|  |
| --- |
| title: “R Notebook” |
| output: html\_notebook |
| Names: Jhonier Coronado, Sebastian Valderrama |
|  |

**ALGORITMOS-ANALISIS NÚMERICO**

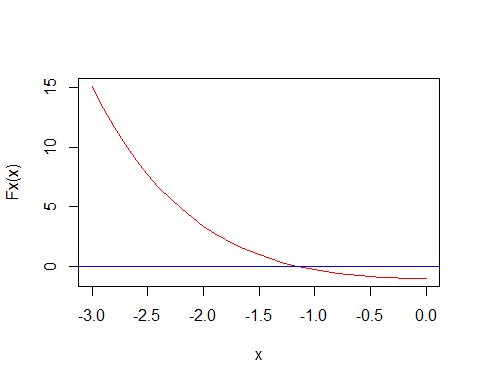
**Bisección**

Este método consiste en obtener una major aproximación de la raíz a partir de un interval inicial (a,b) en el cual hay un cambio de signo en la función, es decir:

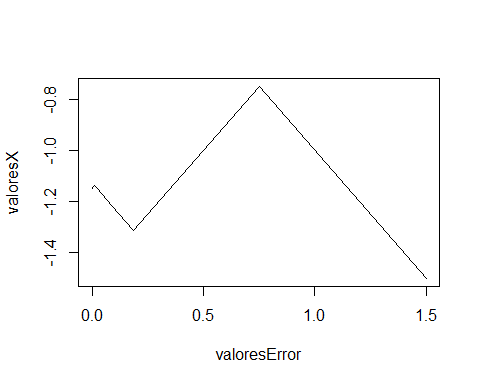
Entonces primero se obtiene un valor medio con respecto a (a,b)

Luego este valor se reemplaza en la función. Si el resultado es menor a 0 y si es mayor . Se iterara hasta que cumpla con la tolerancia requerida. Con respecto a esta definición se construyó el siguiente algoritmo:

rm(list=ls())  
Fx = function(x) (exp(-x) + x -2)  
  
valoresError = c()  
valoresX = c()  
  
biseccion = function(a, b, error){  
 #Para graficar se crea una secuencia de números entre el rango [a,b]  
 x = seq(a, b, 0.1)  
 plot(x, Fx(x), type = "l", col = "red")  
 abline(h = 0, col = "blue")  
 x = b  
 d = (a+b) / 2.0  
 contador = 0  
 e = abs(a-b) /2.0  
 while (e > error){  
 contador = contador + 1  
 if (Fx(a) \* Fx(b) > 0){  
 cat("No se puede aplicar el método\n")  
 }  
 else {  
 if (Fx(x) \* Fx(a) > 0)  
 a = x  
 if (Fx(x) \* Fx(b) > 0)  
 b = x  
 d = x  
 x = (a + b) / 2  
 e = abs(a-b) / 2.0  
   
 valoresX[contador] = x  
 valoresError[contador] = e  
 cat("X: ", x, "\tError: ", e, "\n")  
 }  
 }  
 plot(valoresError, valoresX, type = "l")  
 cat("Iteracciones: ", contador, "Resultado: ", x, "\n")  
}  
  
biseccion(-3, 0, 10e-5)



## X: -1.5 Error: 1.5   
## X: -0.75 Error: 0.75   
## X: -1.125 Error: 0.375   
## X: -1.3125 Error: 0.1875   
## X: -1.21875 Error: 0.09375   
## X: -1.171875 Error: 0.046875   
## X: -1.148438 Error: 0.0234375   
## X: -1.136719 Error: 0.01171875   
## X: -1.142578 Error: 0.005859375   
## X: -1.145508 Error: 0.002929688   
## X: -1.146973 Error: 0.001464844   
## X: -1.14624 Error: 0.0007324219   
## X: -1.145874 Error: 0.0003662109   
## X: -1.146057 Error: 0.0001831055   
## X: -1.146149 Error: 9.155273e-05



## Iteracciones: 15 Resultado: -1.146149

**Método de Newton**

El método de Newton es un método abierto, en el sentido de que no garantiza su convergencia global. La única manera de alcanzar la convergencia es seleccionar un valor inicial lo suficientemente cercano a la raíz buscada.

