2022

COMPUTACIÓN II: PARCIAL 2022

INFORME CON LOS RESULTADOS OBTENIDOS

PABLO GRADOLPH OLIVA | UNIVERSIDAD AUTÓNOMA DE MADRID

PROBLEMA 1

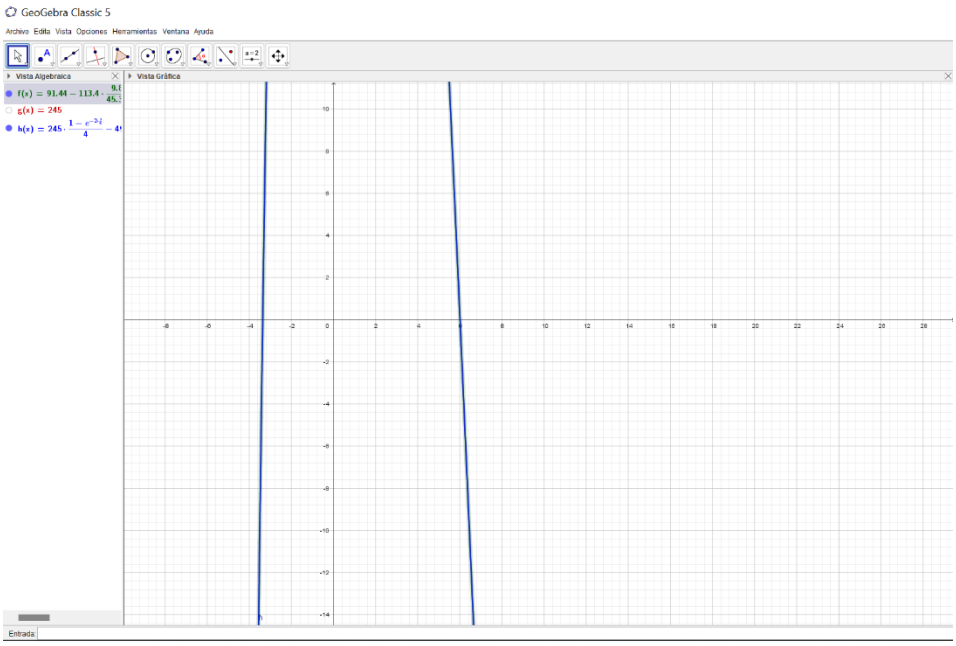
Tras simplificar la ecuación planteada en el enunciado, se obtiene la siguiente ecuación, algo más sencilla:

$$x(t) = \frac{245(1 - e^{-\frac{2t}{5}})}{4} - \frac{49t}{2} + \frac{2286}{25}$$

Cuya derivada respecto al tiempo es:

$$x'(t) = \frac{49 * e^{-2t/5}}{2} - \frac{49}{2}$$

Podemos comprobar mediante representación gráfica que son la misma función. Tras el cálculo de estas ecuaciones y la representación gráfica de la primera de ellas, vemos que tiene dos raíces, una en -3,7 o -3,8 aproximadamente y otra alrededor de 6. Como el tiempo es positivo, nos interesa llegar a la raíz positiva (alrededor de 6). Vemos aquí la representación gráfica:



Los valores iniciales que he utilizado para los distintos métodos son los siguientes:

- Bisección: t0 = 5.5 y t1 = 6.5.
- Secante: t0 = 5.0 y t1 = 5.5.
- Newton: Para el método de Newton con el punto inicial que se pide, el método no converge, por esta razón he elegido un punto inicial más cercano a la raíz, en este caso t0 = 5.5.

Tras la resolución del problema para los distintos métodos obtenemos, para una precisión de 10^{-6}, que la bisección calcula la raíz en 23 iteraciones, la secante lo hace en 3 y Newton en 3 también, jugando con el valor de los puntos iniciales se obtendrían iteraciones distintas. Para esta precisión, la raíz la encuentran para el valor del tiempo t = 6.00599 (tiempo que tarda el

objeto en caer a la superficie). Si aumentamos la precisión, el número de iteraciones también varía.

