Cálculo Numérico - achar zeros de uma função

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.3

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.3

## Warning: package 'tibble' was built under R version 4.1.2

## Warning: package 'tidyr' was built under R version 4.1.2

## Warning: package 'readr' was built under R version 4.1.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tinytex)
```

```
## Warning: package 'tinytex' was built under R version 4.1.3
```

Primeiro passo - Isolamento

- existe uma raiz onde os intervalos mudam de sinal

exemplo de isolamento de intervalo

```
fx <- function(x) {
  y <- x^3 - 9*x +5
  return(y)
}
x=seq(from=0,to=3,by=0.5)
tabela<-cbind(x,fx(x),sign(fx(x)))

print(tabela)
```

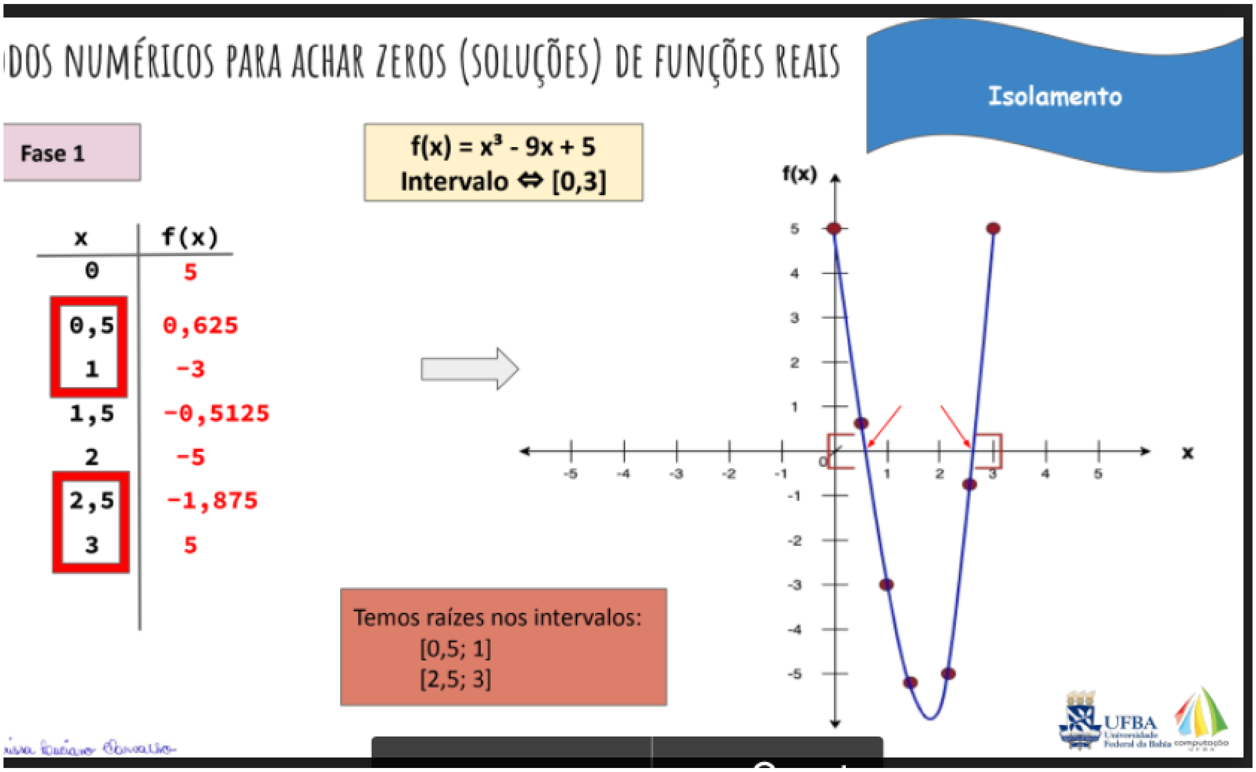

Figure 1: isolamento

```
##        x
## [1,] 0.0  5.000  1
## [2,] 0.5  0.625  1
## [3,] 1.0 -3.000 -1
## [4,] 1.5 -5.125 -1
## [5,] 2.0 -5.000 -1
## [6,] 2.5 -1.875 -1
## [7,] 3.0  5.000  1
```

