Método de la secante

Sebastian Angarita, Hector Rodriguez, Aldemar Ramirez

7/8/2019

Problema

Hallar la raíz de una función a partir de una pareja x_0 y x_1 cercanos a x^* a traves del metodo de Secante.

Solución

Lenguaje de programación: R

Función principal método de la secante

Parametros:

f <- función.

x0 <- x_0 en el rango [a,b] donde se busca la raiz.

x1 <- x_1 en el rango [a,b] donde se busca la raiz.

tol <- tolerancia mínima que debe tener la función.

maxiter <- cantidad máxima de iteraciones.

Valores de retorno:

x2 <- resultado de la raíz.

errorAbsoluto <- vector de errores absolutos de las x_n.

errorRelativo <- vector de errores relativos de las x_n.

x <- vector de las x_n.

Implementación

```
secante = function(f, x0, x1, tol, maxiter = 100){
  errorAbsoluto = c()
  errorRelativo = c()
  x = c()
  f0 = f(x0)
  f1 = f(x1)
  k = 0
  while (abs(x1 - x0) > tol && k <= maxiter ) {
    k = k+1
    pendiente = (f1 - f0)/(x1 - x0)
    if (pendiente == 0) return( cero = NA, f.cero = NA, iter = k, ErrorEst = NA)
    x2 = x1 - f1/pendiente
    f2 = f(x2)
    x0 = x1; f0 = f1
```

```
    x1 = x2; f1 = f2

    x <- c(x,x1)
    errorAbsoluto <- c(errorAbsoluto,abs(x1-x0))
    errorRelativo <- c(errorRelativo,abs(x1-x0)/(abs(x2)))

  }
  if (k > maxiter) {
    warning("No se alcanzó el número de iteraciones")
  }
  else {

  my_list <- list("resultado" = x2, "errorAbsoluto" = errorAbsoluto,
                  "errorRelativo" = errorRelativo, "x" = x)
  return(my_list)
  #return(list(cero=x2, f.cero=f2, iter=k, ErrorEst =abs(x2-x1)))
  }

}
```

La función toma dos aproximaciones iniciales x_0 y x_1, en el paso k+1, x_{k+1} se calcula, usando x_k y x_{k-1}, como la interseccion con el eje X de la recta (secante) que pasa por los puntos $(x_{k-1}, f(x_{k-1}))$ y $(x_k, f(x_k))$.

Entonces, si $f(x_k) - f(x_{k-1})! = 0$.

