



Actividad 3 # Bisección

Métodos Numéricos

Ingeniería en Desarrollo de Software



TUTOR: MIGUEL ANGEL RODRIGUEZ VEGA

ALUMNO: GUSTAVO ALONSO ESPINOZA ROMERO_A3

FECHA: 10/03/2024

Índice

INTRODUCCION	3
DESCRIPCION	4
JUSTIFICACION.....	5
DESARROLLO	6
Método de jacobi	6
Método de gauss sidel.....	9
Método de bisección.....	12
CONCLUSION	15

INTRODUCCION

Para esta última actividad, vamos a realizar el método de bisección, el de el de jacobi y gauss-seidel, todos estos van a ser ejecutados en rstudio, la herramienta que nos ha sido ayudando con nuestras actividades, demostrando que es un programa en el cual se puede uno apoyar a la hora de querer resolver algún problema que tengamos. La ecuación que nos brindan será la que resolveremos con los métodos que nos están solicitando. La ejecución de los métodos será mostrada en el apartado de desarrollo, en dónde podremos ver los pasos que se realizaron para llegar a la solución que se nos esta solicitando en nuestra actividad. Tenderemos también la respuesta de una pregunta que se nos hace, las cuales son: ¿Cuál es el método que resultó más fácil de utilizar? 2. Y ¿Cuál es el método más eficiente? ¿Por qué?

Dando la explicación con nuestras propias palabras de que nos a parecido el resolver nuestra ecuación con estos métodos, y también escribir cual ha sido la mejor opción.

DESCRIPCION

Los métodos que usaremos tienen algunas diferencias, pero todas tienen el mismo objetivo, el cual es resolver y llegar a una solución o aproximación, el método de gauss-seidel es iterativo, con lo cual podemos resolver sistemas de ecuaciones lineales, este método trata de resolver problemas mediante las aproximaciones que te da, el método jacobi hace exactamente lo mismo, solo que este empieza desde una estimación inicial, donde se empezará a resolver y por último el de bisección este es un el el que muchos querrían usar, siendo este un método muy fácil de utilizar, pero que también tiene una gran desventaja, la cual es que toma muchas iteraciones para poder llegar a la aproximación o a la solución. Sabiendo esto se demuestra lo que cada una de ellas es capaz de aportarnos para poder llegar al objetivo o solución, teniendo ciertas ventajas y desventajas de cada una de ellas.

JUSTIFICACION

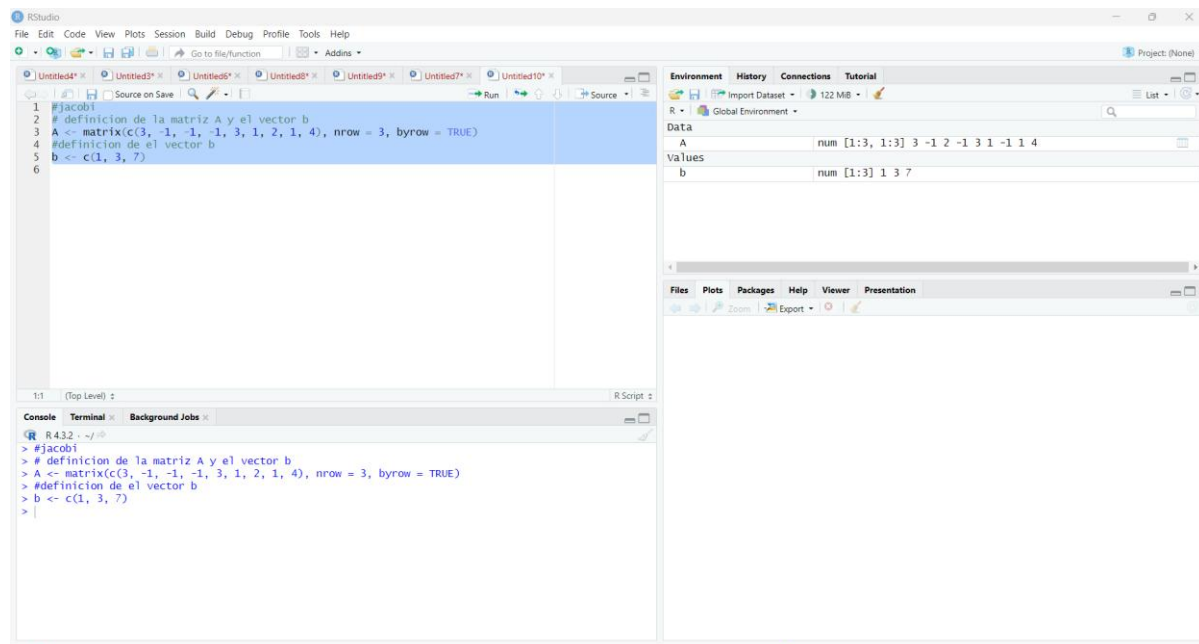
Los métodos que se utilizan, son muy útiles para resolver problemas con números que sean muy grandes, siendo complicado poder resolverlos, con los métodos tenemos la facilidad de que podemos aproximarnos a la solución que estamos buscando o simplemente que podamos llegar a la solución, todos los métodos que se aprendieron nos van a servir de gran ayuda en el momento que se nos presente algún problema que queramos sacarle una solución,