```
muda_sinal_anterior<-c(FALSE)
for (i in 2:nrow(tabela)) {
muda_sinal_anterior[i]  <-!(sign(tabela[i-1,2]) == sign(tabela[i,2]))
}
tabela2<- cbind(tabela, "mudou o sinal?" = muda_sinal_anterior)
print(tabela2)
```

```
##        x            mudou o sinal?
## [1,] 0.0  5.000  1              0
## [2,] 0.5  0.625  1              0
## [3,] 1.0 -3.000 -1              1
## [4,] 1.5 -5.125 -1              0
## [5,] 2.0 -5.000 -1              0
## [6,] 2.5 -1.875 -1              0
## [7,] 3.0  5.000  1              1
```

```r
intervalos<-list(0)
for (i in 2:nrow(tabela2)) {
  if (tabela2[i,4]==1){
  intervalos[[i-1]] <- cbind(tabela2[i-1,1],tabela2[i,1])

  }
  else{
    intervalos[[i-1]] <- NA

  }
}

intervalos<-intervalos[!is.na(intervalos)]
print(intervalos)
```

```
## [[1]]
##   [,1] [,2]
## x  0.5    1
##
## [[2]]
##   [,1] [,2]
## x  2.5    3
```

Método de Bissecção

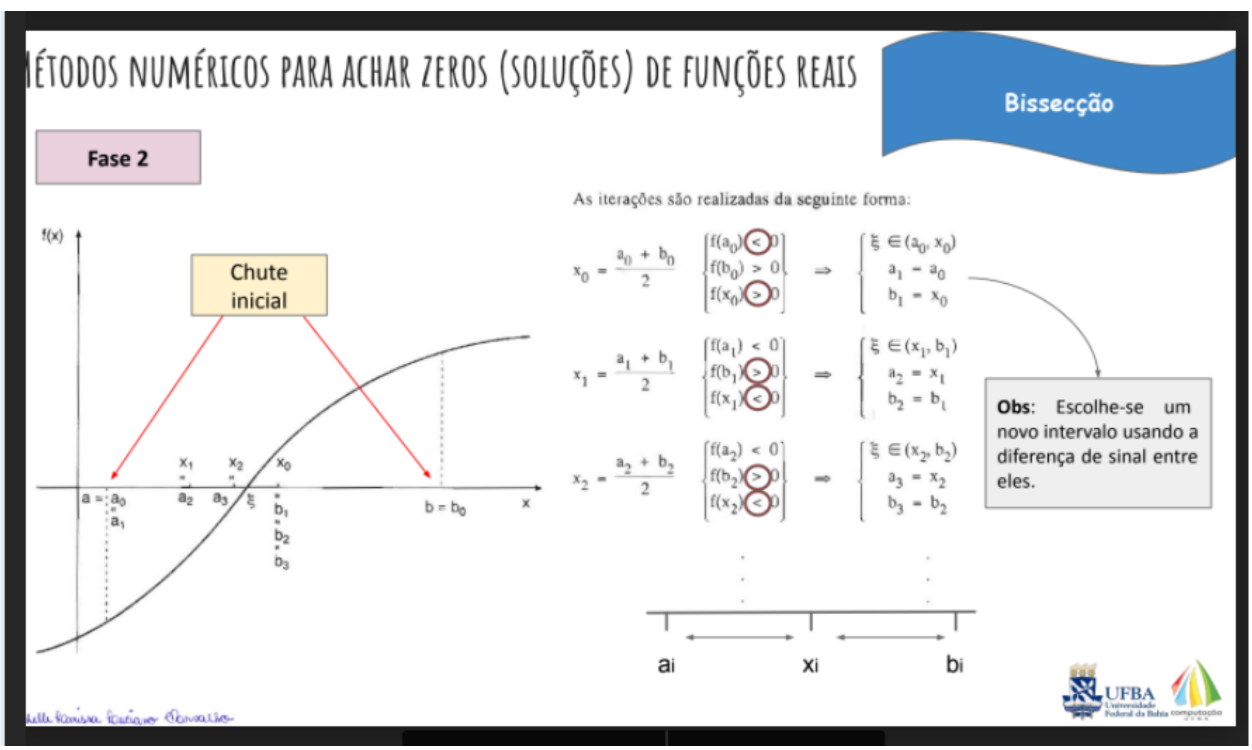

Figure 2: metodo1