$$x_{k-1} = x_k * (x_k - x_{k-1})/f(x_k) - f(x_{k-1})$$

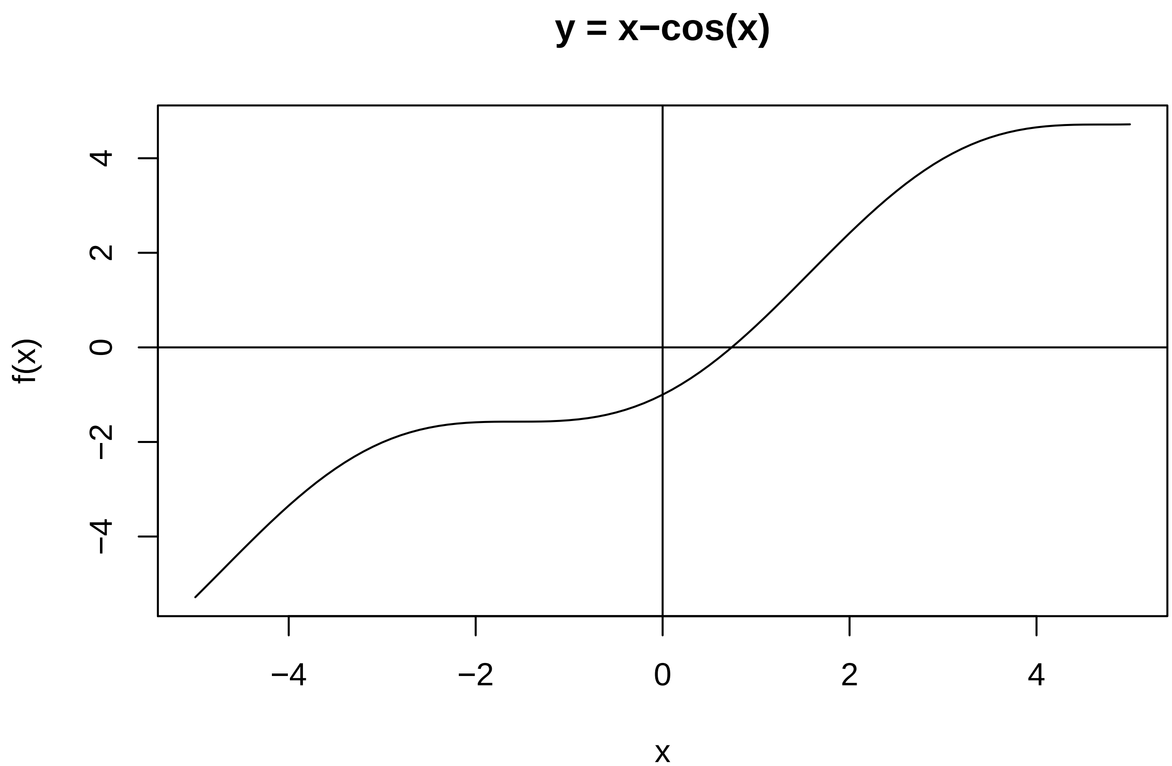
Resultados

Para mostrar el funcionamiento del método se tomara la ecuación:

$$x - cos(x) = 0$$

Gráfica de la función

```
f = function(x) x-cos(x)
curve(f, -5,5); abline(h=0, v=0) #gráfico para decidir un intervalo
title(main="y = x-cos(x)")
```



Como se ve en la gráfica la función tiene una raíz en el intervalo [0,2]. Teniendo en cuenta se tomará $x_0 = 0$ y $x_1 = 2$.

Tabla de resultados

Se muestra una tabla con la cantidad de iteraciones y el resultado de cada una para la ecuación $0 = x - cos(x)$.

```
##--- Pruebas
f = function(x) x-cos(x)
resultados<-secante(f, 0, 2, 1e-15, 10)


tablaErrores <- data.frame(
  "iteraciones" = 1:length(resultados$errorAbsoluto),
  "x_n" = resultados$x,
  "errorAbsoluto" = resultados$errorAbsoluto,
  "errorRelactivo" = resultados$errorRelativo
)
print(tablaErrores)
```

```
##   iteraciones       x_n errorAbsoluto errorRelactivo
## 1           1 0.5854549  1.414545e+00   2.416147e+00
## 2           2 0.7171349  1.316799e-01   1.836195e-01
## 3           3 0.7399008  2.276590e-02   3.076885e-02
## 4           4 0.7390811  8.196294e-04   1.108984e-03
```

```
## 5           5 0.7390851  3.996441e-06   5.407282e-06
## 6           6 0.7390851  7.195670e-10   9.735914e-10
## 7           7 0.7390851  7.771561e-16   1.051511e-15
```

```r
cat("Numero de iteraciones",length(resultados$errorAbsoluto),"\n")
```

```
## Numero de iteraciones 7
```

```r
cat("x = ",resultados$resultado,"\n")
```

