Taller #1
Análisis Numérico

Breayann Ortíz Aldana, Brayan Ricardo García, Andrés Díaz del Castillo
Pontificia Universidad Javeriana
a.diazdelcastillo@javeriana.edu.co , brayan-garcia@javerianae.edu.co, breayanortiz@javeriana.edu.co

I. INTRODUCCION

Mediante el uso de distintos métodos numéricos y su aplicación usando algoritmos para programar en el lenguaje de programación, C++ o Python se resuelven los distintos ejercicios planteados.

II. DESARROLLO

Ejercicio 1.

Para cada una de los siguientes polinomios, hallar P(x) en el valor indicado y el número de operaciones mínimo para hacerlo (sugerencia utilizar el algoritmo Horner).

Solución: El archivo uno.cpp que contiene el algoritmo con la solución se encuentra en esta carpeta del repositorio. En primer lugar, el programa le pide al usuario ingresar el grado del polinomio que desea utilizar.
El programa almacena en dos vectores el grado del polinomio ingresado, posteriormente se le pide ingresar al usuario los coeficientes que acompañaran a las variables con el grado de polinomio ingresado en el primer paso. Ya con los coeficientes y el grado del polinomio ingresados, el programa procede mostrando el polinomio en pantalla.
A continuación, el programa le pide al usuario el valor con el que quiere evaluar el polinomio f(X) el cual está compuesto por los coeficientes ingresados y el grado del polinomio ingreso en el primer paso, y finalmente evalúa este polinomio utilizando el numero ingresado. También existen dos contadores: uno para contar la cantidad de sumas y otro para la cantidad de multiplicaciones que ocurren al evaluar el polinomio. El programa termina al mostrar los valores de los dos contadores la solución del polinomio con el valor evaluado.

Ejercicio 2.

Para el ejercicio número 2 se pide implementar un código, el cual efectúa una serie de operaciones básicas. Además, se pide encontrar la complejidad del código y recorrerlo asignando el valor de 73 a la variable n.

Al implementar el código en R, se puede ver que el resultado de este código al asignar el valor 73 a la variable n, se obtiene este número expresado en una base diferente, específicamente en base 2, con lo cual se comprende que el código convierte un numero decimal a binario.
Recorriendo el algoritmo con n=73 se tiene el siguiente resultado: 1001001

El código utilizado para calcular este número se encuentra adjunto en los archivos de repositorio como Sol#2.
La complejidad de este código viene definida por O (log(n)), esto gracias a que cada interacción el extremo del algoritmo se divide en dos, disminuyendo en cada interacción.

Ejercicio 3.

Utilizando el método de newton se propone el siguiente ejercicio.

Una partica se mueve en el espacio con el vector de posición R(t)=(2*cos(t),sen(t)). Se requiere conocer el tiempo en el que el objeto se encuentra mas cerca del punto **P**(2,1). Utilice el método de newton con cuatro decimales de precisión.

Para afrontar este problema, primero se encuentra la raíz de la siguiente función:

$$f(x) = \sqrt{1 - \frac{x^2}{4}} - \frac{x}{2}$$

Esta función se obtiene de quitar el parámetro t de la función dada y luego encontrar el punto de corte con la recta y=0.5x, esta recta se dedujo gráficamente, ya que al graficarla función parametrizada, y trazar una recta desde el origen hasta el punto **P**, se puede apreciar que la intersección es el punto más cerca.
La raíz de la función da como resultado, con el método de newton:

Raíz función f(x) = 1.414214

Una vez se encuentra este punto se procede a calcular la coordenada y o imagen de dicho punto, con lo cual se obtiene que:

$$f(1.414214) = 0.707106$$

Con estas dos coordenadas es posible encontrar el parámetro t, para el cual esta condición se cumple resolviendo las siguientes ecuaciones:

$$2 * \cos(t) = 1.414214 \ \ y \sin(t) = 0.707106$$

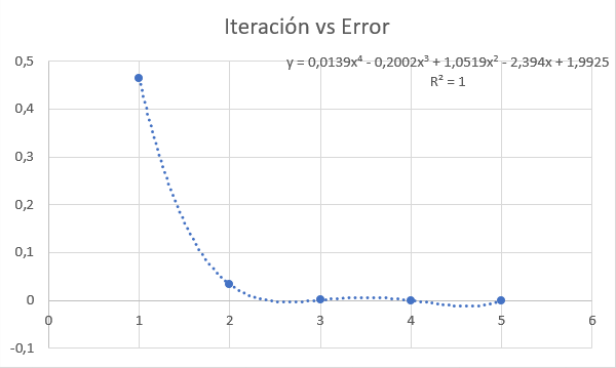
Al resolver estas ecuaciones se tiene como resultado que el parámetro t en ese punto tiene un valor de:

$$t = 0.7854$$

Y ya que se sabe que este parámetro corresponde al tiempo se asume que el resultado está en segundos.

El código necesario para llegar a la solución se encuentra adjunto en los archivos del repositorio como Sol#3.

Analizando la gráfica de convergencia se encuentra que el código tiene convergencia tipo polinomial de grado 4. La grafica que corrobora este hecho se presenta a continuación, en donde se aprecia el tipo de convergencia.

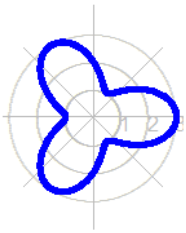


Grafica 1.

Ejercicio 4.