```
m1<-function(a,b){
  x0<- (a+b)/2
  return(x0)
}
```

Teorema de Bolzano e seu corolário

- A função corta o eixo somente uma vez.
- Dado um intervalo [a,b], vamos calcular a raiz entre a e b usando m1
- pega o ponto médio pra encontrar a raiz inicial x0
- saber qual o critério de parada
- se o x0 não está próximo o suficiente, atualiza a raiz e continua dividindo
- dividir o intervalo ao meio e verificar se f(a) e f(b) são opostos pra atualizar a raiz

Exemplo de Bisseção

Começamos com o primeiro intervalo definido em intervalos

```
a<- intervalos[[1]][[1]]
b<- intervalos[[1]][[2]]
```

Usaremos a função fx como teste, m1 como a raiz aproximada e um erro de 0,01

```
erro<- c(0.01)
fx
```

```
## function(x) {
##   y <- x^3 - 9*x +5
##   return(y)
## }
## <bytecode: 0x000000001259eb18>
```

```
m1
```

```
## function(a,b){
##   x0<- (a+b)/2
##   return(x0)
## }
```

Faremoz uma tabela:

```
x_barra<- m1(a,b)
tabela3<- tibble(a=a,b= b, "raiz aproximada"=m1(a,b),"função f(x_barra)" = fx(x_barra))
print(tabela3)
```

```
## # A tibble: 1 x 4
##       a     b `raiz aproximada` `função f(x_barra)`
##   <dbl> <dbl>             <dbl>                <dbl>
## 1   0.5     1              0.75                -1.33
```

Então se faz os seguintes questionamentos:

- f(x_barra) é suficientemente próximo da raiz real?
- é menor que a precião em módulo? se sim, para

Se não:

- verificar sinal de f(x_barra), se for negativo, x_barra substitui b, se for positivo, substitui a.

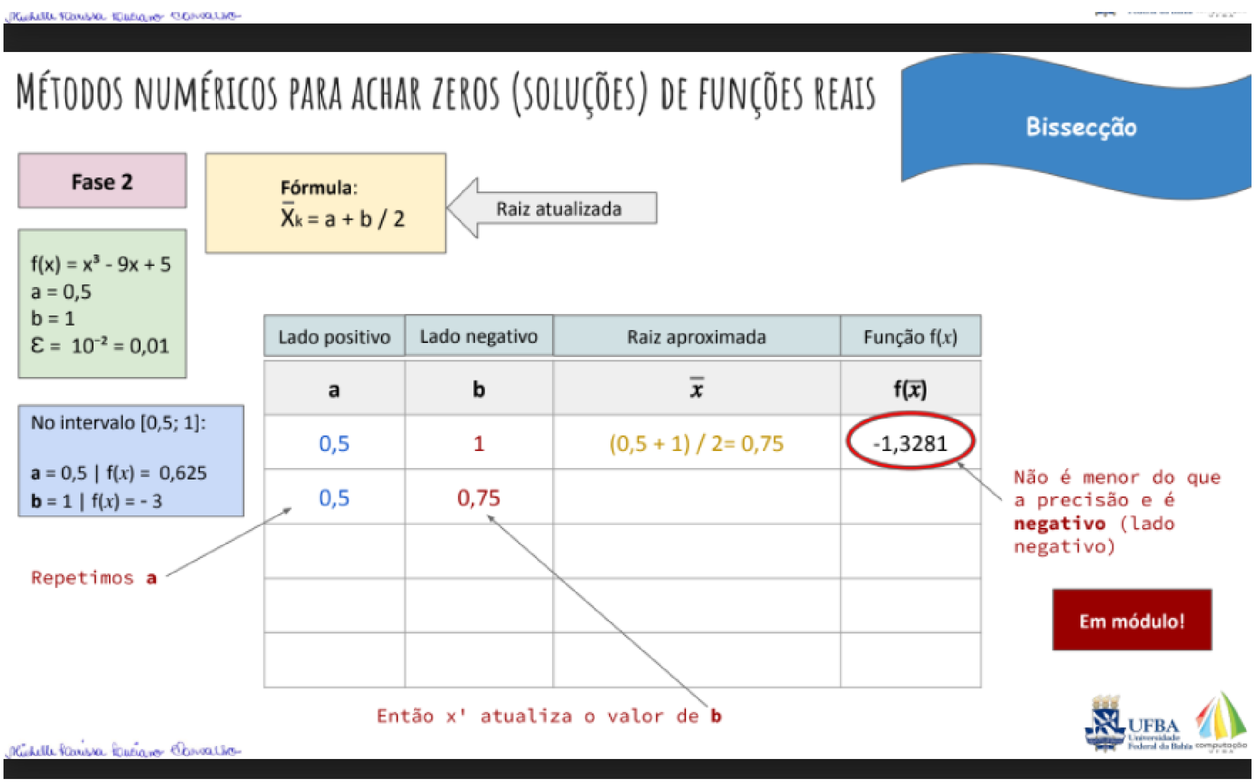

Figure 3: metodo11

