar, pensamos en analizar matemáticamente la función con el fin de simplificarla lo máximo posible. Al realizar esto, nos encontramos con que toda conclusión posible dependía del paquete de datos ingresado por lo que el análisis de función resultaba inútil. Al simplificar de diversas maneras las funciones utilizadas, los márgenes de error hallados eran despreciables. Por ejemplo, estudiamos la diferencia entre los valores de $\mathcal{M}$ y $\widehat{\mathcal{M}}$ calculados por la computadora y simplificando la multiplicación y división por n manualmente. Si bien el resultado de dicha experimentación demostró una leve mejoría en la cantidad de iteraciones y el error final, decidimos no utilizarlo dado que éste no resultaba demasiado significante por lo que preferimos evitar posibles confusiones durante la implementación.

Luego, procedimos a derivar la función con sus parámetros correspondientes:

Nuestra función:

$$\frac{\mathcal{M}(2\beta)}{\mathcal{M}^2(\beta)} = 1 + \beta(\mathcal{R}(\beta) - \mathcal{R}(0)) \Leftrightarrow 0 = 1 + \beta(\mathcal{R}(\beta) - \mathcal{R}(0)) - \frac{\mathcal{M}(2\beta)}{\mathcal{M}^2(\beta)}$$

donde

$$\mathcal{M}(s) = \frac{1}{n}\sum_{i=1}^{n} x_i^s\,, \qquad \widehat{\mathcal{M}}(s) = \frac{1}{n}\sum_{i=1}^{n} x_i^s \log(x_i)\,, \qquad \mathcal{R}(s) = \frac{\widehat{\mathcal{M}}(s)}{\mathcal{M}(s)}$$

Al derivar término a término, se obtuvo:

$$\mathcal{M}'(s) = \frac{1}{n}\sum_{i=1}^{n} x_i^s \log(x_i)\,, \qquad \widehat{\mathcal{M}}'(s) = \frac{1}{n}\sum_{i=1}^{n} x_i^s \log(x_i)^2\,, \qquad \mathcal{R}'(s) = \frac{\widehat{\mathcal{M}}'(s)}{\mathcal{M}'(s)}$$

Por lo tanto, la derivada resultó ser:

$$\mathcal{R}(\beta) + \beta(\mathcal{R}'(\beta)) - \mathcal{R}(0) - \frac{2\mathcal{M}'(2\beta)\mathcal{M}^2(\beta) - 2\mathcal{M}(2\beta)\mathcal{M}(\beta)\mathcal{M}'(\beta)}{\mathcal{M}^4(\beta)}$$

Implementación: El paso siguiente consistió en programar en C++ los métodos utilizados, la función junto con su derivada y el programa para realizar pruebas dado un paquete de datos ingresado. Al implementar las fórmulas, utilizamos el cambio de base de funciones exponenciales (explicado en la *Introducción teórica*) para simplificarlas.
Nos decidimos por el uso de una lista para manipular las muestras dado que, en otro caso, se hubiese necesitado conocer el tamaño de la muestra (como por ejemplo para usar un vector) y no utilizamos

ninguna operación que una lista no pudiera resolver.
El *main.cpp* consistió en un menú necesario para almacenar los datos que serían, más tarde, utilizados por las funciones para resolver los métodos. Para realizarlo, nos limitamos a utilizar herramientas conocidas (if, while, etc.) y $printf$'s.
En $formulas.cpp$, escribimos las fórmulas utilizadas a lo largo del trabajo práctico. Tanto la función principal para hallar β y su derivada como las funciones que definen a $\tilde{\lambda}$ y $\tilde{\sigma}$ se hallan en este archivo. Para que dichas fórmulas fueran legibles, utilizamos las funciones $\mathcal{M}$, $\widehat{\mathcal{M}}$ y $\mathcal{R}$ para realizar las operaciones necesarias y luego terminar por unirlas correctamente.
En *metodos.cpp* implementamos los métodos utilizados para hallar los ceros de función. Éstos están programados de manera tal a recibir los parámetros ingresados por el menú principal y realizar las operaciones necesarias a partir de ellos. Para calcular el tiempo de ejecución de cada prueba experimental introdujimos un clock en milisegundos. Utilizamos, también, un switch para calcular el error experimental.

Una vez encontrado $\tilde{\beta}$, éste es reemplazado en las funciones que fuimos despejando de la siguiente manera para hallar $\tilde{\sigma}$ y $\tilde{\lambda}$:

$$\begin{aligned}
\tilde{\sigma} &= \left(\frac{\sum_{i=1}^{n} x_i^{\tilde{\beta}}}{n\tilde{\lambda}}\right)^{1/\tilde{\beta}} \\
\tilde{\lambda} &= \left[\tilde{\beta}\left(\frac{\sum_{i=1}^{n} x_i^{\tilde{\beta}} \log x_i}{\sum_{i=1}^{n} x_i^{\tilde{\beta}}} - \frac{\sum_{i=1}^{n} \log x_i}{n}\right)\right]^{-1} \\
&= \tilde{\beta}^{-1}\left(\frac{\sum_{i=1}^{n} x_i^{\tilde{\beta}} \log x_i}{\sum_{i=1}^{n} x_i^{\tilde{\beta}}} - \frac{\sum_{i=1}^{n} \log x_i}{n}\right)^{-1} \\
&= \tilde{\beta}^{-1}\left(\frac{n\sum_{i=1}^{n} x_i^{\tilde{\beta}} \log x_i - \sum_{i=1}^{n} x_i^{\tilde{\beta}} \sum_{i=1}^{n} \log x_i}{n\sum_{i=1}^{n} x_i^{\tilde{\beta}}}\right)^{-1} \\
&= \frac{1}{\tilde{\beta}}\left(\frac{n\sum_{i=1}^{n} x_i^{\tilde{\beta}}}{n\sum_{i=1}^{n} x_i^{\tilde{\beta}} \log x_i - \sum_{i=1}^{n} x_i^{\tilde{\beta}} \sum_{i=1}^{n} \log x_i}\right) \\
&= \left(\frac{n\sum_{i=1}^{n} x_i^{\tilde{\beta}}}{\tilde{\beta}n\sum_{i=1}^{n} x_i^{\tilde{\beta}} \log x_i - \tilde{\beta}\sum_{i=1}^{n} x_i^{\tilde{\beta}} \sum_{i=1}^{n} \log x_i}\right) \\
0 &= \frac{\tilde{\beta}}{n}\sum_{i=1}^{n} \log x_i - \log\sum_{i=1}^{n} x_i^{\tilde{\beta}} + \log(n\tilde{\lambda}) - \psi(\tilde{\lambda})
\end{aligned}$$

donde $\tilde{\Theta} = (\tilde{\sigma}, \tilde{\beta}, \tilde{\lambda})$

Problemas hallados: Al realizar dicho programa nos encontramos con distintas complicaciones: nuestra mayor complicación resultó ser que, si bien encontrábamos una buena aproximación de β, $\tilde{\lambda}$ y $\tilde{\sigma}$ resultaban desacertadas del valor original.
Luego de distintas pruebas y un análisis exhaustivo, nos percatamos de que la función $\tilde{\lambda}$ podía ser reescrita de manera tal que quedara eliminado todo tipo de potencias y se lograra la mínima cantidad de operaciones posibles para su resolución (vistas más arriba). Del mismo modo, para poder realizar ciertos cálculos no permitidos en variables de tipo $TFloat$ (como por ejemplo potencia), debíamos pasar la variable a *double* y luego nuevamente a $TFloat$ dentro de las fórmulas mismas. Al tener que realizar este cambio de tipo en numerosas iteraciones, se arrastraba una gran cantidad de error que derivaba en la obtención de resultados imprecisos. Para resolver esto último, decidimos utilizar la

función *exponencial* definida en $TFloat.h$. De esta manera, no resultaba necesario el cambio de tipo por lo que los valores devueltos resultaban más certeros. Por lo tanto, evitamos los errores de casteo que hacían diferir el valor original con la estimación en un 300 %. Por ejemplo, en vez de encontrar $\lambda = 3$, encontrábamos $\lambda = 0{,}08$. Luego de los cambios realizados, la precisión pasó a ser del 85 % aproximadamente.

Modo de uso del programa: Nuestro programa corre en Windows 7 y Ubuntu. Para compilarlo, recomendamos usar g++. Para ejecutarlo es necesario compilar $formulas.cpp$, $metodos.cpp$, $TFloat.cpp$ y, por último, $main.cpp$. Luego, se debe abrir el ejecutable del main.
Una vez abierta la interfaz del programa, se deben completar paso a paso los datos pedidos por éste necesarios para su funcionamiento. Entre los parámetros requeridos se necesitan la precisión de la mantisa, el tipo de error que se desee utilizar (ver criterios de parada en *Introducción teórica*), el máximo error aceptado, la cantidad máxima de iteraciones que el programa debe realizar (conocido como "criterio de parada pesimista") y, finalmente, el Método que se desee utilizar (entre Bisección, Newton y Secante).
En el caso de elegir el método de Newton, se debe seleccionar entre insertar manualmente un p_0 o que éste sea aproximado por el Método de Bisección. En el caso en el que el método Secante sea el escogido, se deben ingresar los valores p_0 y p_1. Por último, al elegir el Método de Bisección, se requiere determinar un valor positivo y uno negativo para que éste pueda hallar el cero que se encuentra entre esos dos valores. En el caso en el que el valor ingresado como positivo o negativo no posea dicho signo, éste volverá a ser pedido. Al finalizar, el programa otorgará los valores de $\tilde{\lambda}$, $\tilde{\beta}$ y $\tilde{\sigma}$ como también el tiempo que tardó en hallar $\tilde{\beta}$, las iteraciones realizadas y el error final.

Recuperación de resultados: Para verificar la correctitud de nuestro programa utilizamos los paquetes de datos otorgados por la cátedra cuyos parámetros eran conocidos. De esta manera, corroboramos los algoritmos realizados y descartamos experimentos innecesarios. Al realizar varias iteraciones llegamos a la conclusión de que los resultados de nuestro programa resultaban satisfactorios para la mayoría de las pruebas realizadas.
A partir de ese entonces, comenzamos a evaluar resultados de manera organizada variando los parámetros paulatinamente. Completamos una planilla en la que congelamos todos los parámetros salvo uno. Por ejemplo, evaluamos tres mantisas distintas (t=51, t=25 y t=15 en la mayoría de los casos) para cada método utilizando como error máximo 0,0000000001, 500 iteraciones, $x\_2\_62\_35.txt$ como muestra, un p_0 y un p_1 determinados. Realizamos lo mismo cambiando los tipos de error, p_0 y p_1 para poder incializar las conclusiones de efectividad. De estas pruebas retuvimos los parámetros a estimar, la duración de la ejecución del programa, la cantidad de iteraciones realizadas por el mismo y el error final obtenido para cada una de ellas. A partir de esto, obtuvimos los resultados expuestos posteriormente.

3. Resultados

Decidimos presentar los resultados con gráficos de histogramas dado que es una herramienta muy simple a la hora de analizarlos y éstos pueden ser vistos de manera clara. Para ello, utilizamos *Matlab* aplicando las funciones otorgadas por la cátedra ($dibujarHistyAjuste.m$, $GGDpdf\_c.m$ y $leer\_datos.m$).
Los tests fueron realizados con los archivos $x_2\_62\_35.txt$, $x\_15\_9\_3.txt$ y $X1.txt$. Utilizamos como mantisa los valores del conjunto t=15,18,25,40,51 y los criterios de parada enunciados en la *Introducción teórica*. Como error máximo permitido, utilizamos los valos de ϵ = 2.88e-005, 1e-10, 0.001, 0.5. Por último, las cantidades máximas de iteraciones evaluadas fueron 4000, 2000 y 500.