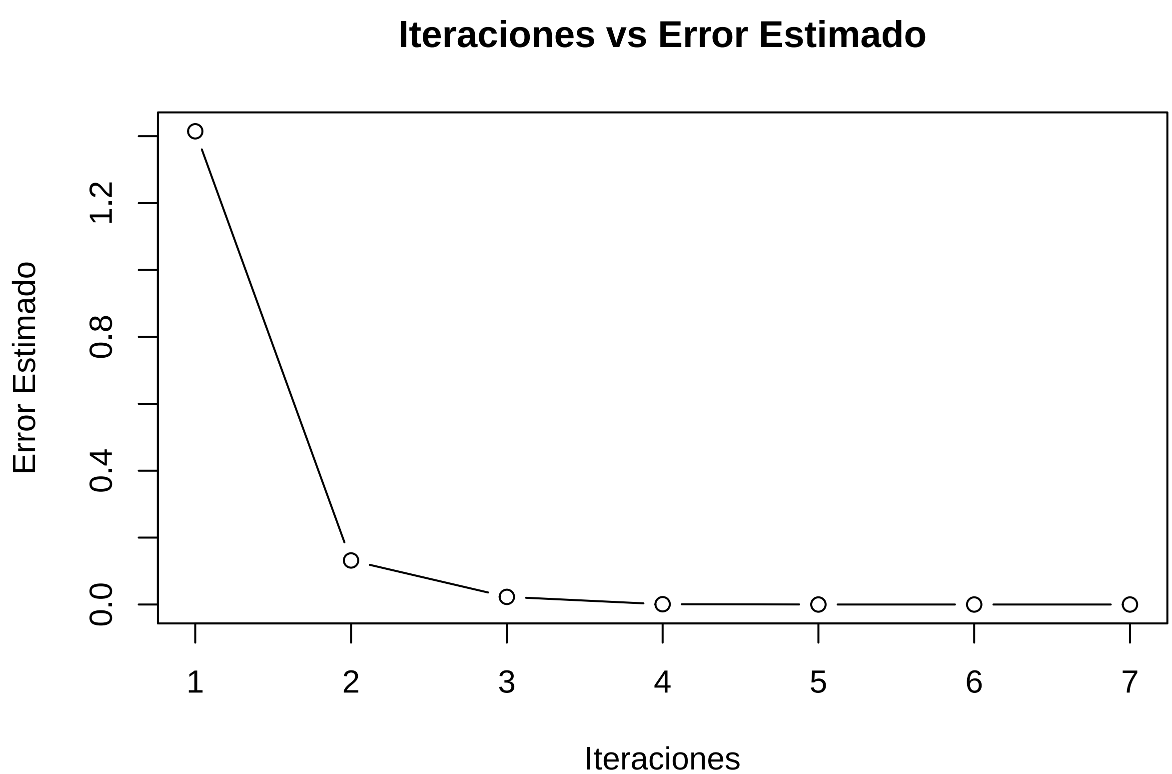
```
## x =  0.7390851
```

```r
cat("Error estimado = ", resultados$errorAbsoluto[length(resultados$errorAbsoluto)],"\n")
```

```
## Error estimado =  7.771561e-16
```

Grafica de iteraciones vs error estimado

```r
plot(x = 1:length(resultados$x), y = resultados$errorAbsoluto,
     xlab = "Iteraciones", ylab = "Error Estimado", type="b",
     main = "Iteraciones vs Error Estimado")
```