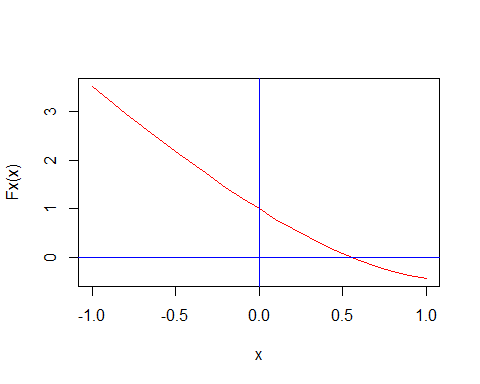
Entonces este método parte de una aproximación inicial y obtiene una aproximación mejor dada por la formula:

De esta manera se ira acercando al valor deseado reemplazando una y otra vez el valor obtenido hasta obtener el error requerido por la tolerancia.

Según lo anterior se contruyo el siguiente algoritmo:

e^x - pi \* x Derivada e^x - pi

rm(list=ls())  
Fx = function(x) exp(x) - pi \* x  
Fx1 = function(x) exp(x) - pi  
  
Newton = function(a, b, error){  
 x = seq(a, b, 0.1)  
 plot(x, Fx(x), type = "l", col = "red")  
 abline(h = 0, v = 0, col = "blue")  
   
 x\_0 = (a + b) / 2  
   
 contador = 0  
 dx = 0  
 repeat {  
 corr = Fx(x\_0) / Fx1(x\_0)  
 x\_1 = x\_0 - corr  
 dx = abs(corr)  
 x\_0 = x\_1  
 contador = contador + 1  
   
 cat(contador, dx, "\n")  
 if (dx <= error)  
 break  
 }  
 cat ("Iteracciones: ", contador, "Resultado: ", x\_1, "\n")  
}  
Newton(-1, 1, 10e-8)



## 1 0.4669422   
## 2 0.08287642   
## 3 0.003998516   
## 4 9.896266e-06   
## 5 6.078286e-11   
## Iteracciones: 5 Resultado: 0.553827

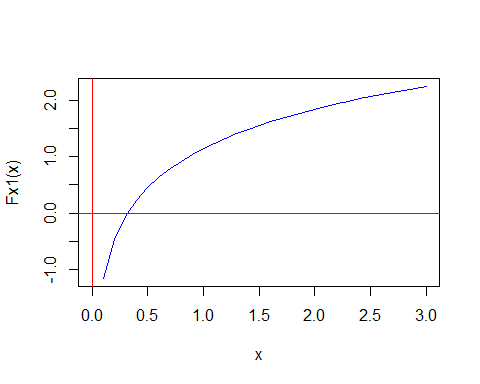
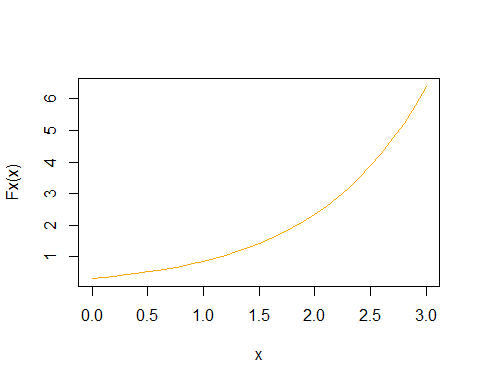
**Método del Punto Fijo**

Un punto fijo de una función *g,* es un número *p* tal que *g(p)=p* . De esta forma se obtiene *g(x)* de *f(x)* y con esto y un punto se realizaran las iteraciones necesarias para hallar el resultado hasta la tolerancia requerida.

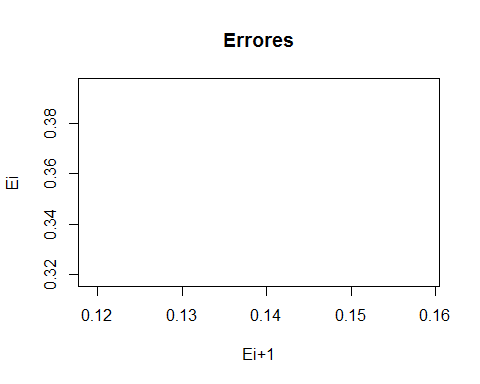
Entonces viendo esto como un proceso primero se debe obtener *g(x)* y escoger un punto . Luego se reeplazara en *g(x)* ; si este valor es el primero no se obtendra un error, y si no se obtendra el error calculandolo con el valor obtenido anteriormente. Finalmente el valor obtenido al reemplazar en *g(x)* será y se repite el proceso otra vez.