```
if (abs(fx(x_barra)) >  erro |sign((fx(x_barra))) == -1){
  b<-x_barra
}else if (abs(fx(x_barra)) >  erro |sign((fx(x_barra))) == 1){
  a<-x_barra
}
```

Agora precisa calcular novamente x_barra e f(x_barra)

```
x_barra<-m1(a,b)
f_x_barra <- fx(x_barra)
```

e repete o chunk anterior

```
if (abs(fx(x_barra)) >  erro & sign((fx(x_barra))) == -1){
  b<-x_barra
} else if (abs(fx(x_barra)) >  erro & sign((fx(x_barra))) == 1){
  a<-x_barra
}
```

e repete

```
x_barra<-m1(a,b)
f_x_barra <- fx(x_barra)
```

e repete

```
if (abs(fx(x_barra)) >  erro & sign((fx(x_barra))) == -1){
  b<-x_barra
} else if (abs(fx(x_barra)) >  erro & sign((fx(x_barra))) == 1){
  a<-x_barra
}
```

e repete

```
x_barra<-m1(a,b)
f_x_barra <- fx(x_barra)
```

e repete

```
if (abs(fx(x_barra)) >  erro & sign((fx(x_barra))) == -1){
  b<-x_barra
} else if (abs(fx(x_barra)) >  erro & sign((fx(x_barra))) == 1){
  a<-x_barra
}
```

e repete

```
x_barra<-m1(a,b)
f_x_barra <- fx(x_barra)
```

FINALMENTE, encontramos um f(x__barra) menor que o erro, então temos uma das raízes

```
f_x_barra> erro
```

```
## [1] FALSE
```

```
print(x_barra)
```

```
## [1] 0.578125
```

Agora falta o outro intervalo (PUTA QUE PARIU)

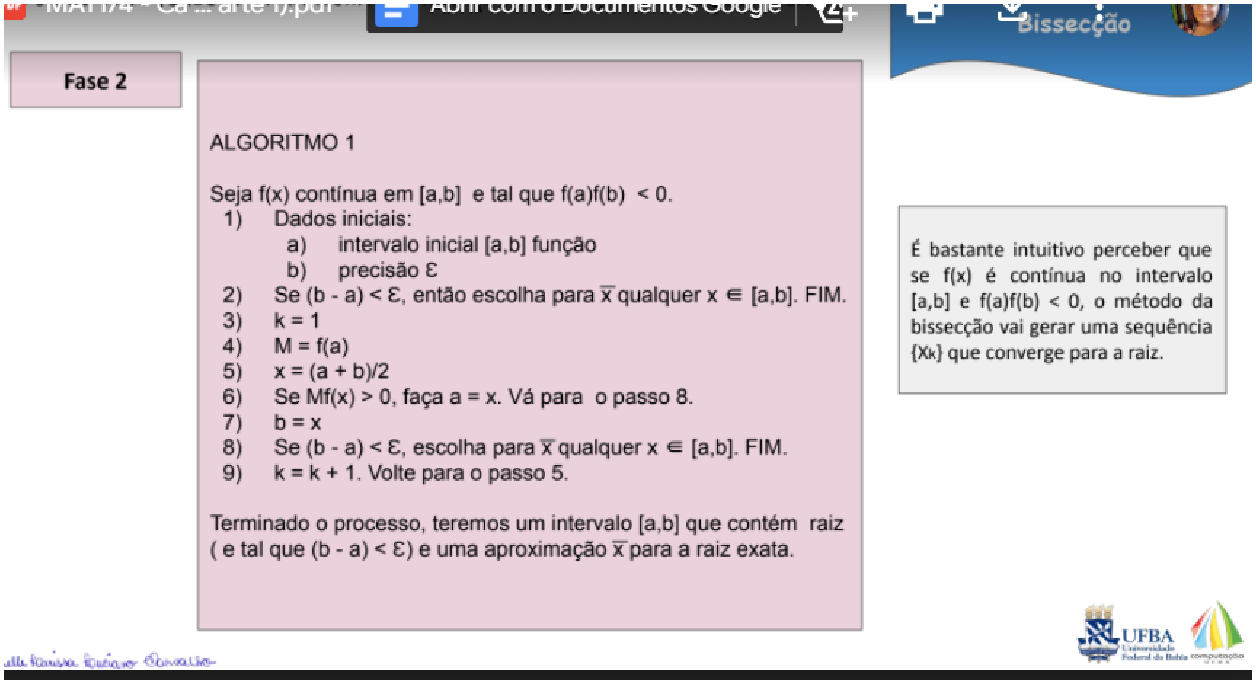

Figure 4: metodo12

Resumo do Método da Bissecção