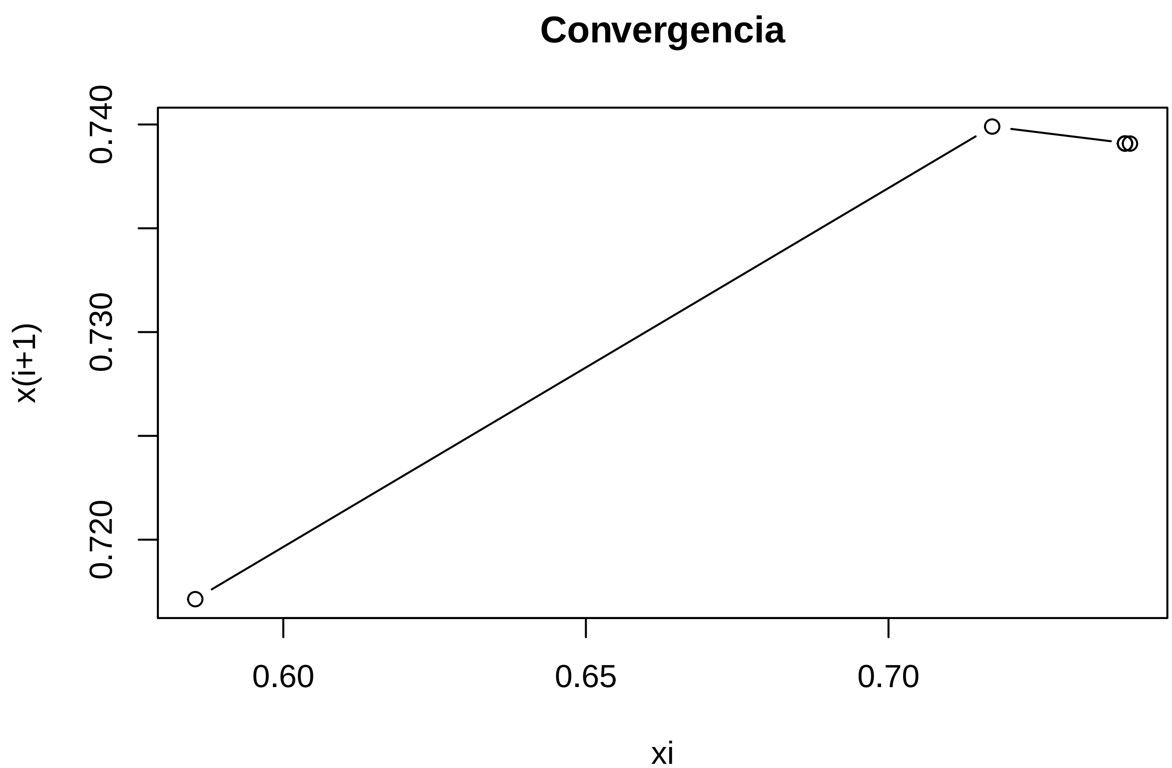


Entre mas iteraciones se realicen aumenta la convergencia del error estimado a cero.

$ErrorEstimadoAbsoluto : x1 - x0$

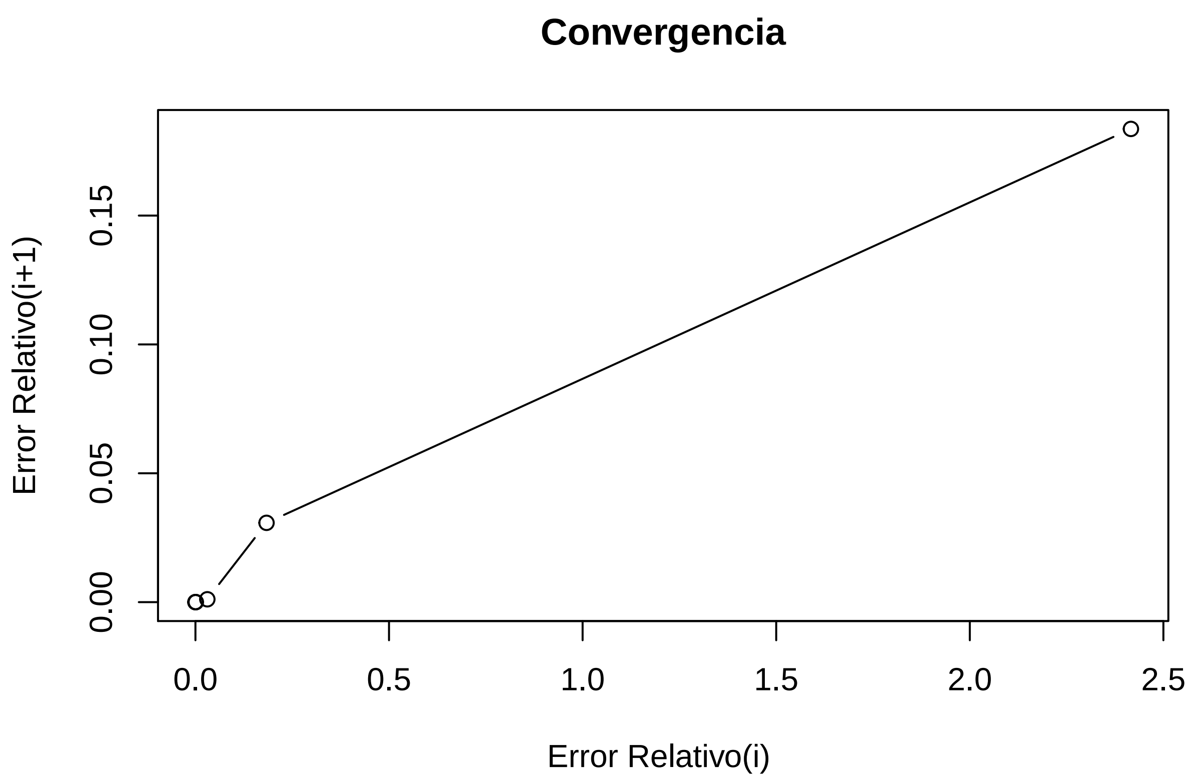
Grafica de x(i) vs x(i+1)

```r
m_i =  resultados$x[-length(resultados$x)]
m_i2 = resultados$x
m_i2 = m_i2[-1]

plot(x =m_i, y =m_i2, xlab = "xi", ylab = "x(i+1)", type="b",main = "Convergencia")
```