Según la anterior definición se conntruyo el siguiente algoritmo:

rm(list=ls())  
Fx = function(x) exp(x) / pi  
Fx1 = function(x) log(x\*pi)  
  
PuntoFijo = function(a, b, error){  
 errorX = 0  
 xInicial = a  
 x = seq(a, b, 0.1)  
 plot(x, Fx(x), type = "l", col = "orange")  
 plot(x, Fx1(x), type = "l", col = "blue")  
 abline(h = 0, v = 0, col = "red")  
 if (Fx(a) < a || Fx(b) < b)  
 cat("El intervalo no es valido\n")  
 else {  
 x\_0 = (a + b) / 2  
 contador = 0  
 fxInicial = Fx(a)  
 done = FALSE  
 valoresX = c()  
 erroresX = c()  
 erroresX1 = c()  
 it = c()  
 x = 0  
 while(abs(xInicial - fxInicial) > error){  
 x = x + 1  
 contador = contador + 1  
 if (xInicial < a){  
 done = TRUE  
 }  
 if (done == FALSE){  
 xInicial = fxInicial  
 fxInicial = Fx(xInicial)  
 } else {  
 fxInicial = xInicial  
 xInicial = Fx1(fxInicial)  
 }  
 cat("Iteraccion: ", contador, "\tValor de X: ", xInicial, "\t\tError: ", errorX, "\n")  
 errorX = xInicial - errorX  
   
 valoresX[x] = xInicial  
 i = 0  
 if (x%%2 == 0){  
 erroresX1[x] = errorX  
 } else{  
 erroresX[x] = errorX  
 }  
 it[x] = contador  
 }  
 erroresX = erroresX[-contador]  
 plot(erroresX1, erroresX, type="l",xlab="Ei+1", ylab="Ei", main="Errores", col="red")  
   
 cat("Iteracciones: ", contador, "Resultado: ", xInicial, "\n")  
   
 }  
}  
  
PuntoFijo(0, 3, 10e-8)



## Iteraccion: 1 Valor de X: 0.3183099 Error: 0   
## Iteraccion: 2 Valor de X: 0.4376131 Error: 0.3183099   
## Iteraccion: 3 Valor de X: 0.4930638 Error: 0.1193033   
## Iteraccion: 4 Valor de X: 0.5211767 Error: 0.3737605   
## Iteraccion: 5 Valor de X: 0.5360364 Error: 0.1474162   
## Iteraccion: 6 Valor de X: 0.5440612 Error: 0.3886202   
## Iteraccion: 7 Valor de X: 0.5484448 Error: 0.155441   
## Iteraccion: 8 Valor de X: 0.5508542 Error: 0.3930038   
## Iteraccion: 9 Valor de X: 0.5521831 Error: 0.1578504   
## Iteraccion: 10 Valor de X: 0.5529173 Error: 0.3943326   
## Iteraccion: 11 Valor de X: 0.5533234 Error: 0.1585847   
## Iteraccion: 12 Valor de X: 0.5535482 Error: 0.3947388   
## Iteraccion: 13 Valor de X: 0.5536726 Error: 0.1588094   
## Iteraccion: 14 Valor de X: 0.5537415 Error: 0.3948632   
## Iteraccion: 15 Valor de X: 0.5537797 Error: 0.1588783   
## Iteraccion: 16 Valor de X: 0.5538008 Error: 0.3949013   
## Iteraccion: 17 Valor de X: 0.5538125 Error: 0.1588995   
## Iteraccion: 18 Valor de X: 0.553819 Error: 0.394913   
## Iteraccion: 19 Valor de X: 0.5538226 Error: 0.158906   
## Iteraccion: 20 Valor de X: 0.5538246 Error: 0.3949166   
## Iteraccion: 21 Valor de X: 0.5538257 Error: 0.1589079   
## Iteraccion: 22 Valor de X: 0.5538263 Error: 0.3949177   
## Iteraccion: 23 Valor de X: 0.5538266 Error: 0.1589085   
## Iteraccion: 24 Valor de X: 0.5538268 Error: 0.3949181   
## Iteraccion: 25 Valor de X: 0.5538269 Error: 0.1589087