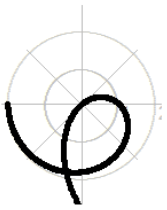
Se pide encontrar los puntos de intersección de dos curvas polares y también la gráfica de estas. Para encontrar la gráfica de estas ecuaciones se utiliza la función en R dada. Esta función se encuentra en el repositorio como EjercicioCuatro.rmd. Para que esta función sirva, se debe definir la ecuación 1 (r=2+cos3t) como r<- 2+cos(3*(dim)) y la ecuación (r=2-e^t) como s<- 2-exp(dim). Una vez ingresadas las ecuaciones, se grafican utilizando polar(dim,**ecuación**,"blue"). Se debe reemplazar **ecuación** por r o s, dependiendo de la curva polar que se quiera graficar.

Grafica de r=2+cos3t



Gráfica 2

Gráfica de r=2-e^t



Grafica 3

Para encontrar los puntos de intersección de las curvas, se pasa de coordenadas polares a cartesianas utilizando el algoritmo que se encuentra en el archivo polar2cart.rmd. Una vez las ecuaciones se encuentren en coordenadas polares, se pueden utilizar los distintos métodos numéricos para encontrar las raíces y así sus puntos de intersección.

Método 1 para encontrar intersección: Uso de algoritmo en RSTUDIO para Hallar raíces con un margen de error 1e-3.

```
f1 <- function(x)  2+cos(3t)
f2  <- function(x) 2 - exp^t

curve(f1,from=1e-3,to=1e8,log="xy")
curve(f2,add=TRUE,col=2)

uniroot(function(x) f1(x)-f2(x),c(10,1e8))$root
```

Método 2 para encontrar intersección: Método de bisección

Para poder determinar los puntos de intersección de las ecuaciones f(x)= 2+cos(3t) y g(x)= 2 – exp^t simplemente podemos igualar f(x) a g(x). Esto funciona ya que estamos buscando coordenadas que satisfagan ambas ecuaciones simultáneamente. Para encontrar las coordenadas, se reescribe las ecuaciones de la siguiente manera: f(x) = g(x) se vuelve f(x) - g(x) = 0 para encontrar las raíces y las coordenadas donde se intersectan.

Se encuentra que el punto de intersección de las curvas es (-0.2498 , -2.3046)

Ejercicio 5.13

Para obtener la enésima raíz de un número, utilizamos el método de Newton- Rhapson. A continuación, está el procedimiento.

El método de Newton- Rhapson establece que

x(k+1) = x(k) - f(x) / f'(x)

f(x) = x^(n) – a

Entonces f'(x) = n*x^(n-1)

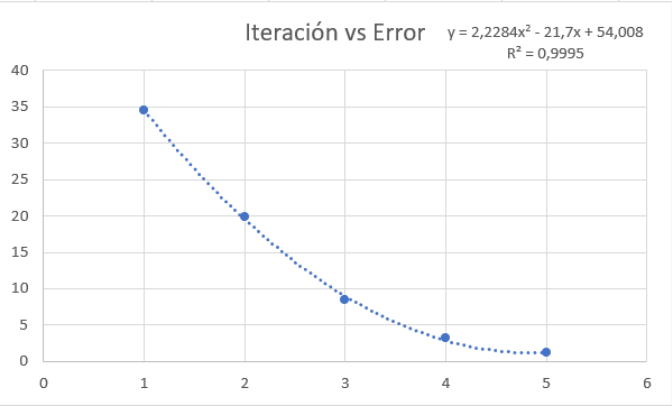
x(k) es el valor de x en la k-esima iteración.
Al simplificar obtenemos la siguiente fórmula para encontrar la enésima raíz de un numero utilizando el k número de iteraciones.

x (k+1) = (1/n) * ((n-1) * x(k) + A/ x(k) ^ (n-1))

Ejercicio 5.14

A) El interval solo debe contener una raiz de la funcion.

B) El orden de convergencia es cuadrático ya que al graficar el error podemos graficarlo con respecto a la iteración y ver su tipo de convergencia. Al ver la línea de tendencia podemos evidenciar una correlación del 99,95% usando el programa Excel.



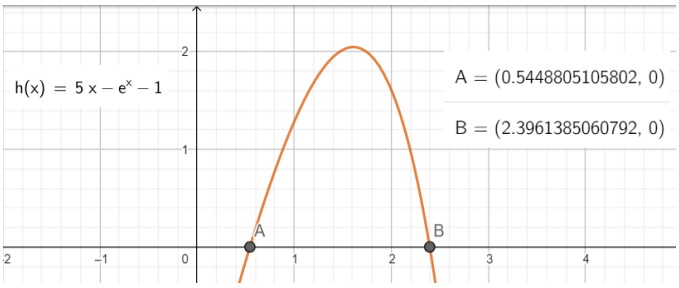
Grafica 4

Ejercicio 15.
La solucion de este algoritmo se encuentra adjunta en el repositorio como Sol#15.

15.a

1. $\int_0^x (5 - e^u)du = 2$
2. $5x\text{-}e^x\text{-}1=0$

15.b



15.c

g(x)-x=0

Intervalo $0 \leq k < 1$

15.d

```
X= 1            Error= 34.47071        i:
1
X= 0.7436564    Error= 19.80494        i:
2
X= 0.6207226    Error= 8.507626        i:
3
X= 0.5720544    Error= 3.188003        i:
4
X= 0.5543807    Error= 1.132524        i:
5
```