Los resultados obtenidos fueron los siguientes:

Analicemos, en primer lugar, el siguiente conjunto de datos:

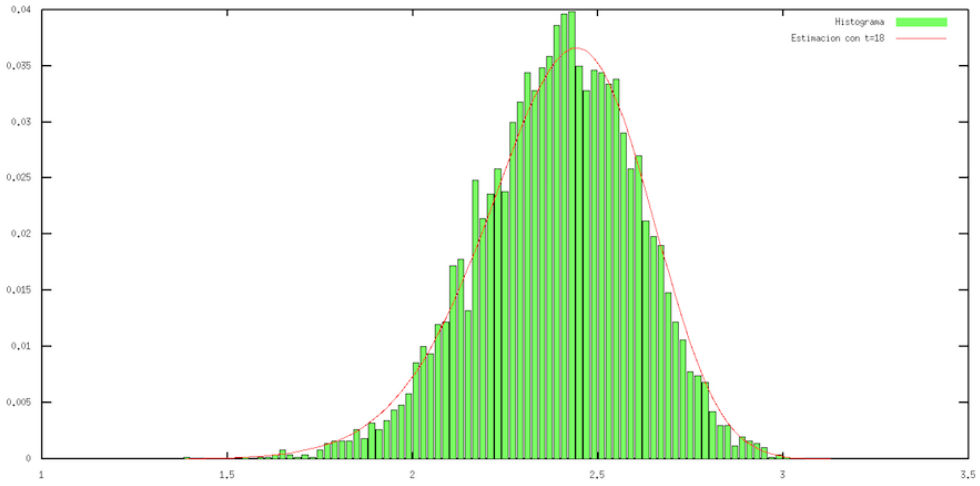

Figura 1: Estimación del paquete de datos x_2_62_35 con Bisección
Precisión t = 18

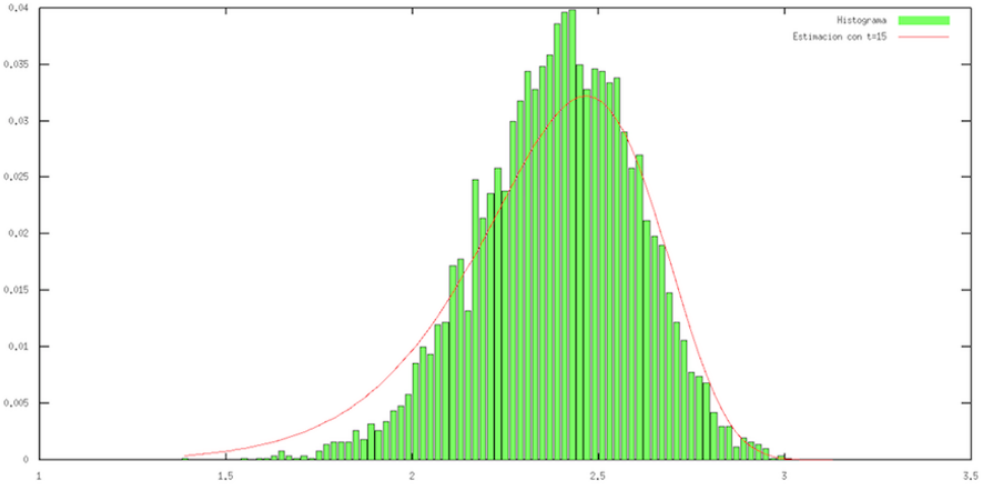

Figura 2: Estimación del paquete de datos x_2_62_35 con Newton
Precisión t = 15

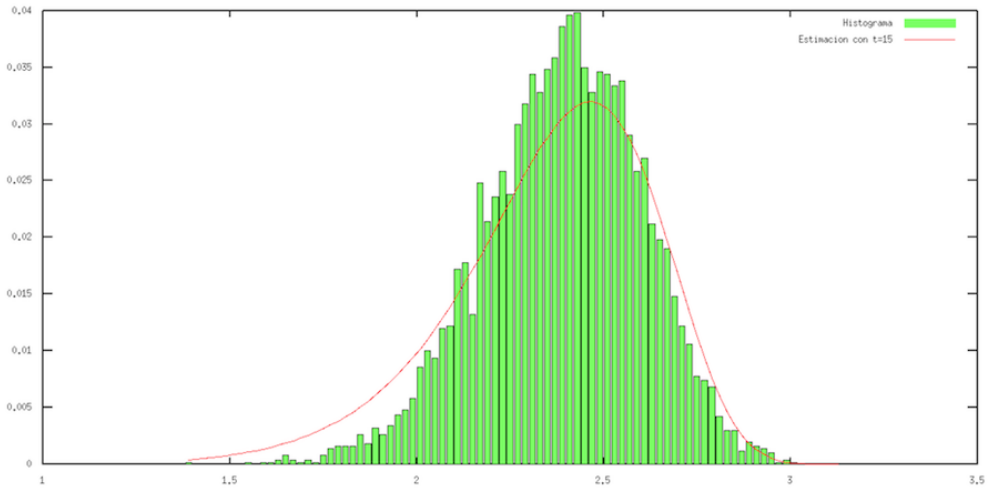

Figura 3: Estimación del paquete de datos x_2_62_35 con Secante
Precisión t = 15

En estos gráficos se evaluaron los tres métodos con lo siguiente:

- Mantisa: 15 para Newton y Secante, 18 para Bisección
- Criterio de parada: $|f(X_n)| < \epsilon$
- Error máximo aceptable: 0.0000000001
- Máxima cantidad de iteraciones: 500
- Muestra evaluada: x_2_62_35.txt

Resultados observados: En el caso del Método de Bisección, podemos observar que la curva de estimación supera el valor de 0.035 (siendo 0.04 el máximo valor del histograma) por lo que se asemeja considerablemente al histograma de datos. Por otro lado, la curva de estimación del Método de Newton no alcanza el valor 0.035 sino que su máximo se mantiene en un valor cercano a 0.032. Dicha curva se encuentra ligeramente desfasada a la derecha en comparación con el histograma. Por último, la curva de estimación del Método de Secante posee su máximo en el valor 0.0325. Esta curva se encuentra también sutilmente desfasada hacia la derecha de acuerdo al histograma de datos.

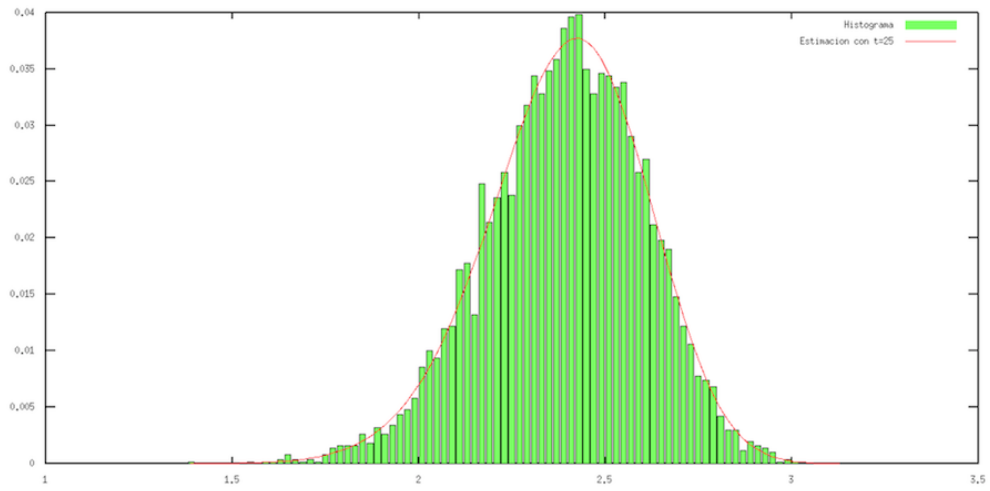

Figura 4: Estimación del paquete de datos x_2_62_35 con Bisección
Precisión t = 25

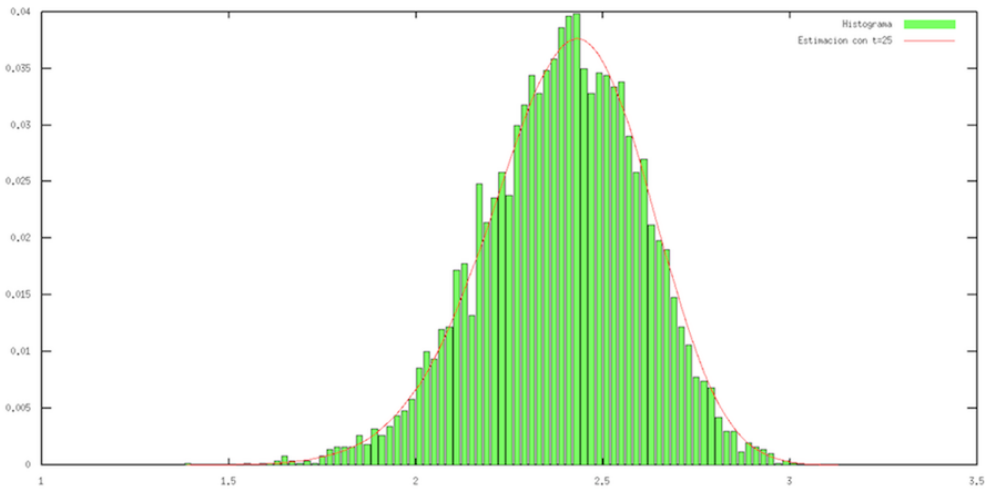

Figura 5: Estimación del paquete de datos x_2_62_35 con Newton
Precisión t = 25

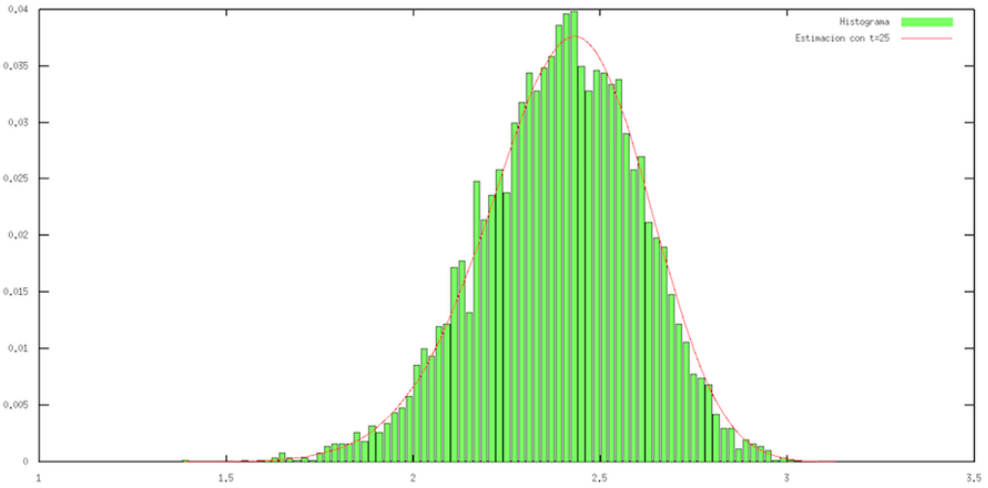

Figura 6: Estimación del paquete de datos x_2_62_35 con Secante
Precisión t = 25

En estos gráficos se evaluaron los tres métodos con lo siguiente:

- Mantisa: 25
- Criterio de parada: $|f(X_n)| < \epsilon$
- Error máximo aceptable: 0.0000000001
- Máxima cantidad de iteraciones: 500
- Muestra evaluada: x_2_62_35.txt

Resultados observados: En primer lugar, podemos observar que el máximo alcanzado por la curva de estimación del Método de Bisección supera el valor de 0.035 alcanzando casi a 0.038. Esta curva se mantiene semejante al histograma de datos en todo su alcance. Luego, podemos ver que en el caso del Método de Newton, el máximo valor alcanzado es cercano al 0.038. Del mismo modo, la curva se mantiene semejante al histograma en todo su alcance, marcando una leve cercanía del lado izquierdo a los valores del interior del histograma. Por último, en el Método de Secante podemos

ver que el máximo alcanzado por la curva es de 0.038. La curva permanece similar al histograma de datos en toda su longitud.