## Iteracciones: 25 Resultado: 0.5538269

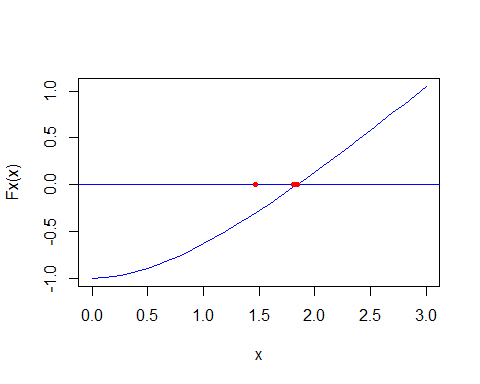
**Método de la Posición falsa**

Este metodo pretende juntar la seguridad del método de la bisección con la rapidez del método de la secante. Se parte de dos puntos tal que . Entonces primero se obtendra un con la siguiente formula:

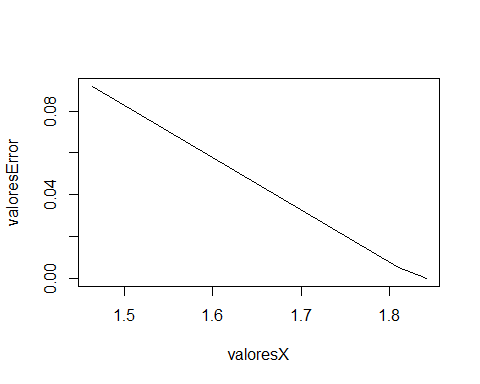
Luego este valor se reeplazara en la función y , como en el metodo de bisección, reemplazara a o a si el resultado de la función es menor o mayor a 0 respctivamente.

Seguún la siguiente definición se construyo el siguiente algoritmo:

Fx = function(x) exp(-x) + x -2  
 Fx1 = function(x) 1 - exp(x)  
   
 Pfalsa = function(a, b, error){  
 e = 1  
 x = seq(a, b, 0.1)  
 plot(x, Fx(x), type = "l", col="blue")  
 abline(h=0, col = "blue")  
 valoresX = c()  
 valoresError = c()  
 it = 0  
 while (e > error) {  
 it = it + 1  
 x = (Fx(b) \* a - Fx(a) \* b) / (Fx(b) - Fx(a))  
 if (Fx(x) == 0) {  
 break  
 }  
 if (Fx(x) \* Fx(a) < 0){  
 b = x  
 }  
 else {  
 a = x  
 }  
 e = abs(Fx(x) / Fx1(x))  
 points(rbind(c(x,0)),pch=19,cex=0.7,col="red")  
 cat("Valor de X: ", x, "\tError: ", e, "\n")  
 valoresX[it] = x  
 valoresError[it] = e  
 }  
 plot(valoresX, valoresError, type = "l")  
 }  
 Pfalsa(0, 3, 10e-8)



## Valor de X: 1.463567 Error: 0.09183747   
## Valor de X: 1.809481 Error: 0.005243437   
## Valor de X: 1.839096 Error: 0.000367304   
## Valor de X: 1.841241 Error: 2.618425e-05   
## Valor de X: 1.841394 Error: 1.868984e-06   
## Valor de X: 1.841405 Error: 1.334168e-07   
## Valor de X: 1.841406 Error: 9.52398e-09



**Método de la secante**

Este método parte de dos puntos y estima la tangent por una aproximación con la siguiente expresión:

Si no se llega al error requerido por la tolerancia se iterara de nuevo con el valor obtenido y el anterior a este como los dos puntos que se necesitan.