```
a<- 0.5
b<- 1
x_barra<-m1(a,b)
f_x_barra <- fx(x_barra)

while (abs(f_x_barra)> erro) {
 if (sign(f_x_barra) == -1){
  b<-x_barra
      x_barra<-m1(a,b)
      f_x_barra <- fx(x_barra)
      print(cbind(x_barra,f_x_barra))
 } else if (sign(f_x_barra) == 1){
  a<-x_barra
  x_barra<-m1(a,b)
      f_x_barra <- fx(x_barra)
      print(cbind(x_barra,f_x_barra))

 }

      }
```

```
##      x_barra  f_x_barra
## [1,]   0.625 -0.3808594
##      x_barra f_x_barra
## [1,]  0.5625 0.1154785
##      x_barra  f_x_barra
```

```
## [1,] 0.59375 -0.1344299
##       x_barra    f_x_barra
## [1,] 0.578125 -0.009899139
```

```r
print(paste0("resultado final é:",x_barra))
```

```
## [1] "resultado final é:0.578125"
```

Metódo da Posição Falsa

O método da posição falsa só difere no método de obter o x_barra.

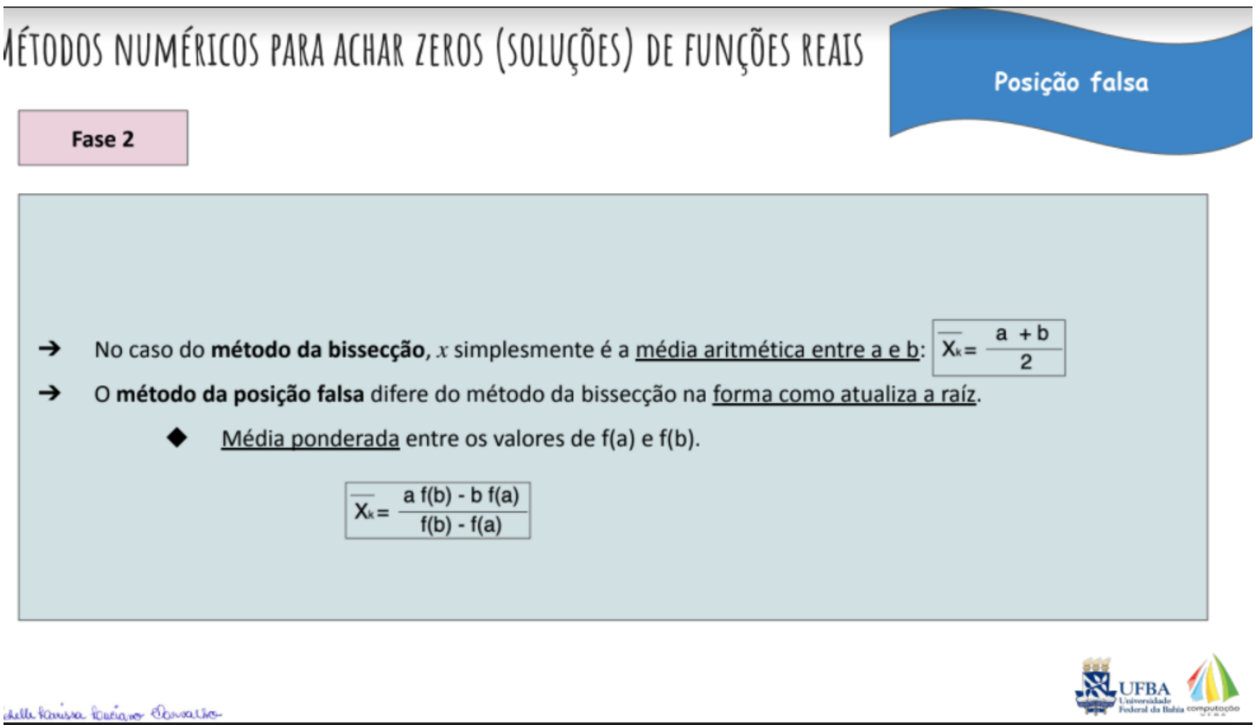

Figure 5: metodo12