El objetivo de la actividad fue que aprendiéramos y conociéramos los diferentes métodos que hay, y así poder tener una variedad de conocimientos, teniendo diferentes alternativas a la hora de que queramos utilizar alguno, o que necesitemos hacer uso de alguno. Cada uno de ellos tienen sus complicaciones y es mejor uno que otro, pero al final de todo el objetivo es el mismo, el cual es llegar a la solución de algún problema matemático que necesite solución.

DESARROLLO

Método de jacobi

En esta parte podemos ver la matriz y el vector que necesitamos para empezar a usar método de Jacobi y llegar a la solución que estamos buscando, esta matriz y vector contienen los valores de la función que se solicita resolver.



```
1 #jacobi
2 #definición de la matriz A y el vector b
3 A <- matrix(c(3, -1, -1, -1, 3, 1, 2, 1, 4), nrow = 3, byrow = TRUE)
4 #definición de el vector b
5 b <- c(1, 3, 7)
6
```

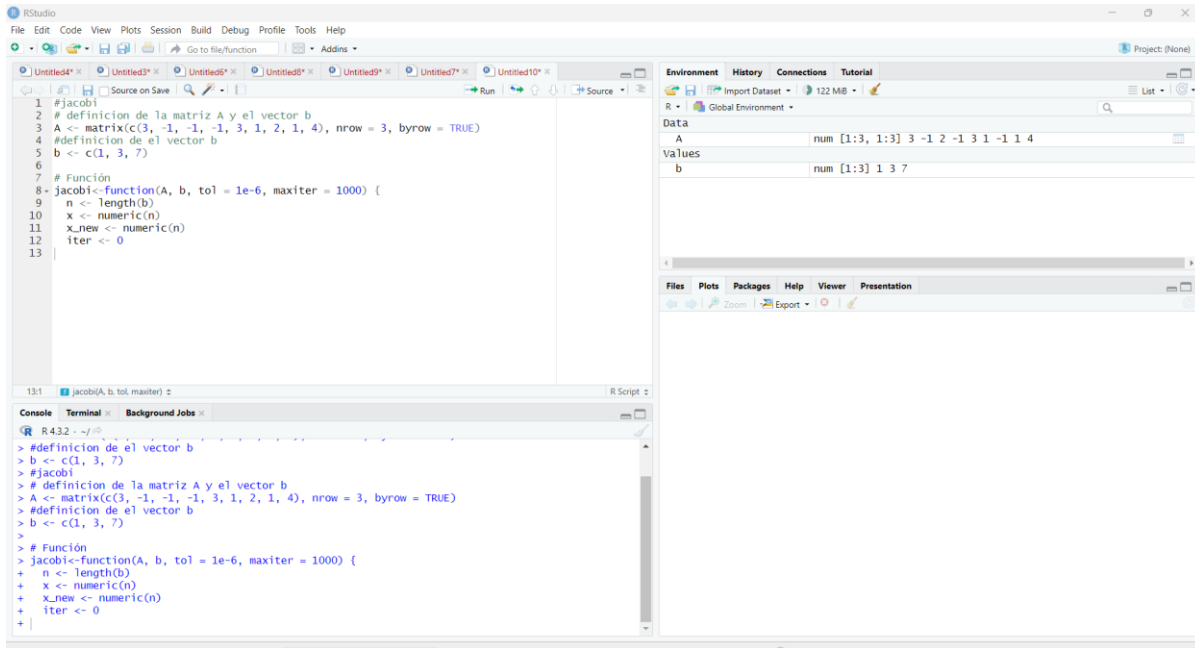
Environment

Object	Class	Attributes
A	matrix	[1:3, 1:3] 3 -1 2 -1 3 1 -1 1 4
b	vector	[1:3] 1 3 7

Console

```
R 4.3.2 ~ ./
> #jacobi
> #definición de la matriz A y el vector b
> A <- matrix(c(3, -1, -1, -1, 3, 1, 2, 1, 4), nrow = 3, byrow = TRUE)
> #definición de el vector b
> b <- c(1, 3, 7)
>
```

Aquí colocamos función, el máximo de iteraciones que queremos que se realizan, y el error permitido.