De acuerdo a lo anterior se construyo el siguiente algoritmo:

rm(list=ls())  
Fx = function(x) exp(x) - pi \* x  
Fx1 = function(x) exp(x) - pi  
  
Secante = function(x1, x2, error){  
 x = (Fx(x2) \* x1 - Fx(x1) \* x2) / (Fx(x2) - Fx(x1))  
 err = 1  
 contador = 0  
 while (err > error){  
 contador = contador + 1  
 x1 = x2  
 x2 = x  
 x = (Fx(x2) \* x1 - Fx(x1) \* x2) / (Fx(x2) - Fx(x1))  
 if (Fx(x) == 0)  
 break  
 err = abs(Fx(x) / Fx1(x))  
 cat("Valor X: ", x, "\t\tValor del Error: ", err, "\t\tIteraccion: ", contador, "\n")  
 }  
}  
  
Secante(0, 1, 10e-8)

## Valor X: 0.464349 Valor del Error: 0.08524539 Iteraccion: 1   
## Valor X: 0.5626182 Valor del Error: 0.008839985 Iteraccion: 2   
## Valor X: 0.5542804 Valor del Error: 0.0004534648 Iteraccion: 3   
## Valor X: 0.5538245 Valor del Error: 2.495487e-06 Iteraccion: 4   
## Valor X: 0.553827 Valor del Error: 7.02433e-10 Iteraccion: 5

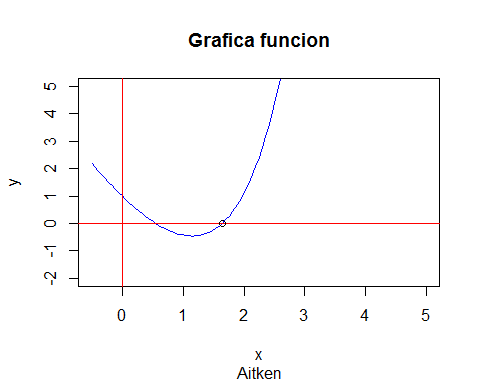
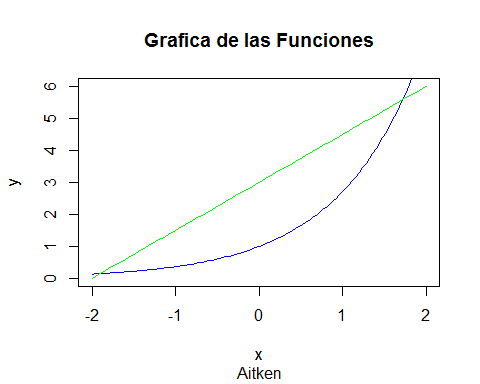
**Método de Aitken**

Es un método de aceleración de la convergencia que dada una sucesión , se calcula la nueva sucesión definida de la siguiente manera

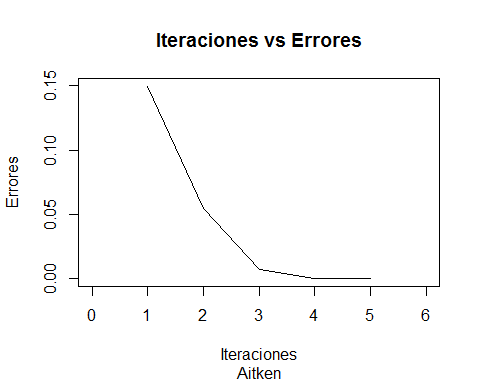
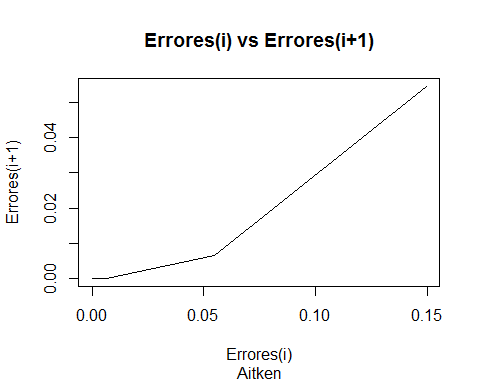
Y se puede escribir como:

En esencia este algoritmo trabajara con un punto pero para acelerar el proceso se escojeran los tres primeros puntos y de esta manera se llegara al resultado según la toleracia requerida en menos iteraciones. Con base a esta definición se realizó el siguiente algoritmo:

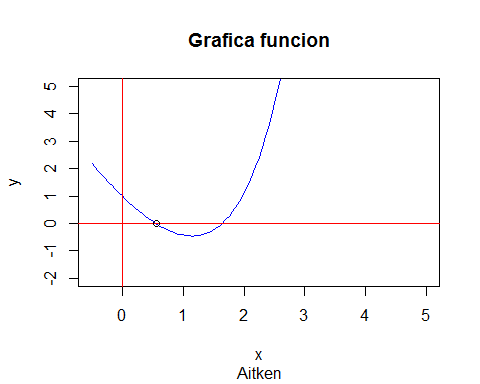
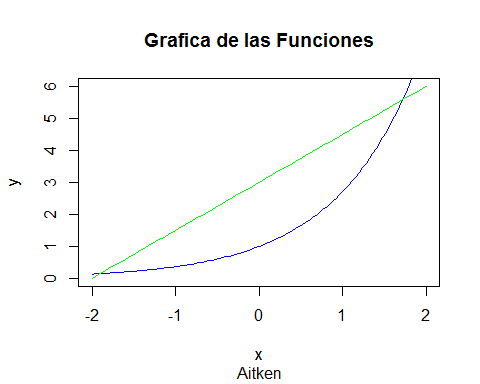
rm(list=ls())  
fx<-function(x)  
{  
 signif(exp(1), 5)^x  
}  
  
fx1<-function(x)  
{  
 signif(pi,5)\*x  
}  
  
fx2<-function(x)  
{  
 signif(exp(1), 5)^x-signif(pi,5)\*x  
}  
  
aitken = function(f, m, x0, tol)  
{  
   
 plot(fx, xlim = c(-2,2), ylim = c(0,6), col = "blue", main = "Grafica de las Funciones", sub = "Aitken", xlab = "x", ylab = "y")  
 par(new=TRUE)  
 curve(fx1, type = "l", col="green", axes=FALSE, ylab = "y")  
 par(new=FALSE)  
   
 iteraciones<-c()  
   
 Er1<-c()  
 Er2<-c()  
   
 k<-0  
 E1<-0  
   
 g<-parse(text=f)  
 fx = function(x){eval(g[[1]])}  
 d.<-D(parse(text=f ), "x")  
 df<-function(x) eval(d.)  
   
 plot(fx, xlim = c(-0.5,5), ylim = c(-2,5), col = "blue", main = "Grafica funcion", sub = "Aitken", xlab = "x", ylab = "y")  
 abline(h = 0, v=0, col= "red")  
   
 repeat  
 {  
   
 x1 = x0 - m\*(fx(x0)/df(x0))  
 dx = abs(x1-x0)  
 E2 = E1  
 E1 = dx/x1  
 cat("X=", x1, "\t", "E=", dx, "\t e=", E1,"\t Iteracion", k+1,"\n")  
   
 if(k >= 1)  
 {  
 Er1<-c(Er1, E2)  
 Er2<-c(Er2, E1)  
 }  
   
 k = k + 1  
   
 if (dx < tol) break;  
   
 x0 = x1  
   
   
 }  
   
 points(x1,0)  
   
 plot(fx, xlim = c(0,max(Er1)), ylim = c(0,max(Er2)), col = "white", main = "Errores(i) vs Errores(i+1)", sub = "Aitken", xlab = "Errores(i)", ylab = "Errores(i+1)")  
 lines(Er1, Er2, type = "l")  
   
 Er1<-c(Er1,Er2[k])  
 iteraciones<-c(1:k)  
 plot(fx, xlim = c(0,iteraciones[k]), ylim = c(0,Er1[1]), col = "white", main = "Iteraciones vs Errores", sub = "Aitken", xlab = "Iteraciones", ylab = "Errores")  
 lines(iteraciones, Er1, type = "l")  
}  
  
aitken("2.7182^x-3.1415\*x", 1, 2, 10^-8)