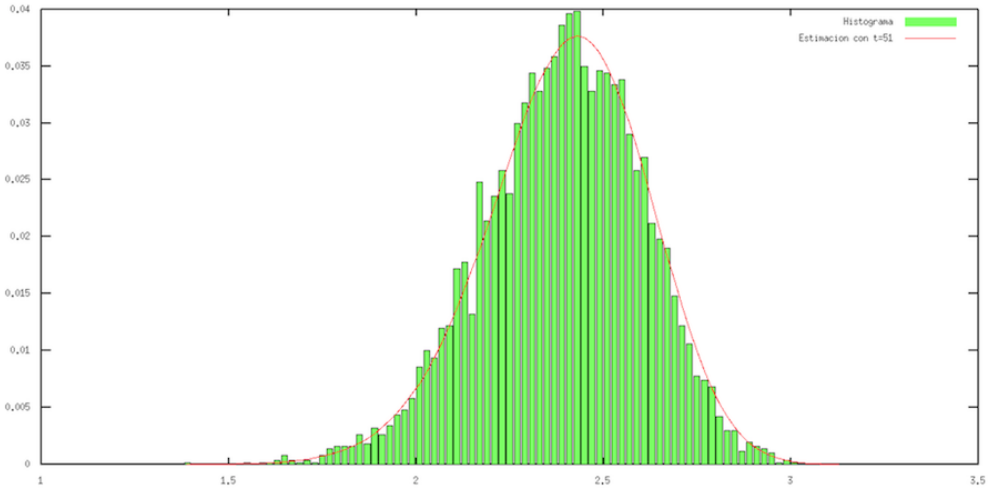

Figura 7: Estimación del paquete de datos x_2_62_35 con Bisección
Precisión t = 51

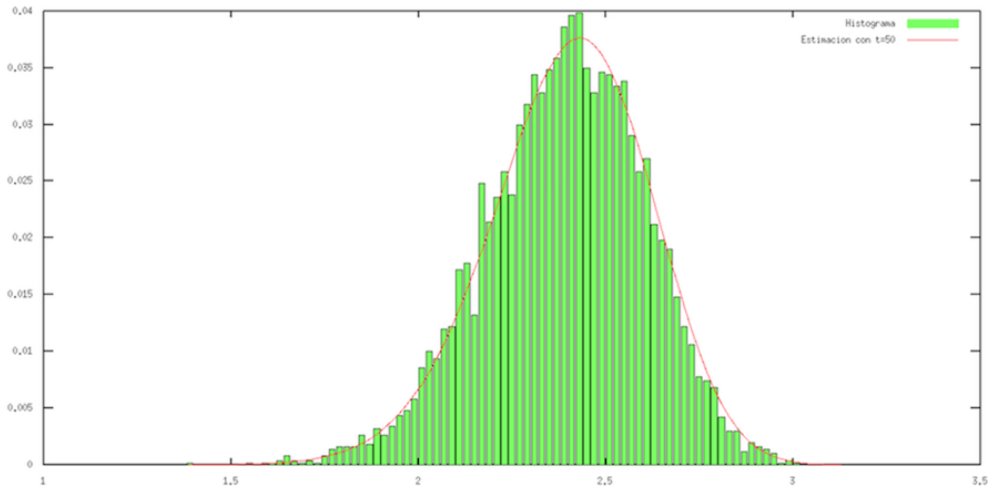

Figura 8: Estimación del paquete de datos x_2_62_35 con Newton
Precisión t = 50

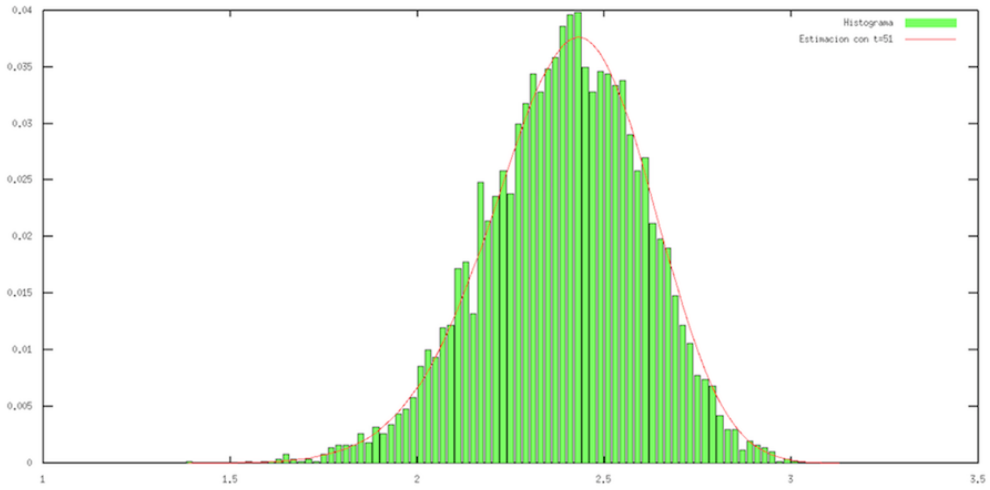

Figura 9: Estimación del paquete de datos x_2_62_35 con Secante
Precisión t = 51

En estos gráficos se evaluaron los tres métodos con lo siguiente:

- Mantisa: 51 para Bisección y Secante, 50 para Newton
- Criterio de parada: $|f(X_n)| < \epsilon$
- Error máximo aceptable: 0.0000000001
- Máxima cantidad de iteraciones: 500
- Muestra evaluada: x_2_62_35.txt

Resultados observados: En primer lugar, podemos observar que el máximo alcanzado por la curva de estimación del Método de Bisección es 0.038. Esta curva se mantiene semejante al histograma de datos en todo su alcance. Luego, podemos ver que en el caso del Método de Newton, el máximo valor alcanzado es cercano al 0.038. Del mismo modo, a curva se mantiene semejante al histograma en todo su alcance. Por último, en el Método de Secante podemos ver que el máximo alcanzado por la curva es de 0.038. La curva permanece similar al histograma de datos en toda su longitud.

Otros resultados obtenidos fueron los siguientes:

- Al probar los métodos con una precisión menor a 15, los resultados obtenidos fueron Nan en la gran mayoría de los casos.
- Al variar los criterios de parada, los resultados se mantuvieron muy semejantes al criterio de parada analizado anteriormente.

4. Discusión

Para lograr relativizar los resultados, decidimos crear gráficos comparativos en los que se estudiaran únicamente esos parámetros. Para ello, tomamos 3 datos de nuestra muesta (uno con precisión t = 18, uno con t = 25 y otro con t = 51). Dichos gráficos resultaron ser los siguientes:

Comparación de la cantidad de iteraciones respecto de la precisión de la mantisa.

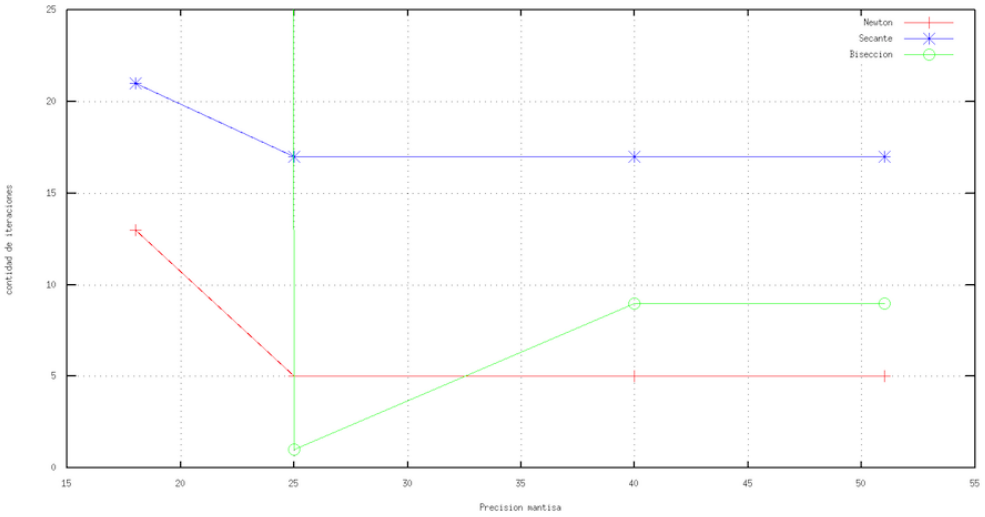

Figura 10: Comparación de la cantidad de iteraciones con respecto a la precisión de la mantisa. Paquete de datos x_2_62_35.txt

- Error máximo aceptable: 0.0000288
- Criterio de parada: $|f(X_n)| < \epsilon$
- p_0 en Newton: 10
- p_0 en Secante: 10
- p_1 en Secante: 11
- Intervalo en Bisección: [10,3]
- Máxima cantidad de iteraciones: 4000
- Muestra evaluada: x_2_62_35.txt

Observaciones: En esta figura se compara la cantidad de iteraciones con respecto a la precisión de la mantisa del paquete de datos x_2_62_35 con un error de 0.0000288.
Podemos observar que, tanto en el Método de Newton como en el de Secante, la cantidad de iteraciones se mantiene relativamente constante entre una precisión de 50 y 25 (en 5 para Newton y en 17 para Secante). Luego, dicho valor disminuye en Secante (pasando a valer 21 siendo 18 la precisión) y aumenta en Newton (pasando a ser 13 iteraciones en 18 de mantisa). Por otro lado, podemos ver que el Método de Bisección se mantiene constante en 9 iteraciones hasta 40 de mantisa. Luego, una vez que la precisión pasa a valer 25, dicho valor desciende a 1 para alcanzar las 4000 iteraciones en 18 de precisión. Esto último no es del todo claro en nuestra figura dado que debimos recortarla para que fueran claros los demás valores alcanzados por las curvas.