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for a Jacobi iteration function and its execution.
- Environment:** Shows the global environment with variables `A` and `b`.
- Console:** Shows the output of the executed code.

```
1 #jacobi
2 # definicion de la matriz A y el vector b
3 A <- matrix(c(3, -1, -1, -1, 3, 1, 2, 1, 4), nrow = 3, byrow = TRUE)
4 #definicion de el vector b
5 b <- c(1, 3, 7)
6
7 # Función
8 jacobi<-function(A, b, tol = 1e-6, maxiter = 1000) {
9   n <- length(b)
10  x <- numeric(n)
11  x_new <- numeric(n)
12  iter <- 0
13 }
```

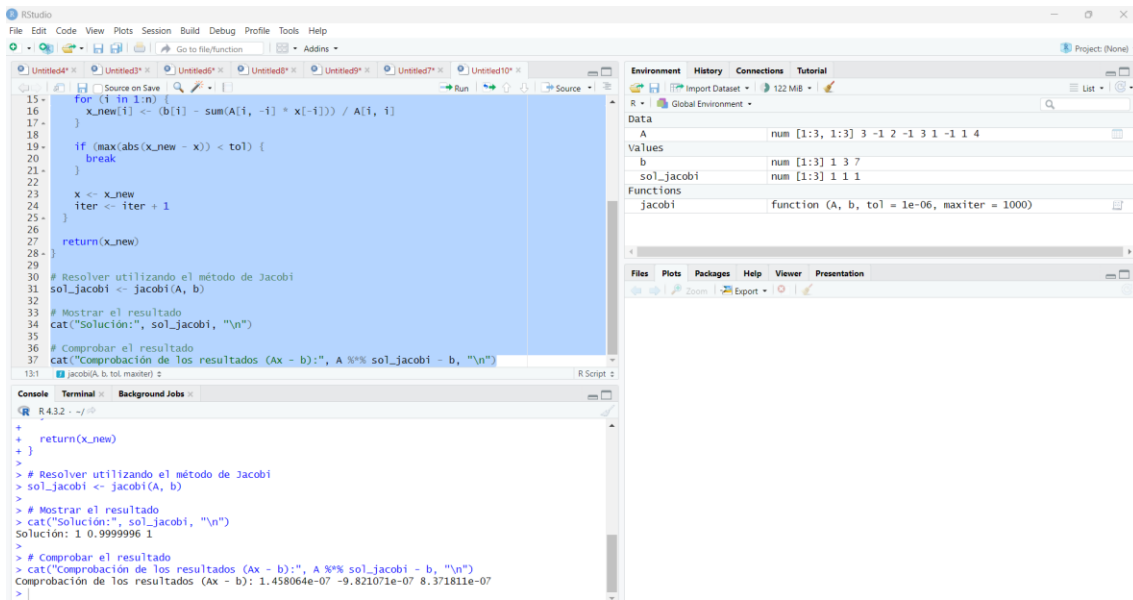
Environment:

Variable	Class	Value
A	num [1:3, 1:3]	3 -1 2 -1 3 1 -1 1 4
b	num [1:3]	1 3 7

Console:

```
> #definicion de el vector b
> b <- c(1, 3, 7)
> #jacobi
> # definicion de la matriz A y el vector b
> A <- matrix(c(3, -1, -1, -1, 3, 1, 2, 1, 4), nrow = 3, byrow = TRUE)
> #definicion de el vector b
> b <- c(1, 3, 7)
>
> # Función
> jacobi<-function(A, b, tol = 1e-6, maxiter = 1000) {
+   n <- length(b)
+   x <- numeric(n)
+   x_new <- numeric(n)
+   iter <- 0
+ }
```

Se usaron el ciclo “FOR” y el “IF”, los cuales permiten que hagamos las iteraciones hasta llegar a la solución, después se le da la orden de que imprima los resultados en cada iteración y al final mostrar la solución.



```
15- for (i in 1:n) {
16-   x_new[i] <- (b[i] - sum(A[i, -i] * x[-i])) / A[i, i]
17- }
18- if (max(abs(x_new - x)) < tol) {
19-   break
20- }
21- }
22- x <- x_new
23- iter <- iter + 1
24- }
25- }
26- return(x_new)
27- }
28- }
29- }
30- # Resolver utilizando el método de Jacobi
31- sol_jacobi <- jacobi(A, b)
32- }
33- # Mostrar el resultado
34- cat("Solución:", sol_jacobi, "\n")
35- }
36- # Comprobar el resultado
37- cat("Comprobación de los resultados (Ax - b):", A %*% sol_jacobi - b, "\n")
38- }
```