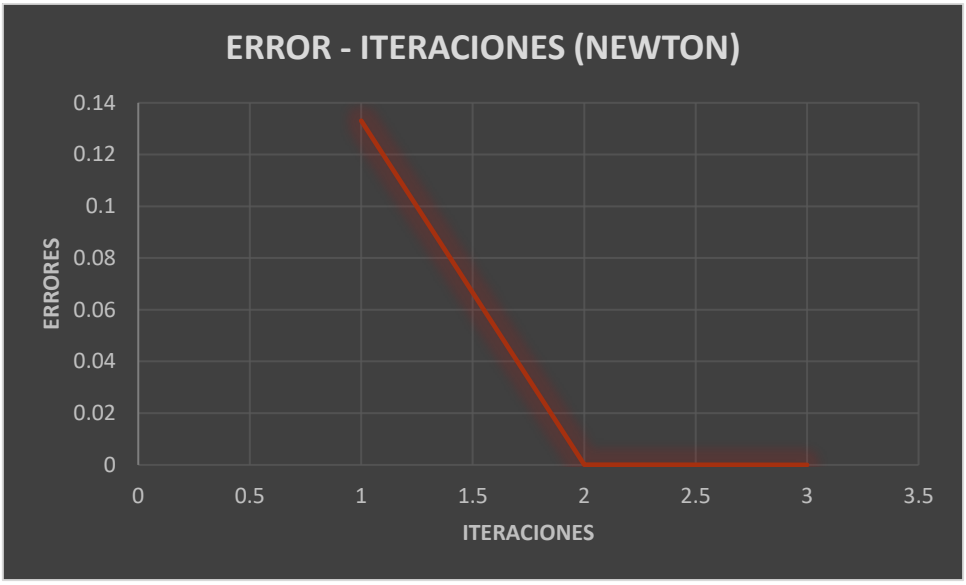
La diferencia entre el método de Newton y Secante (ya que tienen el mismo número de iteraciones) la encontramos en que el valor x(t) es mucho más cercano a cero, del orden de 10^{-13} mientras que en el método de la secante el valor x(t) es del orden de 10^{-7}. Comprobar ejecutando el programa "ParcialProblema1.cpp".

Respecto al error cometido en cada iteración para los distintos métodos, veamos las gráficas. Los errores los puedes ver al ejecutar el programa debajo de cada método, cada línea es la siguiente iteración. No he podido guardarlo en un fichero puesto que he hecho funciones recursivas y entonces en cada iteración se me sobrescribía el fichero.

Gráfica del método de la bisección:

Gráfica del método de la secante:

Gráfica del método de Newton:



Vemos como por el método de la bisección necesitamos muchas más iteraciones, puesto que el error va bajando en cada iteración mucho más despacio que en los otros dos métodos. En cuanto a los otros dos métodos, pese a que la gráfica es muy similar y no se aprecia la diferencia, si nos vamos a los valores exactos, vemos como por el método de Newton el error decrece mucho más rápido, y si pusiésemos una tolerancia más cercana a cero, por el método de la secante necesitaríamos más iteraciones que por el método de Newton.

PROBLEMA 2

He calculado los valores de a1, a2 y a3 por el método de Gauss-Seidel con una precisión de 10^{-6}. Con un vector nulo como valor inicial se obtiene la siguiente matriz de soluciones:

$$\begin{bmatrix} a1 \\ a2 \\ a3 \end{bmatrix} = \begin{bmatrix} 5.06936e-06 \\ 2.00213e-09 \\ -1.41015e-11 \end{bmatrix}$$

Se puede comprobar el resultado ejecutando el código “ParcialProblema2.cpp”. Esta solución se consigue tras 79 iteraciones. En cuanto al error cometido en cada iteración, se puede ver la tabla con los valores en el fichero “ErroresIteracionesGauss.txt”. Aquí dejo la representación gráfica:

Podemos ver que en todas las iteraciones el error cometido es muy bajo (del orden de 10^{-3} desde la primera iteración). También podemos comprobar que a medida que vamos iterando este error se va reduciendo, como hemos puesto una tolerancia de 10^{-6}, el error cometido en la última iteración es del orden de 10^{-7}.

Usamos el método de Gauss-Seidel para acelerar la convergencia del método de Jacobi. El problema de usar el método de Jacobi para este problema es precisamente la convergencia. Al tratarse de una matriz no diagonal dominante no podemos asegurar la convergencia para ambos métodos, y en este problema no podemos conseguir una matriz diagonal dominante de ninguna manera puesto que los valores más grandes se encuentran en la tercera columna de cada fila. Por esta razón, en el caso de este problema, el método de Jacobi no sirve para resolver el sistema (También lo he comprobado con código y el programa se queda iterando indefinidamente).