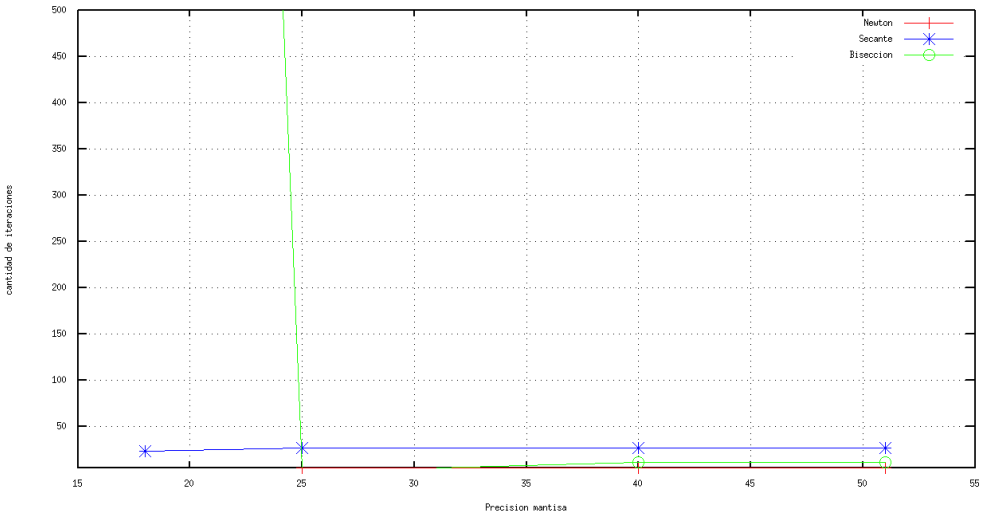

Figura 11: Comparación de la cantidad de iteraciones con respecto a la precisión de la mantisa.
Paquete de datos x_2_62_35.txt - Error 0.000001

- Error máximo aceptable: 0.000001
- Criterio de parada: $|f(X_n)| < \epsilon$
- p_0 en Newton: 10
- p_0 en Secante: 10
- p_1 en Secante: 11
- Intervalo en Bisección: [10,3]
- Máxima cantidad de iteraciones: 4000
- Muestra evaluada: x_2_62_35.txt

Observaciones: En esta figura se compara la cantidad de iteraciones con respecto a la precisión de la mantisa del paquete de datos x_2_62_35 con un error de 0.000001.
Si bien en nuestro gráfico el máximo valor mostrado en la cantidad de iteraciones es 1000, éste debería ser, en realidad, 4000. Dado que la escala debía quedar proporcional para que los demás métodos pudieran ser reconocidos gráficamente, decidimos recortarlo. Podemos observar que el Método de Bisección se mantiene en una cantidad de 11 iteraciones en 51 de precisión para luego bajar a 1 en 25 y finalizar por remontar a las 4000 iteraciones en 18. Por otro lado, al pasar de 25 a 51 de mantisa, la cantidad de iteraciones se mantiene constante en 5 en el Método de Newton. Al pasar de 25 a 18 de precisión, la cantidad de iteraciones de dicho método pasa a 4000. Del mismo modo, entre 51 y 18 de precisión, podemos ver que el Método Secante se mantiene en una cantidad de 27 iteraciones.

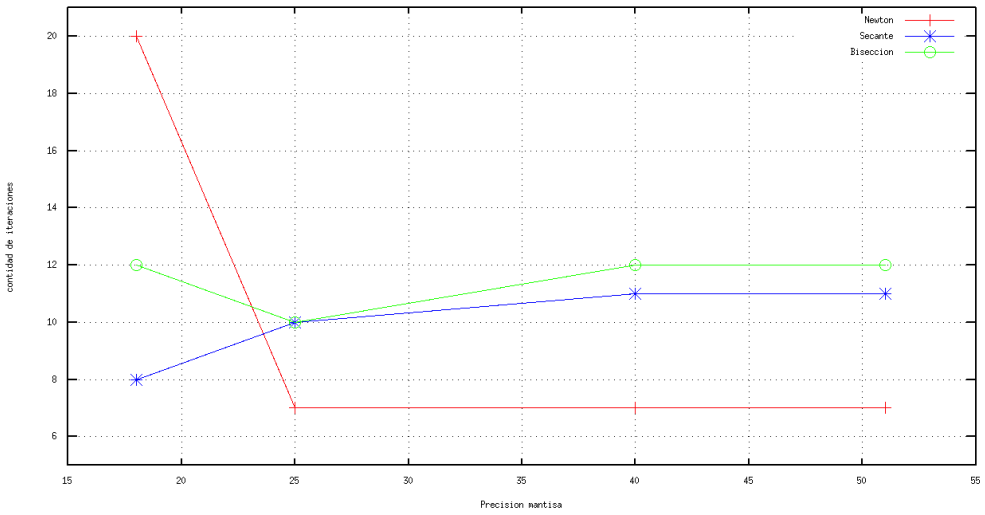

Figura 12: Comparación de la cantidad de iteraciones con respecto a la precisión de la mantisa. Paquete de datos X1.txt

- Error máximo aceptable: 0.0000288
- Criterio de parada: $|f(X_n)| < \epsilon$
- p_0 en Newton: 10
- p_0 en Secante: 10
- p_1 en Secante: 11
- Intervalo en Bisección: [10,3]
- Máxima cantidad de iteraciones: 4000
- Muestra evaluada: X1.txt

Observaciones: En este gráfico se compara la cantidad de iteraciones con respecto a la precisión de la mantisa del paquete de datos X1 con un error de 0.0000288.
Podemos percibir que el Método de Newton mantiene la cantidad de iteraciones firme en 7 entre una precisión de 25 y 51 de mantisa. Luego, al disminuir la precisión a 18, pasa a 20 iteraciones. Por otra parte, el Método de Secante se mantiene en 11 iteraciones entre 51 y 40 para luego disminuirlas a 10 hasta 25 de precisión y volver a bajarlas a 8. Finalmente, el Método de Bisección se mantiene constante en 12 iteraciones hasta alcanzar el valor de 40 de precisión, para disminuirlas a 10 iteraciones en 25 y terminar incrementandolas 12 nuevamente con 18 de mantisa.

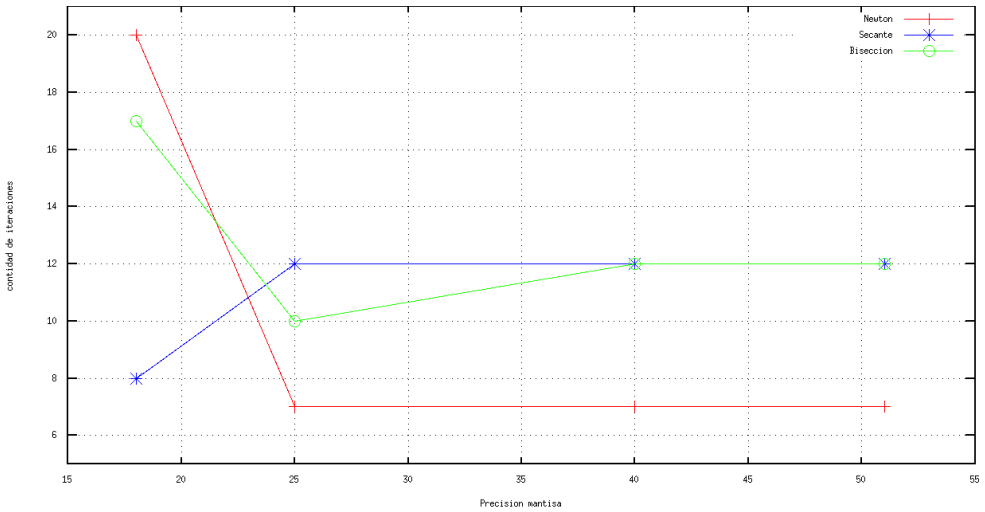

Figura 13: Comparación de la cantidad de iteraciones con respecto a la precisión de la mantisa.
Paquete de datos X1.txt - Error de 0.000001

- Error máximo aceptable: 0.000001
- Criterio de parada: $|f(X_n)| < \epsilon$
- p_0 en Newton: 10
- p_0 en Secante: 10
- p_1 en Secante: 11
- Intervalo en Bisección: [10,3]
- Máxima cantidad de iteraciones: 4000
- Muestra evaluada: X1.txt

Observaciones: En esta figura, se compara la cantidad de iteraciones con respecto a la precisión de la mantisa del paquete de datos X1 con el error 0.000001.
Podemos observar que el Método de Newton mantiene constante la cantidad de iteraciones (en 7) entre una precisión de 51 y 25 . Luego, al disminuir la precisión, dicho valor disminuye a 20. En el caso de Secante, la cantidad de iteraciones se mantiene en 12 hasta una precisión de 25 para luego disminuir este valor llegando a las 8 iteraciones en 18 de mantisa. Por otro lado, el Método de Bisección se mantiene constante en 12 iteraciones hasta llegar a 40 de precisión para luego descender a 10 en 25 de precisión y elevarse nuevamente a 17 una vez que la mantisa alcanza un valor de 18.

Análisis sobre las observaciones:
A partir de las observaciones realizadas previamente, podemos señalar que al alcanzar un valor de precisión menor a 25, la cantidad de iteraciones se eleva (como en Newton y Bisección). En cuanto la precisión a alcanzar es demasiado alta, dicha subida se vuelve excesiva al punto de hacer divergir gran parte de las experimentaciones. Por este motivo, nos resultó imprescindible el uso de una cota superior para la cantidad de iteraciones a realizar por el programa dado que, en el caso contrario, el programa podría haber alcanzado un bucle infinito.

En el caso de Secante, se puede visualizar de forma clara que en la mayoría de los casos y siempre que la precisión fuera pequeña, la cantidad de iteraciones suele disminuir y no divergir, por lo que alcanza el error buscado.

Por otro lado, se puede notar que cuando la precisión varía entre 51 y 25, la cantidad de iteraciones se mantiene constante fluctuando muy levemente en algunos casos (como por ejemplo Newton en la Figura 10). Analizando todos los tests realizados, pudimos observar que el Método de Newton es el que menos iteraciones realiza, siguiéndolo el Método de Bisección para terminar por el Método de Secante.

Análisis de errores:

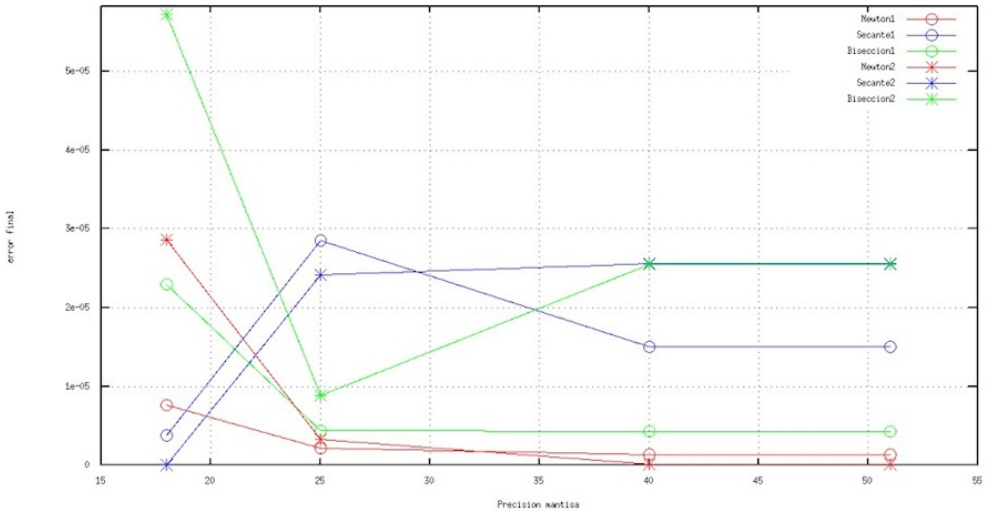

Figura 14: Comparación del error final con respecto a la precisión de la mantisa.
Criterio de parada: $|f(X_n)| < \epsilon$