```r
m2<- function(a,b){
  x0<- (a*fx(a) -b*fx(b))/(fx(a)-fx(b))
  return(x0)
}
```

O critério de parada passa a ser a diferença entre b e a menor que o erro ou f(a ou b ou x) menor que erro.

```r
c1<-abs(b-a)>erro
c2<-((abs(fx(a)) | abs(fx(b))|abs(fx(x_barra)))>erro)
condition<- (c1|c2)
```

Exemplo utilizando fx

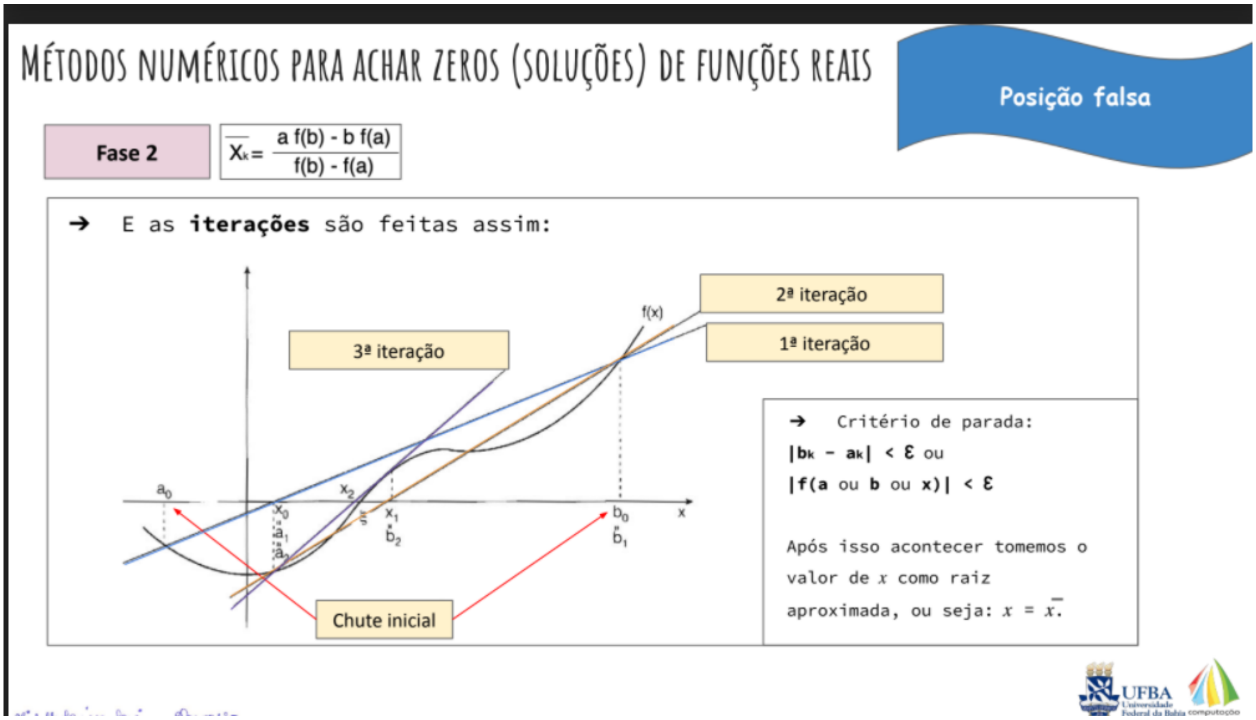

Figure 6: metodo22