Grafica de Error Relativo(i) vs Error Relativo(i+1)

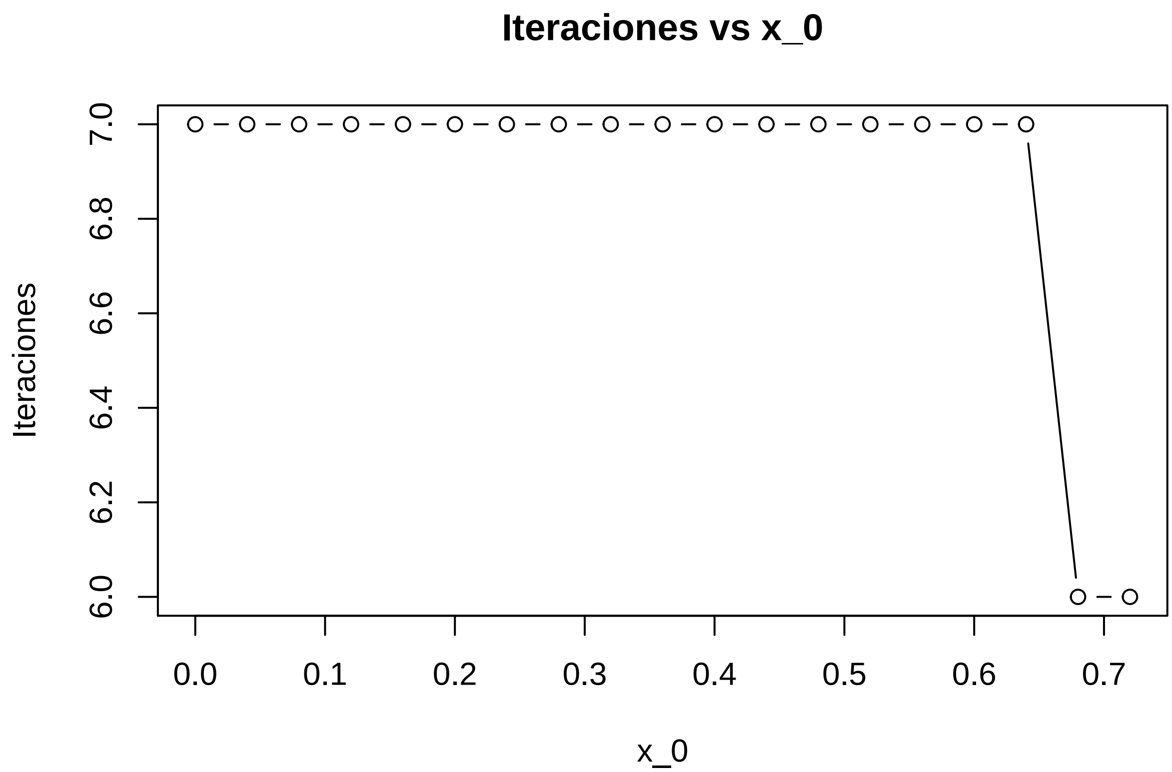
```r
m_i =  resultados$errorRelativo[-length(resultados$errorRelativo)]
m_i2 = resultados$errorRelativo
m_i2 = m_i2[-1]

plot(x =m_i, y =m_i2, xlab = "Error Relativo(i) ",
     ylab = "Error Relativo(i+1)", type="b",main = "Convergencia")
```



De acuerdo con las gráficas el método tiene una convergencia lineal.