- Criterio de parada: $|f(X_n)| < \epsilon$
- Máxima cantidad de iteraciones: 4000
- Error máximo aceptable: 0.0000288
- Secante1 - Newton1 - Bisección1: Corresponden a la muestra X1.txt
- Secante2 - Newton2 - Bisección2: Corresponden a la muestra x_2_62_35.txt

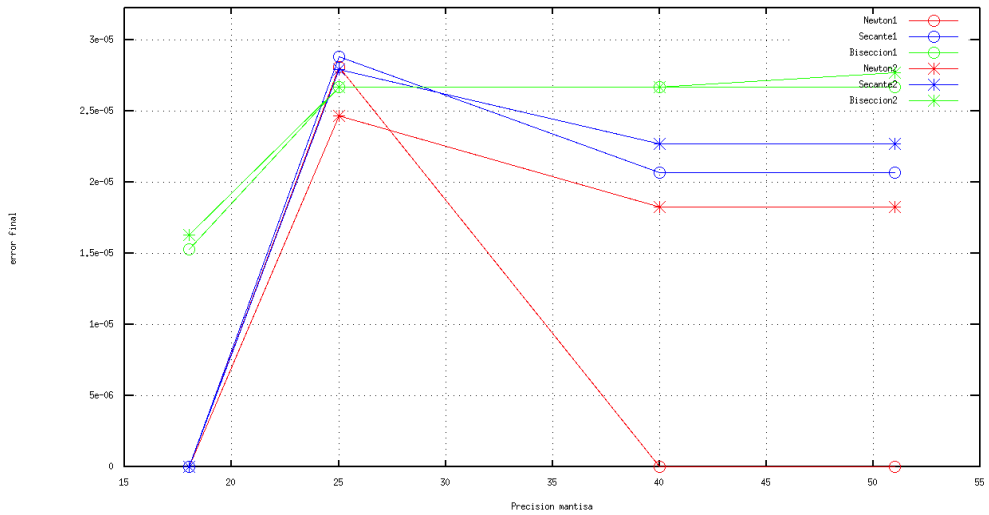

Figura 15: Comparación del error final con respecto a la precisión de la mantisa.
Criterio de parada: $|f(X_n) - f(X_{n-1})| < \epsilon$

- Criterio de parada: $|f(X_n) - f(X_{n-1})| < \epsilon$
- Máxima cantidad de iteraciones: 4000
- Error máximo aceptable: 0.0000288
- Secante1 - Newton1 - Bisección1: Corresponden a la muestra X1.txt
- Secante2 - Newton2 - Bisección2: Corresponden a la muestra x_2_62_35.txt

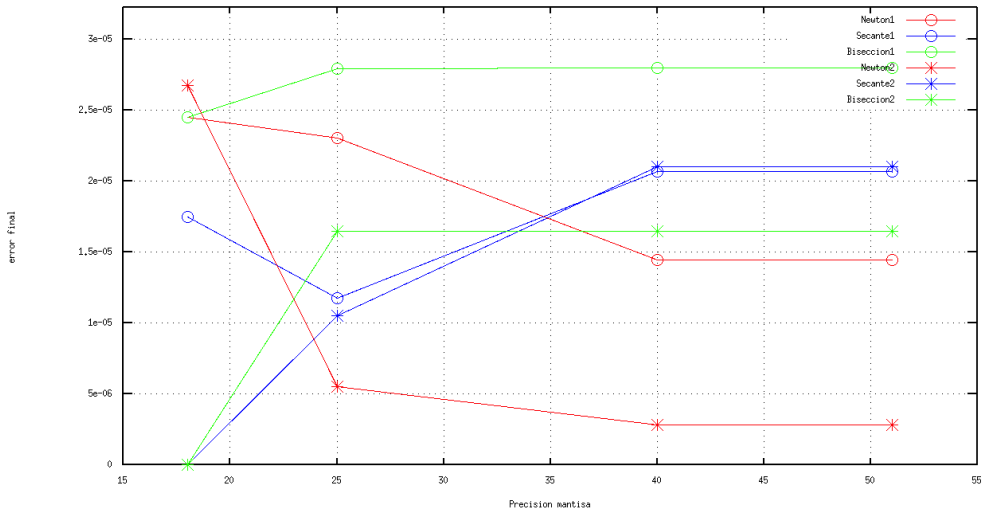

Figura 16: Comparación del error final con respecto a la precisión de la mantisa.
Criterio de parada: $\frac{|X_n - X_{n-1}|}{|X_{n-1}|} < \epsilon$

- Criterio de parada: $\frac{|X_n - X_{n-1}|}{|X_{n-1}|} < \epsilon$
- Máxima cantidad de iteraciones: 4000
- Error máximo aceptable: 0.0000288
- Secante1 - Newton1 - Bisección1: Corresponden a la muestra X1.txt
- Secante2 - Newton2 - Bisección2: Corresponden a la muestra x_2_62_35.txt

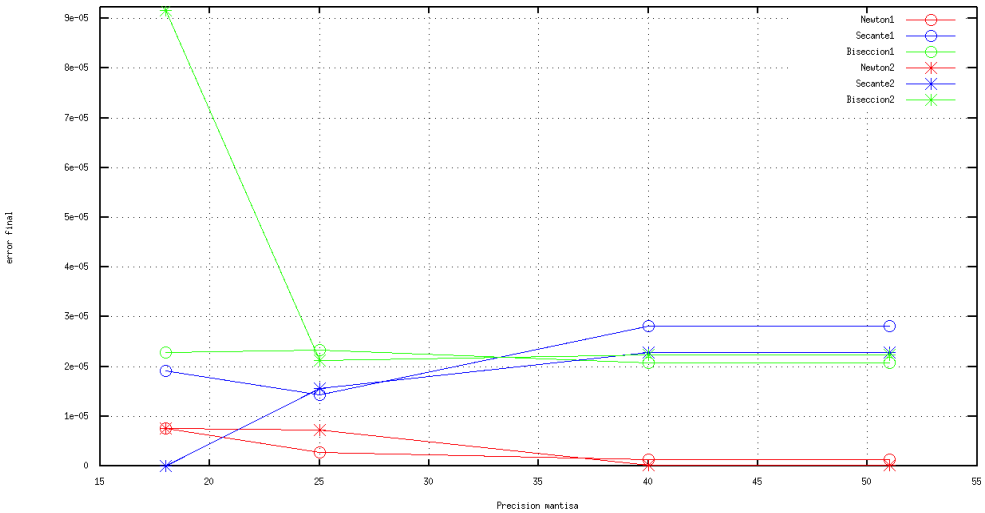

Figura 17: Comparación del error final con respecto a la precisión de la mantisa.
Criterio de parada: $|f(X_n) - f(X_{n-1})| < \epsilon$

- Criterio de parada: $|f(X_n) - f(X_{n-1})| < \epsilon$
- Máxima cantidad de iteraciones: 4000
- Error máximo aceptable: 0.0000288
- Secante1 - Newton1 - Bisección1: Corresponden a la muestra X1.txt
- Secante2 - Newton2 - Bisección2: Corresponden a la muestra x_2_62_35.txt

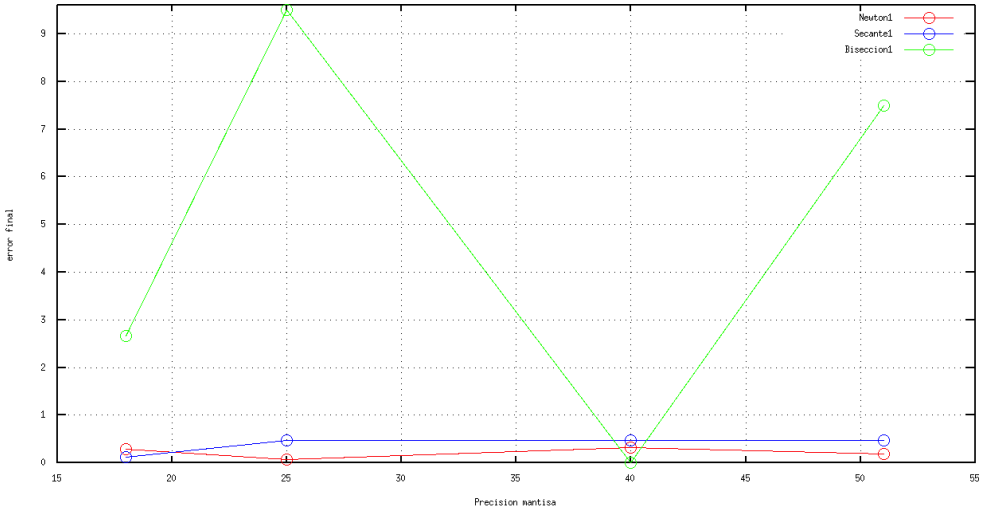

Figura 18: Comparación del error final con respecto a la precisión de la mantisa.
Criterio de parada: $\frac{|f(X_n)-f(X_{n-1})|}{|f(X_{n-1})|} < \epsilon$

- Criterio de parada: $\frac{|f(X_n)-f(X_{n-1})|}{|f(X_{n-1})|} < \epsilon$
- Máxima cantidad de iteraciones: 4000
- Error máximo aceptable: 0.0000288
- Secante1 - Newton1 - Bisección1: Corresponden a la muestra X1.txt
- Secante2 - Newton2 - Bisección2: Corresponden a la muestra x_2_62_35.txt

Observaciones de los gráficos de criterios de parada:

En el caso de los 5 criterios de parada, decidimos hacer un análisis general entre ellos ya que no exponían muchas diferencias en cuanto a su esencia. Para poder analizarlos correctamente, deberíamos realizar una división en cuanto a la cantidad de precisión a utilizar. Si miramos los valores de t más altos (entre 25 y 50), lo primero que puede ser visto en los criterios es que el Método de Newton es el que menos error suele tener. Esta característica lo beneficia mucho y es la causa por la que suele ser uno de los métodos de mayor convergencia. A continuación de éste, se encuentra el Método Secante y, por último, el de Bissección.
Si analizamos los t's más chicos, el panorama parece ser completamente distinto ya que la regularidad del error que se obtiene con valores altos de t no se obtiene con los bajos. Es por esto que en muchos casos, al ser t muy chico, el error aumenta mucho y esto hace que no se pueda llegar con facilidad a la cota establecida por el error mínimo. Por lo tanto, éste termina resultando del método por la cota superior de iteraciones.