```r
a<- 0.5
b<- 1
x_barra<-m2(a,b)
f_x_barra <- fx(x_barra)
#k<-0
while (abs(b-a) & abs(fx(a)) & abs(fx(b)) & abs(fx(x_barra)) > erro) {
 if (sign(f_x_barra) == -1){
  b<-x_barra
      x_barra<-m2(a,b)
      f_x_barra <- fx(x_barra)
      print(cbind(x_barra,f_x_barra))
      } else if (sign(f_x_barra) == 1){
  a<-x_barra
  x_barra<-m2(a,b)
      f_x_barra <- fx(x_barra)
      print(cbind(x_barra,f_x_barra))

 }

      }
```

```
##        x_barra f_x_barra
## [1,] 0.8299914 -1.898154
##        x_barra f_x_barra
## [1,] 0.7482506 -1.315326
##        x_barra  f_x_barra
## [1,] 0.6682864 -0.7161164
##        x_barra  f_x_barra
```

```
## [1,] 0.5898599 -0.1035067
##         x_barra f_x_barra
## [1,] 0.5127674 0.5199158
##        x_barra f_x_barra
## [1,] 0.525567 0.4150692
##         x_barra f_x_barra
## [1,] 0.5383998 0.3104703
##         x_barra f_x_barra
## [1,] 0.5512664 0.2061296
##         x_barra f_x_barra
## [1,] 0.5641676 0.1020576
##         x_barra    f_x_barra
## [1,] 0.5771043 -0.001734783
```

```
print(paste0("resultado final é:",x_barra))
```

```
## [1] "resultado final é:0.577104338590049"
```

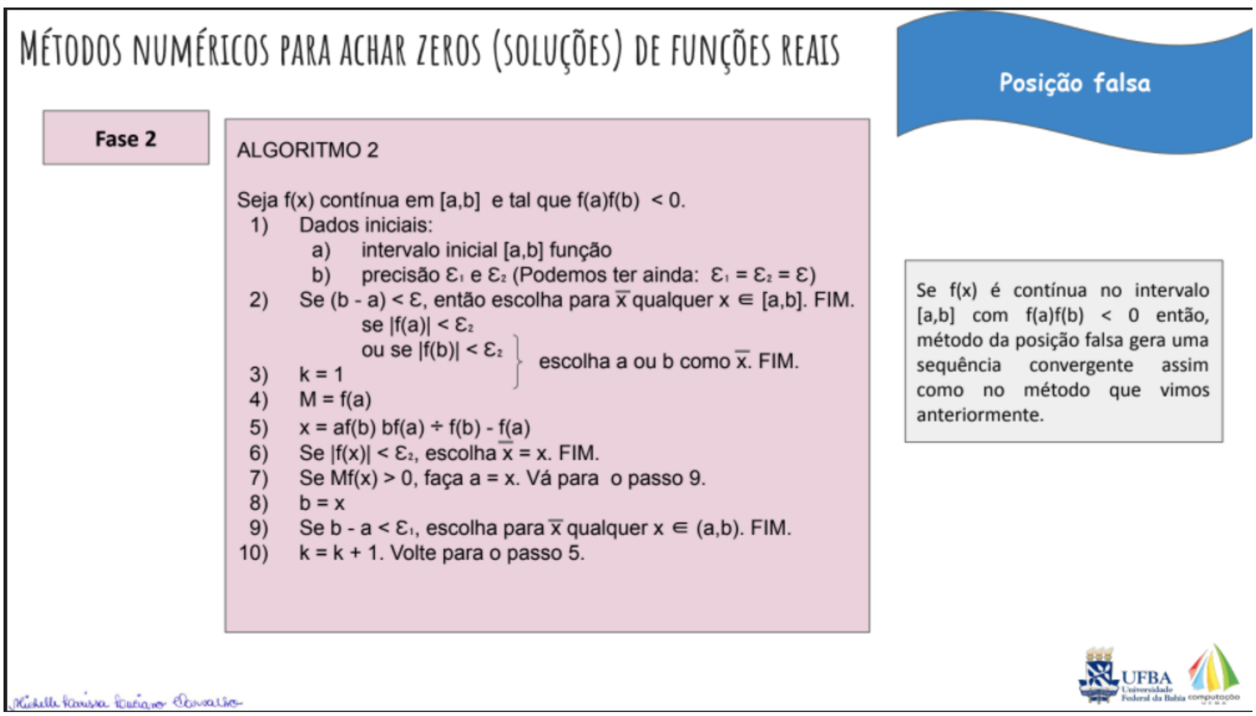

Figure 7: metodo23