Environment

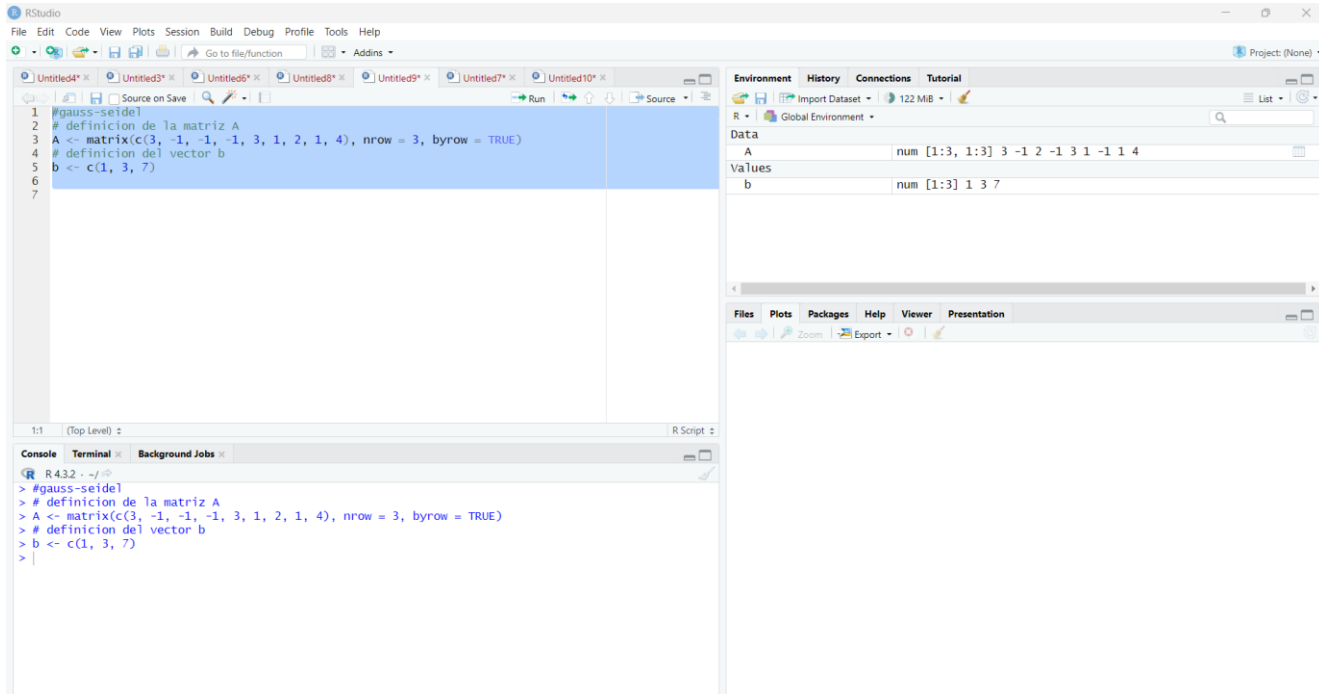
Variable	Class	Value
A	num	[1:3, 1:3] 3 -1 2 -1 3 1 -1 1 4
b	num	[1:3] 1 3 7
sol_jacobi	num	[1:3] 1 1 1
jacobi	function	function (A, b, tol = 1e-06, maxiter = 1000)

Console

```
> return(x_new)
> }
> # Resolver utilizando el método de Jacobi
> sol_jacobi <- jacobi(A, b)
> # Mostrar el resultado
> cat("Solución:", sol_jacobi, "\n")
Solución: 1 0.9999996 1
> # Comprobar el resultado
> cat("Comprobación de los resultados (Ax - b):", A %*% sol_jacobi - b, "\n")
Comprobación de los resultados (Ax - b): 1.458064e-07 -9.821071e-07 8.371811e-07
>
```


Método de gauss sidel

En esta parte podemos ver la matriz y el vector que necesitamos para empezar a usar método de gauss seidel y llegar a la solución que estamos buscando, esta matriz y vector contienen los valores de la función que se solicita resolver.