Análisis del tiempo de ejecución de cada método:

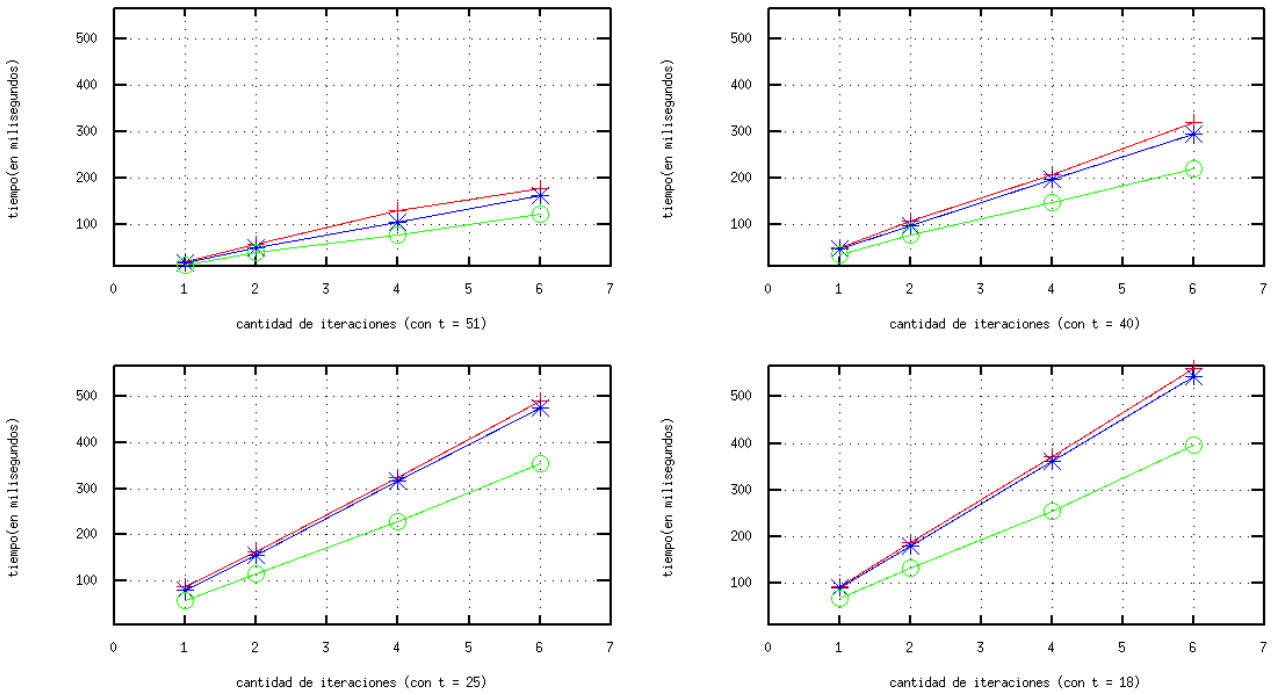

Figura 19: Comparación del tiempo de ejecución del programa respecto de la cantidad de iteraciones. Pruebas realizadas con t=18, t=25, t=40 y t=51

El gráfico fue confeccionado con el paquete $x\_2\_62\_35.txt$ utilizando para Newton p_0=10, para Secante p_0=11 y p_1=10 y para Bisección los extremos 3 y 10. Además, se realizó con 0.000000001 como error máximo.

En este gráfico, se puede observar claramente cómo a menor precisión, mayor es la demora al ejecutar las iteraciones. Además, se puede ver cómo cada iteración realizada por el Método de Newton tiene mayor retardo que las realizadas por Método de Secante, que, a su vez, es mayor que las ejecutadas por el Método de Bisección.

A) Casos particulares:
En la muestra $x\_2\_62\_35$, vimos que el Método de Bisección con $|f(X_n)| < \epsilon$ y 25 iteraciones, resolvió el problema en una única iteración devolviendo $\tilde{\beta}$=6.5 (siendo β=6.2) con un error final de 8.82149e-006. Dicho resultado resultó sorprendente dado a que el valor más próximo a β hallado gracias a estimaciones resultó ser 6.44. Por lo tanto quedamos perplejos al ver que con una única iteración se pudiera alcanzar un valor tan próximo al buscado.

B) Variaciones:
No siempre un criterio de parada fue el mejor para un método determinado. Las variaciones de la precisión y el ϵ permitieron hallar las combinaciones más eficientes, diferenciando claramente entre una elección y otra.

C) Resultados parciales:
Se destacan de la muestra la franja de t=[40,52] con pocas variaciones para el ejemplo, la franja t=(40,25] con leves pero mayores variaciones, y la franja t=(25,15) donde los resultados se alejaban sensiblemente de los reales.

Además de los resultados ($\tilde{\beta}$, $\tilde{\lambda}$ y $\tilde{\sigma}$), las diferentes precisiones modifican sensiblemente los tiempos y las iteraciones. A menor precisión hay más iteraciones y éstas consumen más tiempo.

Los métodos elegidos para las funciones dadas se comportaron mejor con los criterios de parada 1, 2 y 4. El criterio número 5 sólo nos brindó información con un error menor a 0.5.

Observando los tiempos y las iteraciones notamos que las iteraciones de Bisección son mucho más rápidas que las de Secante y Newton. Haciendo que poca diferencia entre la cantidad de iteraciones de un método y otro dé como resultado que Bisección termine antes su ejecución.

En cuanto a los errores resultantes al finalizar los métodos, Newton siempre termina más cerca del cero que los otros métodos.

Los resultados obtenidos en cuanto a cantidad de iteraciones realizadas por los métodos es coherente con la teoría: Newton converge más rápido que Secante, y éste que Biseccion.
También encontramos coherentes los resultados de tiempo (por su relación con la cantidad de iteraciones) y de error, confirmando la teoría.

Encontramos que para ésta función, la forma de parada 5 no nos brindaba información (los métodos realizaban en la mayoría de los casos la máximo cantidad de iteraciones).
Ésta comprobación se hizo con varios parámetros de entrada para cada función (ϵ = 1E-10, 1E-6, 0.0001, 0.001, 0.5)
Los test con $\epsilon \leq 0.5$ no aportaron datos pero los tests con $\epsilon \geq 0.001$ sí. Los datos aportados no son significativos en comparación con el resto de los datos obtenidos.
Gracias a los últimos tests y al análisis de tError=5, creemos que el inconveniente viene a raíz de que éste critero de parada da un número muy chico, y por eso suele no aportar mucha información.

5. Conclusiones

Hemos concluido que el mejor tiempo de ejecución resultó ser el del Método de Newton que, a su vez, fue el que menos iteraciones realizó para hallar los ceros de función como también el que menos error obtuvo.

- Con t<40 los valores se comienzan a diferenciar, alejándose del β real. Con t<=25 se diferencian más rápido.

- Con t<=18 se observan algunos errores finales igualados a 0, de lo que concluimos que el número a representar está fuera de la escala de números que se pueden representar con ésta precisión.

- Con t<15 se observan varios (o todos) los valores como Nan.

- El tError=5 no nos dio datos útiles de error ni iteraciones paraϵ >0.5

Hemos encontrado de forma empírica que para una función a la cual le conocemos la derivada, el Método de Newton es más eficaz que los métodos Secante y Bisección. Ésta eficiencia está determinada por la precisión, la cantidad de iteraciones y el tiempo que le toma al algoritmo resolver el problema dado.
También hemos notado las diferencias entre una precisión y otra y los diferentes criterios de parada. Como las variaciones podían resultar desde pequeñas variaciones hasta en métodos que no

cumplían su cometido.

Las diferencias de los distintos ϵ sólo dieron resultados interesantes al analizar tError=5, ya que en los otros casos dieron el esperado resultado de, a menor ϵ, menor error y menos iteraciones.

En el caso del criterio de parada Nro 5, entendimos que el criterio es tan riguroso que sólo con un ϵ chico y mucha precisión resulta útil.

Encontramos un tope a las iteraciones entre 2000 y 4000, para distinguir los métodos que terminaban de los que no. Decidimos mostrar todos los experimentos con 4000 iteraciones máximas para que sean más claros.

Hemos notado que, si bien hay una diferencia sensible entre la cantidad de iteraciones de Newton y las de los otros métodos, entre Bisección y Secante no siempre es tan notable. Incluso, hubo casos donde Bisección fue mucho más rápido que Secante por el hecho de que cada iteración de Bisección es más rápida que las de Secante.

6. Apéndices

6.1. A

El objetivo del trabajo práctico es implementar un programa que permita estimar los parámetros $\Theta = (\sigma, \beta, \lambda)$ a partir de un conjunto de n datos. Para ello, se deberá resolver la ecuaciones (5) o (6). Evaluando los distintos métodos vistos en clase que permitan resolver este problema, se deberá realizar una implementación cumpliendo lo siguiente:

1. Implementar al menos dos métodos (de los cuales uno de ellos debe ser el método de Newton) con aritmética binaria de punto flotante con t dígitos de precisión en la mantisa. El valor t debe ser un parámetro de la implementación, con $t < 52$.

2. Realizar experimentos numéricos con cada método implementado en el ítem anterior elegiendo varias instancias de prueba y en función de las cantidad de dígitos t de precisión en la mantisa (experimentar con al menos 3 valores distintos de t).

3. Para cada método implementado se deberán mostrar resultados obtenidos en cuanto a cantidad necesaria de iteraciones, tiempo de ejecución, precisión en el resultado, y cualquier otro parámetro que considere de interés evaluar.

4. Realizar el gráfico del histograma de los datos y el ajuste obtenido. Extraer conclusiones sobre la efectividad de cada método observando los resultados anteriores.

6.2. B

Implementación de nuestro programa:

main.cpp

```
#include <iostream>
#include <sstream>
#include <cstdio>
#include <cmath>
#include <fstream>
#include <string>
#include <cstring>
#include <list>
#include "metodos.h"
#include "TFloat.h"
```

```
using namespace std;

size_t precision(const size_t maxT)
{
        size_t res = -1;
    while(res > maxT || res < 1)
    {
        cout << "\nIngrese los dígitos de precisión en la mantisa
        (número entero) entre 1 y ";
        cout << maxT  << endl;
        cin >> res;
    }
    return res;
}

int tipoError(int maxElem)
{
        int res = -1;
    while (res <1 || res >maxElem)
    {
        /*tipoError
         * 1 => |f(c)|< error
         * 2 => |an - an-1|< error
         * 3 => |an - an-1|/|an-1| < error
         * 4 => |f(an) - f(an-1)| < error
         * 5 => |f(an) - f(an-1)|/|f(an-1)|< error
         */
        cout << "\n Ingrese el tipo de error (entre 1 y ";
        cout << maxElem;
        cout << ")\n    1 implica |f(c)|< error\n
        2 implica |an - an-1|< error
        \n      3 implica |an - an-1|/|an| < error\n
        4 implica |f(an) - f(an-1)| < error\n
        5 implica |f(an) - f(an-1)|/|f(an-1)|< error" << endl;
        cin >> res;
    }

    return res;
} //tipoError es el tipo de tolencia que se acepta, sacado de una lista. Entonces es entero.

double error()
{
        double res = -1;
    while(res >=1 || res <= 0)
    {
        printf("\nIngrese el error máximo aceptado entre 0 y 1 (float).\n");
        cin >> res;
    }

    return res;
} // Es el Epsilon (error) que ponemos como cota. Es un punto flotante

int maxIteraciones()
{
        int res = 0;
    while (res <1)
    {
```

```
        printf("\nIngrese la cantidad máxima de iteraciones para
               el método de estimación que elija\n");
        cin >> res;
    }

    return res;
} // Es la máxima cantidad de iteraciones que harán los métodos. Es entero


void pedirExtremosParaBiseccion(TFloat *a, TFloat *b, bool *fin,
                                int n, list<double>::iterator it)
{
        double aD,bD;
        do
        {
                cout << "\nIngrese el extremo negativo a" << endl;
                cin >> aD;
                *a = aD;
        }while(fDeS(*a, n, it).dbl()>0);

        if(fDeS(*a, n, it) == 0)
        {
                cout << "\n¡Encontraste el cero!\n El cero es ";
                cout <<aD;
                cout << "\n Pulse una tecla para salir." << endl;
                cin >> aD;
                *fin = true;
        }
        if(!(*fin))
        {
                do
                {
                        cout << "\nIngrese el extremo positivo b" << endl;
                        cin >> bD;
                        *b = bD;
                }while(fDeS(*b, n, it).dbl()<0);

                if(fDeS(*b, n, it) == 0)
                {
                        cout << "\n¡Encontraste el cero!\n El cero es ";
                        cout <<bD;
                        cout << "\n Pulse una tecla para salir." << endl;
                        cin >> bD;
                        *fin = true;
                }
        }
} // Pide a y b. Verifica que f(a) sea negativo y f(b) positivo
               y mayor que a. Luego devuelve a y b por referencia


TFloat sacoP0(int cantMuestra, list<double>::iterator muestra, bool *fin,
              TFloat *p1, int metodo, size_t precision, double error,
              int tipoError, int maxIteraciones)
{
        //Acá consigo p0 con bisección o por pantalla
        TFloat p0(0.0,precision);
        double p0Double, p1Double;

        int iteracionesQRealiza = 0;
```

```
        double errorFinal = 0;
        float tiempo = 0;

        int p0PorUsuario;

        printf("Ingrese '1' para que el programa busque p0. Ingrese '2'
           para ingresarlo por consola \n");
        cin >> p0PorUsuario;

        if(p0PorUsuario == 1)
        {
                TFloat a = TFloat(precision);
                TFloat b = TFloat(precision);
                pedirExtremosParaBiseccion(&a, &b, fin, cantMuestra, muestra);
                if (metodo == 1)
                        p0 = biseccion(cantMuestra, muestra, precision, error,
                           a, b, tipoError, maxIteraciones,
                           &iteracionesQRealiza,
                           &errorFinal, &tiempo, p1, true);

                else
                        p0 = biseccion(cantMuestra, muestra, precision, error,
                           a, b, tipoError, maxIteraciones,
                           &iteracionesQRealiza,
                           &errorFinal, &tiempo, p1, false);

        }
        else
        {
                printf("\nIngrese la estimación de Beta, p0 \n");
                cin >> p0Double;
                if (metodo == 1)
                {
                        printf("\nIngrese la segunda estimación de Beta, p1 \n");
                        cin >> p1Double;
                }
                TFloat p0b(p0Double, precision);
                TFloat p1b(p1Double, precision);
                *p1 = p1b;
                p0 = p0b;
        }

        return p0;
} //Saco p0 (y p1 si usa secante). Para eso discrimino entre pedirlo por consola y obtenerlos
haciendo biseccion.

int main(void)
{

    list<double> muestra;

    printf("-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
           *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-");
    printf("\nBienvenido al programa del TP1 de MetNum,
           primer cuatrimestre 2013.\n");

    size_t _precision = precision(52);

    int _tipoError = tipoError(5);
```

```
    double _error = error();

    int _maxIteraciones = maxIteraciones();

        TFloat Beta(0.0,_precision);

    string archivo;
    char respuesta = 'n';
    int cantMuestra = -1;

    while (respuesta == 'n')
    {
        cout << "\n Ingrese el archivo donde está la muestra
        (sin olvidar su extensión)\nÉste
        debe encontrarse en la misma carpeta que el ejecutable.\n" << endl;
        cin >> archivo;
        //tengo q pasar de string a char* para poder abrir un archivo con el nombre que me
pasan por pantalla.
            char * cArchivo = new char [archivo.length()+1];
            strcpy(cArchivo, archivo.c_str());

        ifstream entrada(cArchivo);
        if(entrada.good())
        {
            entrada >> cantMuestra;
            while(!entrada.eof())
            {
                float nuevoElemento;
                entrada >> nuevoElemento;
                if(nuevoElemento != EOF)
                    muestra.push_back(nuevoElemento);
            }
            respuesta = 's';
        }
        else
        {
            printf("\nNo se cargó el archivo. Salir? (s/n)\n");
            cin >> respuesta;
            if(respuesta == 's')
            {
                return 0;
            }
        }
        entrada.close();
    }

    int metodo = 8;
    while (metodo < 0 || metodo > 2)
    {
        printf("\nIngrese el método numérico que desee utilizar para aproximar
               Beta\n  0 = Newton\n    1 = Secante \n    2 = Bisección\n");
        cin >> metodo;
    }
    bool fin = false;
    TFloat p0(0.0,_precision);
    TFloat p1(0.0,_precision);
    if (metodo == 0 || metodo == 1)
    {
```

```
    p0 = sacoP0(cantMuestra, muestra.begin(), &fin, &p1, metodo, _precision,
                _error, _tipoError, _maxIteraciones);

    if(fin)
            return 0;
}
TFloat a(0.0,_precision);
TFloat b(0.0,_precision);
//ésto me lo devuelven las funciones.
/**/int iteracionesQrealiza;
/**/float tiempo;
/**/double errorFinal;

if(metodo == 1)
    Beta = secante(cantMuestra, muestra.begin(), _precision, _error,
                   p0, p1, _tipoError, _maxIteraciones,
                   &iteracionesQrealiza, &errorFinal, &tiempo);
else
    if(metodo == 2)
            {
                    pedirExtremosParaBiseccion(&a, &b, &fin, cantMuestra,
                                       muestra.begin());
            if(fin)
                    return 0;
                    Beta = biseccion(cantMuestra, muestra.begin(), _precision,
                             _error, a, b, _tipoError, _maxIteraciones,
                             &iteracionesQrealiza, &errorFinal, &tiempo,
                             &p1, false);
            }
    else
            Beta = newton(cantMuestra, muestra.begin(), _precision, _error,
                      p0, _tipoError, _maxIteraciones, &iteracionesQrealiza,
                      &errorFinal, &tiempo);


TFloat l(0.0,_precision);
TFloat s(0.0,_precision);
l = lambda(Beta, cantMuestra, muestra.begin());
s = sigma(Beta, l, cantMuestra, muestra.begin());

cout << "\n\n*********************************" << endl;
cout << "**Resumen:\n*-Precisión (t, cantidad de dígitos de mantisa): ";
cout << _precision << endl;
cout << "*-Error: ";
cout << _error << endl;
cout << "*-Tipo de error: ";
switch(_tipoError)
{
    case 1:
            cout << "|f(c)|< error" << endl;
            break;
    case 2:
        cout << "|an - an-1|< error" << endl;
        break;
    case 3:
        cout << "(|an - an-1| / |an-1|)< error" << endl;
        break;
    case 4:
        cout << "|f(an( - (an-1)|< error" << endl;
```

```
            break;
        case 5:
            cout << "(|f(an) - f(an-1)| / |f(an-1)|)< error" << endl;
            break;
    }
    cout << "*-Máxima cantidad de iteraciones: ";
    cout << _maxIteraciones << endl;
    cout << "*-Método: ";
    switch(metodo)
    {
        case 0:
                cout << "Newton" << endl;
                break;
        case 1:
            cout << "Secante" << endl;
            break;
        case 2:
            cout << "Biseccion" << endl;
            break;
    }
    if(metodo == 2)
    {
        cout << "       a: ";
        cout << a.dbl() << endl;
        cout << "       b: ";
        cout << b.dbl() << endl;
    }
    else
    {
        cout << "       p0: ";
        cout << p0.dbl() << endl;
        if(metodo == 1)
        {
                cout << "       p1: ";
                cout << p1.dbl() << endl;
        }
    }


        cout << "\nSigma = ";
        cout << s.dbl();
        cout << " ; Beta = ";
        cout << Beta.dbl();
        cout << " ; Lambda = ";
        cout << l.dbl();
        cout << "\nTardó ";
        cout << tiempo;
        cout << " milisegundos, realizó ";
        cout << iteracionesQrealiza;
        cout << " iteraciones y terminó con un error de ";
        cout << errorFinal << endl;
    return 0;
}
```