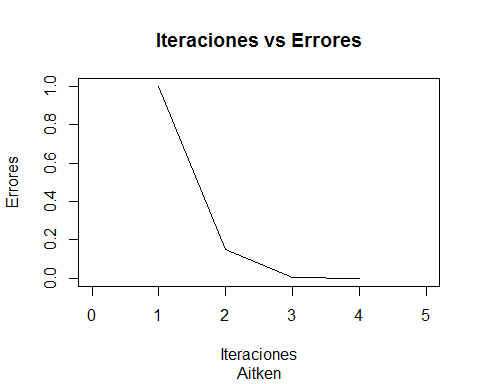
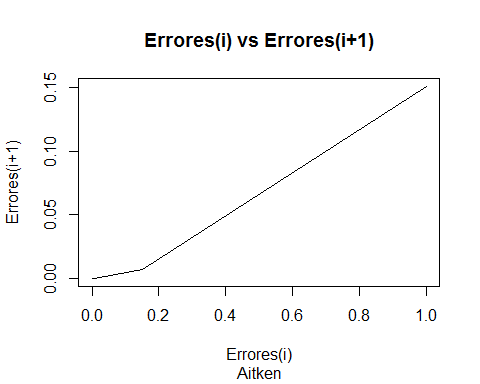
## X= 1.739666 E= 0.2603344 e= 0.1496462 Iteracion 1   
## X= 1.6496 E= 0.09006603 e= 0.05459873 Iteracion 2   
## X= 1.638732 E= 0.01086763 e= 0.00663173 Iteracion 3   
## X= 1.638579 E= 0.0001525962 e= 9.312712e-05 Iteracion 4   
## X= 1.638579 E= 2.987868e-08 e= 1.823451e-08 Iteracion 5   
## X= 1.638579 E= 1.776357e-15 e= 1.084084e-15 Iteracion 6



aitken("2.7182^x-3.1415\*x", 1, 0, 10^-8)

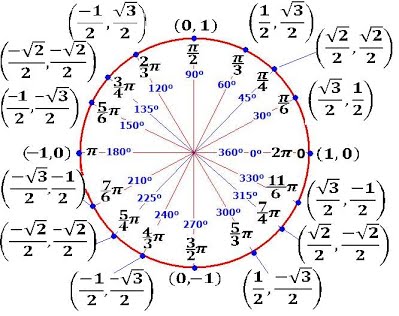


## X= 0.4669558 E= 0.4669558 e= 1 Iteracion 1   
## X= 0.5498345 E= 0.08287861 e= 0.1507338 Iteracion 2   
## X= 0.5538331 E= 0.003998596 e= 0.007219858 Iteracion 3   
## X= 0.553843 E= 9.896336e-06 e= 1.786849e-05 Iteracion 4   
## X= 0.553843 E= 6.078171e-11 e= 1.097454e-10 Iteracion 5



**Metodo de Graficar Polares**

Con el siguiente algoritmo, dadas funciones trigonometricas se graficaran reeplazando por valores de 0 a 2π.



Gracias a funciones de R se pudo construir el siguiente algoritmo:

rm(list=ls())  
  
dimension = seq(-pi, pi, pi/300)  
r = 1 - 2\*cos(dimension)  
r2 = 4-3\*sin(dimension)  
  
interseccion = function(r1, r2, theta){  
   
}  
  
Polar = function(theta, r, color){  
 x = 0  
 y = 0  
 axisX = 1  
   
 for (i in 1:length(r)) {  
 if (is.nan(r[i]) == TRUE){  
 r[i] = 0  
 }  
 }  
   
 angulo = seq(-max(theta), max(theta), theta[2] - theta[1])  
 x = r \* cos(theta)  
 y = r \* sin(theta)  
 plot(c(-max(r), max(r)), c(-max(r), max(r)), xlab = "Radio", ylab = "Radio")  
   
 aux = max(r)  
  
   
 while (aux > 0) {  
 f\_i = aux \* sin(angulo)  
 circun = aux \* cos(angulo)  
 #Graficar la circunferencia  
 points(circun, f\_i, pch="-", col="blue", cex = 0.3)  
 text(axisX + 0.2, -0.2, axisX, col="blue")  
 axisX = axisX + 1  
 aux = aux - 1  
  
 }  
 abline(v=((max(circun) + min(circun)) / 2), col="blue")  
 abline(h=((max(circun) + min(circun)) / 2), col="blue")  
 #Graficar las rectas que atraviesan  
 segments(-max(r) + 0.5, -max(r) + 0.5, max(r) - 0.5, max(r) - 0.5, col="blue")  
 segments(-max(r) + 0.5, max(r) - 0.5, max(r) - 0.5, max(r) + 0.5, col="blue")  
 points(x, y, pch = 20, col = color, cex = 1)  
}  
  
Polar(dimension, r, "green")  
par(new=T)  
Polar(dimension, r2, "orange")