The screenshot shows the RStudio interface. The script editor on the left contains the following R code:

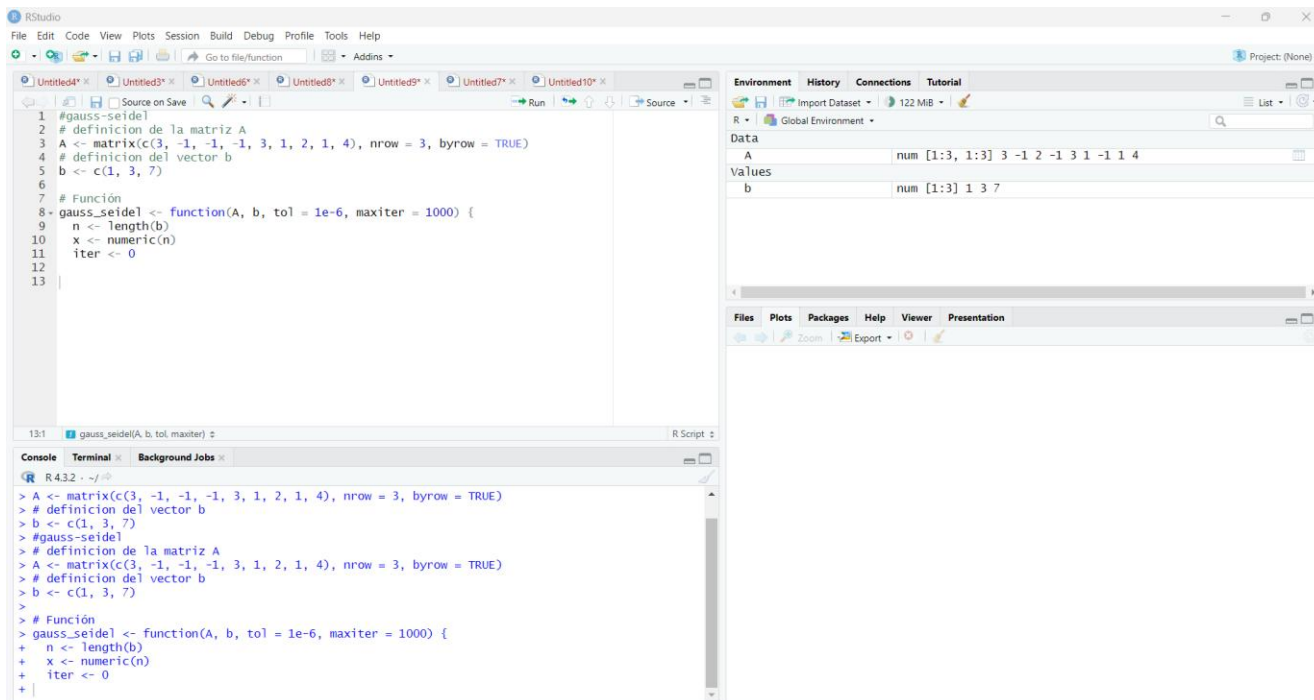
```
1 #gauss-seidel
2 # definicion de la matriz A
3 A <- matrix(c(3, -1, -1, -1, 3, 1, 2, 1, 4), nrow = 3, byrow = TRUE)
4 # definicion del vector b
5 b <- c(1, 3, 7)
6
7
```

The Environment pane on the right shows the objects created in the Global Environment:

Object	Class	Value
A	matrix	[1:3, 1:3] 3 -1 2 -1 3 1 -1 1 4
b	vector	[1:3] 1 3 7

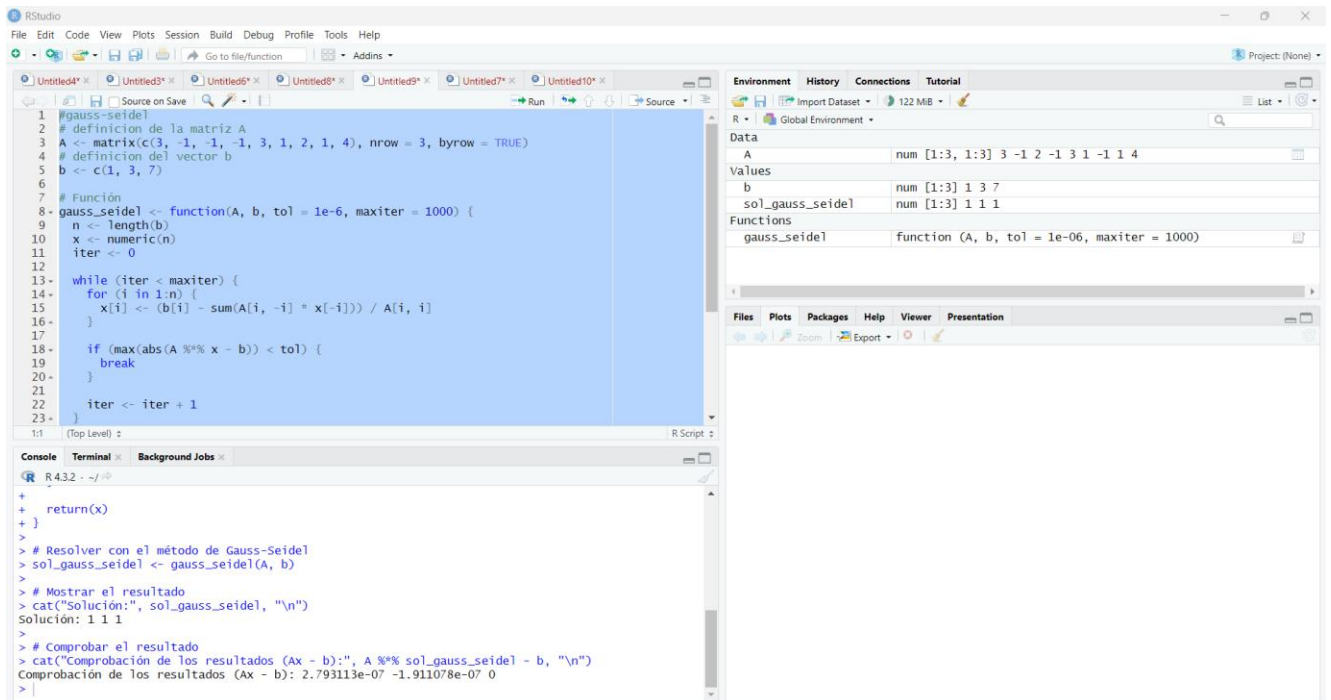
The Console at the bottom shows the execution of the code:

```
> #gauss-seidel
> # definicion de la matriz A
> A <- matrix(c(3, -1, -1, -1, 3, 1, 2, 1, 4), nrow = 3, byrow = TRUE)
> # definicion del vector b
> b <- c(1, 3, 7)
>
```



Aquí colocamos función, el máximo de iteraciones que queremos que se realizan, y el error permitido.

Se usaron el ciclo “WHILE” y la condicional “IF”, los cuales permiten que hagamos las iteraciones, hasta llegar a la solución, después se le da la orden de que imprima los resultados en cada iteración que se hace y al final mostrar la solución.



The screenshot displays the RStudio interface with a script editor, an environment pane, and a console. The script defines a function `gauss_seidel` that iteratively solves a system of linear equations $Ax = b$ using the Gauss-Seidel method. The function takes a matrix `A`, a vector `b`, a tolerance `tol`, and a maximum number of iterations `maxiter`. It uses a `while` loop to iterate until the solution converges within the specified tolerance. The console shows the execution of the function and the resulting solution vector `sol_gauss_seidel`.

```
1 #gauss-seidel
2 #definición de la matriz A
3 A <- matrix(c(3, -1, -1, -1, 3, 1, 2, 1, 4), nrow = 3, byrow = TRUE)
4 #definición del vector b
5 b <- c(1, 3, 7)
6
7 # Función
8 gauss_seidel <- function(A, b, tol = 1e-6, maxiter = 1000) {
9   n <- length(b)
10  x <- numeric(n)
11  iter <- 0
12
13  while (iter < maxiter) {
14    for (i in 1:n) {
15      x[i] <- (b[i] - sum(A[i, -i] * x[-i])) / A[i, i]
16    }
17
18    if (max(abs(A %*% x - b)) < tol) {
19      break
20    }
21
22    iter <- iter + 1
23  }
24
25  return(x)
26 }
```

Environment pane:

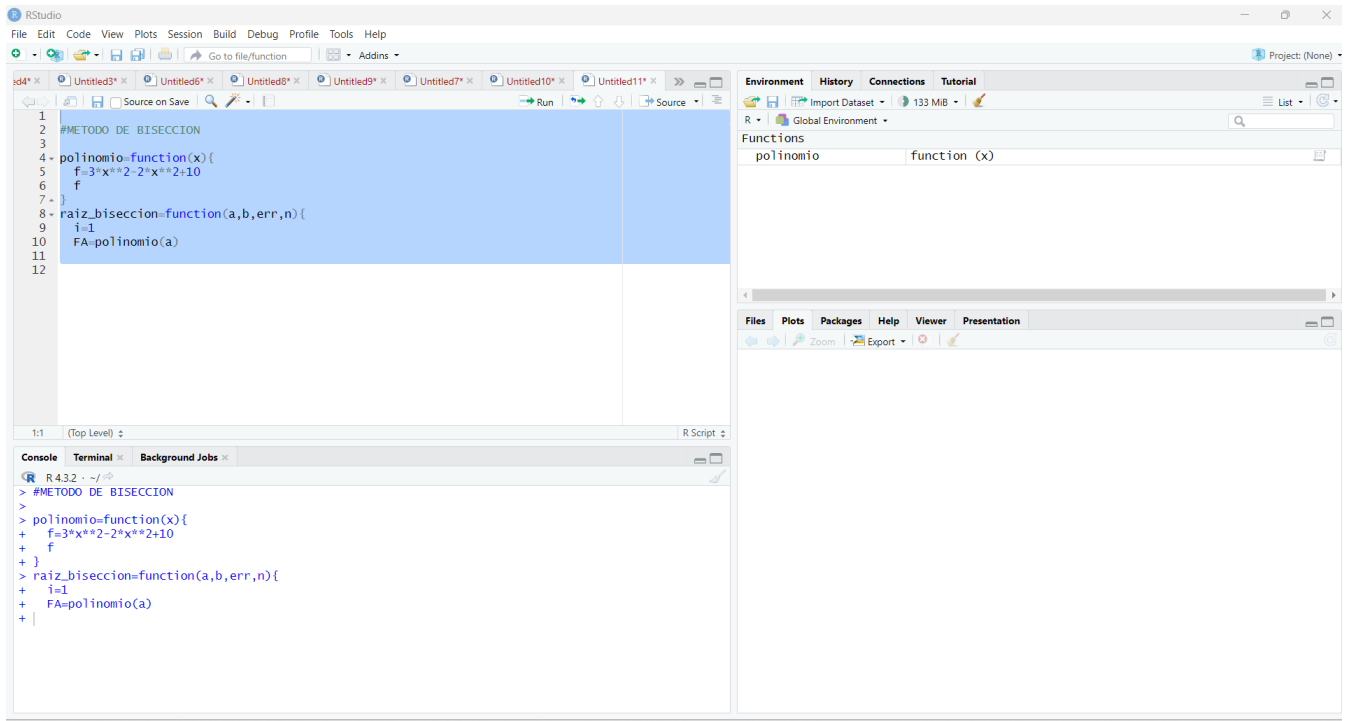
Object	Class	Attributes
A	matrix	[1:3, 1:3] 3 -1 -1 -1 3 1 2 1 4
b	numeric	[1:3] 1 3 7
sol_gauss_seidel	numeric	[1:3] 1 1 1
gauss_seidel	function	function (A, b, tol = 1e-06, maxiter = 1000)

Console:

```
> # Resolver con el método de Gauss-Seidel
> sol_gauss_seidel <- gauss_seidel(A, b)
> # Mostrar el resultado
> cat("Solución:", sol_gauss_seidel, "\n")
Solución: 1 1 1
>
> # Comprobar el resultado
> cat("Comprobación de los resultados (Ax - b):", A %*% sol_gauss_seidel - b, "\n")
Comprobación de los resultados (Ax - b): 2.793113e-07 -1.911078e-07 0
>
```

Método de bisección

Aquí colocamos la función, la cual es: $3x^2 - 2x^2 + 10$ y también colocamos donde se almacenan los valores iniciales, el margen de error y el numero de iteraciones.



The screenshot shows the RStudio interface with the following components:

- Source Editor:** Contains the R script for the bisection method.

```
1  
2 #METODO DE BISECCION  
3  
4 polinomio=function(x){  
5   f=3*x^2-2*x^2+10  
6   f  
7 }  
8 raiz_biseccion=function(a,b,err,n){  
9   i=1  
10  FA=polinomio(a)
```
- Environment:** Shows the function `polinomio` as a function of `x`.
- Console:** Displays the execution of the script, showing the function definitions being entered into the R environment.

Ponemos los ciclos “WHILE”, la condicional” IF” Y “ELSE”

The screenshot shows the RStudio interface with a script editor, console, and environment pane. The script defines a function `polinomio` and a function `raiz_biseccion` that uses a `while` loop and `if-else` conditional to find the root of a polynomial.

```
11 while(i<n){
12   p=a + (b-a)/2
13   FP=polinomio(p)
14   print(c(i,p))
15
16   if(FP==0 | (b-a)/2<err){
17     return(p)
18   }
19   i=i+1
20
21   if(FP*FA>0){
22     a=p
23     FA=FP
24   } else{
25     b=p
26   }
27 }
28 return(paste("el metodo fallo luego de ",n,"iteraciones"))
29 }
```

The console shows the execution of the `raiz_biseccion` function, displaying the iteration number and the value of the polynomial at each step.

```
R 4.3.2 ~ ./
+ if(FP==0 | (b-a)/2<err){
+   return(p)
+ }
+ i=i+1
+ if(FP*FA>0){
+   a=p
+   FA=FP
+ } else{
+   b=p
+ }
+ return(paste("el metodo fallo luego de ",n,"iteraciones"))
+ }
```

The environment pane shows the functions `polinomio` and `raiz_biseccion` loaded in the Global Environment.