metodos.cpp

```
/*#include <cmath>
#include <iostream>
```

```
#include <list>
#include <time.h>
#include "metodos.h"
#include "formulas.h"
#include "TFloat.h"

using namespace std;

TFloat newton(int n, list<double>::iterator m, size_t precision, double error,
    TFloat p0, int tipoError, int maxIteraciones, int *iteracionesQRealiza,
    double *errorFinal, float *tiempo){
    TFloat xn = p0;
  TFloat xnMas1(0.0, precision);
  bool _error = 0;
  int i=1;

    clock_t t_ini, t_fin;

    t_ini = clock();
  while(i <= maxIteraciones && _error == 0){

    xnMas1 = xn - (fDeS(xn, n, m)/(fPrimaDeS(xn, n, m)));

    switch (tipoError) { //calculo _error
      case 1:
        if(fabs(fDeS(xnMas1, n, m).dbl()) < error){
          *errorFinal = fabs(fDeS(xnMas1, n, m).dbl());
          _error=1;
        }
        break;
      case 2:
        if(fabs((xnMas1-xn).dbl()) < error){
          *errorFinal = fabs((xnMas1-xn).dbl());
          _error=1;
        }
        break;
      case 3:
        if(fabs(((xnMas1-xn)/ xn).dbl()) < error){
          *errorFinal = fabs(((xnMas1-xn)/ xn).dbl());
          _error=1;
        }
        break;
      case 4:
        if(fabs((fDeS(xnMas1, n, m)-fDeS(xn, n, m)).dbl())< error){
          *errorFinal = fabs((fDeS(xnMas1, n, m)-fDeS(xn, n, m)).dbl());
          _error=1;
        }
        break;
      case 5:
        if(fabs(((fDeS(xnMas1, n, m)-fDeS(xn, n, m))/fDeS(xn, n, m)).dbl())<
            error){
          *errorFinal = fabs(((fDeS(xnMas1, n, m)-
                        fDeS(xn, n, m))/fDeS(xn, n, m)).dbl());
          _error=1;
        }
        break;

    }
    xn = xnMas1;
```

```
    i++;
  }

  *iteracionesQRealiza = --i;

    t_fin = clock();

    *tiempo = (((float)t_fin-(float)t_ini) * 1000)/CLOCKS_PER_SEC;


  switch (tipoError) { //calculo _error
    case 1:
      if(_error==0){
        *errorFinal = fabs(fDeS(xnMas1, n, m).dbl());
      }
      break;
    case 2:
      if(_error==0){
        *errorFinal = fabs((xnMas1-xn).dbl());
      }
      break;
    case 3:
      if(_error==0){
        *errorFinal = fabs(((xnMas1-xn)/ xn).dbl());
      }
      break;
    case 4:
      if(_error==0){
        *errorFinal = fabs((fDeS(xnMas1, n, m)-fDeS(xn, n, m)).dbl());
      }
      break;
    case 5:
      if(_error==0){
        *errorFinal = fabs(((fDeS(xnMas1, n, m)-
                        fDeS(xn, n, m))/fDeS(xn, n, m)).dbl());
      }
      break;
    }

  return xn;
}

TFloat secante(int n, list<double>::iterator m, size_t precision, double error,
               TFloat p0, TFloat p1, int tipoError, int maxIteraciones,
               int *iteracionesQRealiza, double *errorFinal, float *tiempo){
    TFloat xnMenos1 = p0;
    TFloat xn = p1;
  TFloat xnMas1(0, precision);
  bool _error = 0;
  int i=1;

  clock_t t_ini, t_fin;
    t_ini = clock();

  while(i <= maxIteraciones && _error == 0){
    xnMas1 = xn - ((xn - xnMenos1 )/(fDeS(xn, n, m)-
            fDeS(xnMenos1, n, m))) * fDeS(xn, n, m);

    switch (tipoError) { //calculo _error
```

```
    case 1:
      if(fabs(fDeS(xnMas1, n, m).dbl()) < error){
        *errorFinal = fabs(fDeS(xnMas1, n, m).dbl());
        _error=1;
      }
      break;
    case 2:
      if(fabs((xnMas1-xn).dbl()) < error){
        *errorFinal = fabs((xnMas1-xn).dbl());
        _error=1;
      }
      break;
    case 3:
      if(fabs(((xnMas1-xn)/ xn).dbl()) < error){
        *errorFinal = fabs(((xnMas1-xn)/ xn).dbl());
        _error=1;
      }
      break;
    case 4:
      if(fabs((fDeS(xnMas1, n, m)-fDeS(xn, n, m)).dbl())< error){
        *errorFinal = fabs((fDeS(xnMas1, n, m)-fDeS(xn, n, m)).dbl());
        _error=1;
      }
      break;
    case 5:
      if(fabs(((fDeS(xnMas1, n, m)-fDeS(xn, n, m))/fDeS(xn, n, m)).dbl())<
          error){
        *errorFinal = fabs(((fDeS(xnMas1, n, m)-
                    fDeS(xn, n, m))/fDeS(xn, n, m)).dbl());
        _error=1;
      }
      break;

  }
  xn = xnMas1;
  i++;
}

*iteracionesQRealiza = --i;
  t_fin = clock();
  *tiempo = (((float)t_fin-(float)t_ini) * 1000)/CLOCKS_PER_SEC;

  switch (tipoError) { //calculo _error
  case 1:
    if(_error==0){
      *errorFinal = fabs(fDeS(xnMas1, n, m).dbl());
    }
    break;
  case 2:
    if(_error==0){
      *errorFinal = fabs((xnMas1-xn).dbl());
    }
    break;
  case 3:
    if(_error==0){
      *errorFinal = fabs(((xnMas1-xn)/ xn).dbl());
    }
    break;
  case 4:
```

```
      if(_error==0){
        *errorFinal = fabs((fDeS(xnMas1, n, m)-fDeS(xn, n, m)).dbl());
      }
      break;
    case 5:
      if(_error==0){
        *errorFinal = fabs(((fDeS(xnMas1, n, m)-
        fDeS(xn, n, m))/fDeS(xn, n, m)).dbl());
      }
      break;
    }

  return xn;
}

TFloat biseccion(int n, list<double>::iterator m, size_t precision,
                    double error, TFloat a, TFloat b, int tipoError,
                    int maxIteraciones, int *iteracionesQRealiza,
                    double *errorFinal, float *tiempo, TFloat *c2,
                    bool sacarC2){
  TFloat c(0.0, precision);
    TFloat anMenos1(0.0, precision);
  bool _error = 0;
  int i=1;

    clock_t t_ini, t_fin;
    t_ini = clock();
  do
  {
    *c2 = c;
    while(i <= maxIteraciones && _error == 0){

      c = (a+b)/2;

      if((fDeS(a, n, m)*fDeS(c, n, m)).dbl() < 0){
                anMenos1 = a;
                b = c;
            }
      else{
                anMenos1 = b;
                a = c;
      }

      switch (tipoError) { //calculo _error
        case 1:
          if(fabs(fDeS(c, n, m).dbl()) < error){
            *errorFinal = fabs(fDeS(c, n, m).dbl());
            _error=1;
          }
          break;
        case 2:
          if(fabs((c-anMenos1).dbl()) < error){
            *errorFinal = fabs((c-anMenos1).dbl());
            _error=1;
          }
          break;
        case 3:
          if(fabs(((c-anMenos1)/ anMenos1).dbl())< error){
            *errorFinal = fabs(((c-anMenos1)/ anMenos1).dbl());
```

```
          _error=1;
        }
        break;
      case 4:
        if(fabs((fDeS(c, n, m)-fDeS(anMenos1, n, m)).dbl())< error){
          *errorFinal = fabs((fDeS(c, n, m)-fDeS(anMenos1, n, m)).dbl());
          _error=1;
        }
        break;
      case 5:
        if(fabs(((fDeS(c, n, m)-
          fDeS(anMenos1, n, m))/fDeS(anMenos1, n, m)).dbl()) < error){
          *errorFinal = fabs(((fDeS(c, n, m)-
                        fDeS(anMenos1, n, m))/fDeS(anMenos1, n, m)).dbl());
          _error=1;
        }
        break;
    }
    i++;
  }

sacarC2 = !sacarC2;
if (sacarC2 == true)
  i--;
else
{
  i = maxIteraciones -2;
  _error = 0;
}
}while(sacarC2 == false);

*iteracionesQRealiza = i;

  t_fin = clock();
  *tiempo = (((float)t_fin-(float)t_ini) * 1000)/CLOCKS_PER_SEC;

  switch (tipoError) { //calculo _error
    case 1:
      if(_error==0){
        *errorFinal = fabs(fDeS(c, n, m).dbl());
      }
      break;
    case 2:
      if(_error==0){
        *errorFinal = fabs((c-anMenos1).dbl());
      }
      break;
    case 3:
      if(_error==0){
        *errorFinal = fabs(((c-anMenos1)/ anMenos1).dbl());
      }
      break;
    case 4:
      if(_error==0){
        *errorFinal = fabs((fDeS(c, n, m)-fDeS(anMenos1, n, m)).dbl());
      }
      break;
    case 5:
      if(_error==0){
```

```
          *errorFinal = fabs(((fDeS(c, n, m)-
                        fDeS(anMenos1, n, m))/fDeS(anMenos1, n, m)).dbl());
        }
      break;
      }

  return c;
}
```

metodos.h

```
#ifndef METODOS_
#define METODOS_

#include <cmath>
#include <iostream>
#include <list>
#include "formulas.h"
#include "TFloat.h"
using namespace std;


/*tipoError
 * 1 => |f(c)|< error
 * 2 => |an - an-1|< error
 * 3 => |an - an-1|/|an-1| < error
 * 4 => |f(an) - f(an-1)| < error
 * 5 => |f(an) - f(an-1)|/|f(an-1)|< error
 */

TFloat newton(int n, list<double>::iterator m, size_t precision,
              double error, TFloat p0, int tipoError,
              int maxIteraciones, int *iteracionesQRealiza,
              double *errorFinal, float *tiempo);
TFloat secante(int n, list<double>::iterator m, size_t precision,
               double error, TFloat p0, TFloat p1, int tipoError,
               int maxIteraciones, int *iteracionesQRealiza,
               double *errorFinal, float *tiempo);
TFloat biseccion(int n, list<double>::iterator m, size_t precision,
                 double error, TFloat a, TFloat b,
                 int tipoError, int maxIteraciones,
                 int *iteracionesQRealiza, double *errorFinal,
                 float *tiempo, TFloat *c2, bool sacarC2);

#endif
```

formulas.cpp

```
#include <cmath>
#include <iostream>
#include <list>
#include "formulas.h"
#include "TFloat.h"

using namespace std;

/*Cálculo de las funciones básicas (MdeS,MsombreroDeS,RdeS)*/
```

```
TFloat MdeS(TFloat s, int n, list<double>::iterator it){
        size_t _precision = s.precision();
        TFloat sum(0.0, _precision);

        for(int i = 0; i<n; i++)
        {
                sum=sum+((s*log(*it)).exponencial());
                it++;
        }
        return sum/n;
}

TFloat MsombreroDeS(TFloat s, int n, list<double>::iterator it){
        size_t _precision = s.precision();
        TFloat sum(0.0, _precision);

        for(int i = 0; i<n; i++)
        {
                sum=sum+(((s*log(*it)).exponencial())*log(*it));
                it++;
        }
        return sum/n;
}

TFloat RdeS(TFloat s, int n, list<double>::iterator it){
    return MsombreroDeS(s,n,it)/MdeS(s,n,it);
}

/*Cálculo de las derivadas de las funciones básicas(MsombreroPrimaDeS,RprimaDeS)
 *
 * Aclaración: MPrimaDeS = MsombreroDeS*/

TFloat MPrimaDeS(TFloat s, int n, list<double>::iterator it){
        return MsombreroDeS(s, n, it);
}

TFloat MsombreroPrimaDeS(TFloat s, int n, list<double>::iterator it){
        size_t _precision = s.precision();
        TFloat sum(0.0, _precision);

        for(int i = 0; i<n; i++)
        {
                sum=sum+((s*log(*it)).exponencial()*pow(log(*it),2));
                it++;
        }
        return sum/n;
}

TFloat RprimaDeS(TFloat s, int n, list<double>::iterator it){
    TFloat numerador = MsombreroPrimaDeS(s, n, it)*MdeS(s, n, it) -
    MPrimaDeS(s, n, it)* MsombreroDeS(s, n, it);
    TFloat denominador = MdeS(s, n, it)*MdeS(s, n, it);
    return numerador/denominador;
}

/*Cálculo de f(s) y f'(s) (f(s) es la funcion (5) igualada a 0)*/

TFloat fDeS(TFloat s, int n, list<double>::iterator it){
```

```
    size_t _precision = s.precision();
    TFloat cero(0.0, _precision);
    return s*(RdeS(s, n, it) - RdeS(cero,n, it)) -
    (MdeS(s*2, n, it)/(MdeS(s, n, it)*MdeS(s, n, it))) + 1;
}

TFloat fPrimaDeS(TFloat s, int n, list<double>::iterator it){
    size_t _precision = s.precision();
    TFloat cero(0.0, _precision);
        TFloat _MdeS = MdeS(s, n, it);
    TFloat numerador = MPrimaDeS(s*2, n, it)*_MdeS*_MdeS*2 -
    MdeS(s*2, n, it)*MPrimaDeS(s, n, it)*_MdeS*2;
    TFloat denominador = _MdeS*_MdeS*_MdeS*_MdeS;
    return RdeS(s, n, it) + s*RprimaDeS(s, n, it) - RdeS(cero,n, it) -
    numerador/denominador;
}

/*Cálculo de Lambda*/

TFloat lambda(TFloat beta, int n, list<double>::iterator it){
    size_t _precision = beta.precision();
    TFloat res(0.0, _precision);
    TFloat cero(0.0, _precision);
    return (MdeS(beta, n, it)/((MsombreroDeS(beta,n,it)-
            MsombreroDeS(cero,n,it)*MdeS(beta, n, it))*beta));
}

/*Cálculo de sigma*/

TFloat sigma(TFloat beta, TFloat lambda, int n, list<double>::iterator it){
    size_t _precision = beta.precision();
    TFloat base(0.0, _precision);
        double exponente = 1;
        double beta1 = beta.dbl();

        base = MdeS(beta, n, it)/lambda;
    exponente = 1/beta1;
    double base1 = base.dbl();
    double resultD = pow(base1, exponente);
    TFloat resultTF = TFloat(resultD, _precision);
    return resultTF;
}
```

formulas.h

```
#ifndef FORMULAS_
#define FORMULAS_

#include <cmath>
#include <iostream>
#include "TFloat.h"
using namespace std;

/*Cálculo de f(s) y f'(s) (f(s) es la funcion (5) igualada a 0)*/

TFloat fDeS(TFloat s, int n, list<double>::iterator it);

TFloat fPrimaDeS(TFloat s, int n, list<double>::iterator it);
```

```
/*Cálculo de Lambda*/

TFloat lambda(TFloat beta, int n, list<double>::iterator it);

/*Cálculo de sigma*/

TFloat sigma(TFloat beta, TFloat lambda, int n, list<double>::iterator it);

#endif
```

7. Referencias

- http://departamento.us.es/edan/php/asig/LICFIS/LFIPC/Tema8FISPC0809.pdf
- BURDEN, RICHARD L. ; Análisis numérico, 7ma ed. 2002. México, Thomson Learning.
- http://es.wikipedia.org/wiki/Metodo_de_Newton
- http://www.slideshare.net/mfatela/cambio-de-base-en-funciones-exponenciales