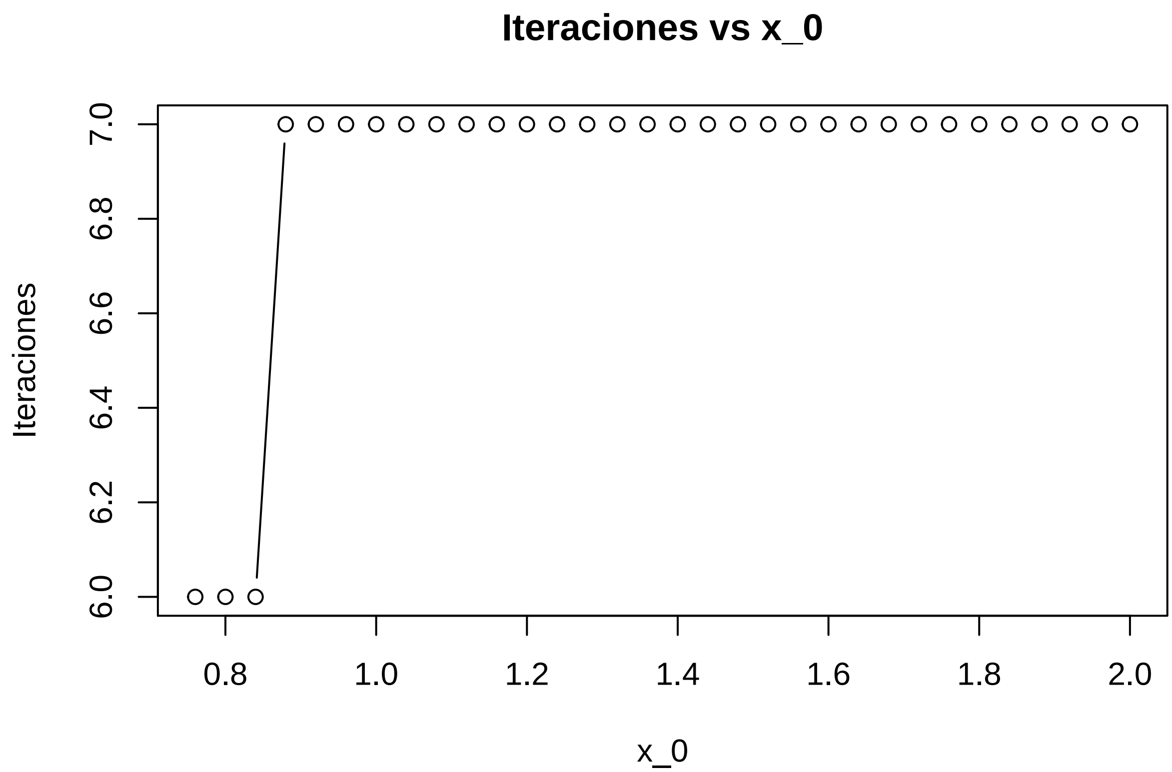
Grafica de iteraciones vs x0 (Aproximacion por la izquierda)

```
l=0
v = c()
iteraciones = c()
while(l<0.74) {
  v <- c(v,l)
  resultados<-secante(f, l, 2, 1e-15, 10)

  tablaErrores <- data.frame(
    "iteraciones" = 1:length(resultados$errorAbsoluto),
    "x_n" = resultados$x,
    "errorAbsoluto" = resultados$errorAbsoluto,
    "errorRelactivo" = resultados$errorRelativo
  )
  iteraciones <- c(iteraciones,length(resultados$errorAbsoluto))
  l = l+0.04
  #print(tablaErrores)
}
plot(x =v, y =iteraciones, xlab = "x_0", ylab = "Iteraciones",
     type="b",main = "Iteraciones vs x_0")
```



Grafica de iteraciones vs x0 (Aproximacion por la derecha)

```
l=2
v = c()
iteraciones = c()
while(l>0.74) {
  v <- c(v,l)
  resultados<-secante(f,0,l, 1e-15, 10)

  tablaErrores <- data.frame(
    "iteraciones" = 1:length(resultados$errorAbsoluto),
    "x_n" = resultados$x,
    "errorAbsoluto" = resultados$errorAbsoluto,
    "errorRelactivo" = resultados$errorRelativo
  )
  iteraciones <- c(iteraciones,length(resultados$errorAbsoluto))
  l = l-0.04
  #print(tablaErrores)
}
plot(x =v, y =iteraciones, xlab = "x_0", ylab = "Iteraciones",
     type="b",main = "Iteraciones vs x_0")
```



Mientras mas alejado se encuentre x_0 del x^* son necesarias mas cantidad de iteraciones, aunque se puede observan que es una cantidad constante hasta cierto x_0(alejando x_0 una unidad de la raíz solo se gana una iteración de más).

Diferencias y mejoras al método de Newton Raphson

El método no llega a aser tan rapido como el método de Newton Raphson pero por la ventaja que tiene sobre este es que no pide a la computadora calcular la derivada de la función lo cual puede llegar a ser un procedimiento complejo para la máquina.