Y finalmente ponemos los valores iniciales, el error y las iteraciones, para que nos arroje las aproximaciones a la solución y su solución.

The screenshot shows the RStudio interface with a script editor, console, and environment pane. The script defines the `polinomio` and `raiz_biseccion` functions. The console shows the execution of the `raiz_biseccion` function with initial values `0, 1, 0.000001, 50`, displaying the iteration number and the value of the polynomial at each step.

```
1 #METODO DE BISECCION
2
3 polinomio=function(x){
4   f=3*x^3+2*x^2+10
5   f
6 }
7
8 raiz_biseccion=function(a,b,err,n){
9   i=1
10  FA=polinomio(a)
11
12  while(i<n){
13    p=a + (b-a)/2
14    FP=polinomio(p)
15    print(c(i,p))
16
17    if(FP==0 | (b-a)/2<err){
18      return(p)
19    }
20    i=i+1
21
22    if(FP*FA>0){
23      a=p
24      FA=FP
25    } else{
26      b=p
27    }
28  }
29  return(paste("el metodo fallo luego de ",n,"iteraciones"))
30 }
```

The console shows the execution of the `raiz_biseccion` function, displaying the iteration number and the value of the polynomial at each step.

```
R 4.3.2 ~ ./
+ if(FP*FA>0){
+   a=p
+   FA=FP
+ } else{
+   b=p
+ }
+ return(paste("el metodo fallo luego de ",n,"iteraciones"))
+ }
> raiz_biseccion(0,1,0.000001,50)
[1] 1.0 0.5
[1] 2.00 0.75
[1] 3.000 0.875
```

The environment pane shows the functions `polinomio` and `raiz_biseccion` loaded in the Global Environment.

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

led4* x Untitled3* x Untitled6* x Untitled8* x Untitled9* x Untitled7* x Untitled10* x Untitled11* x

```
1 #METODO DE BISECCION
2
3
4 polinomio=function(x){
5   f=3*x^3-2*x^2+10
6   f
7 }
8 raiz_biseccion=function(a,b,err,n){
9   i=1
10  FA=polinomio(a)
11
12  while(i<n){
13    p=a + (b-a)/2
14    FP=polinomio(p)
15    print(c(i,p))
16
17    if(FP==0 | (b-a)/2<err){
18      return(p)
19    }
20    i=i+1
21
22    if(FP*FA>0){
23      a=p
24    } else {
25      b=p
26    }
27  }
28 }
```

Environment History Connections Tutorial

R Global Environment 134 MB

Functions

polinomio	function (x)
raiz_biseccion	function (a, b, err, n)

Files Plots Packages Help Viewer Presentation

Console Terminal Background Jobs

R 4.3.2 ~ / ~

```
[1] 7.00000000 0.99999999
[1] 8.00000000 0.9960938
[1] 9.00000000 0.9980469
[1] 10.00000000 0.9990234
[1] 11.00000000 0.9995117
[1] 12.00000000 0.9997559
[1] 13.00000000 0.9998779
[1] 14.00000000 0.999939
[1] 15.00000000 0.9999695
[1] 16.00000000 0.9999847
[1] 17.00000000 0.9999924
[1] 18.00000000 0.9999962
[1] 19.00000000 0.9999981
[1] 20.00000000 0.999999
[1] 0.9999999
>
```

CONCLUSION

Al finalizar la última actividad, se han adquirido los conocimientos de los diferentes métodos que hay, y como es que estos función para ayudar a solucionar problemas matemáticos que son complejos, aprendimos no solo a utilizar estos métodos en Excel, si no también con la herramienta “RSTUDIO” se realizan estos métodos de una manera sencilla aprendiendo a cómo utilizar esta herramienta, estos conocimientos que se aprendieron nos ayudaran mucho en el ámbito laboral, ya que son necesarios a la hora de estar trabajando en cualquier lugar.

Es por eso que fue de suma importancia el aprender cada uno de ellos, porque conocer como es que funciona los métodos, es mucho mas sencillo de saber cual es que te puede funcionar, cual es el que se te facilita más, cual es el mejor de todos, y así poder resolver lo que necesites en tu trabajo sin que tengas ningún tipo de dificultades.

