

# PM 520 - Lecture 3

Finding roots and fixed points  
Chapter 10 of Jones et al.

# Assignment1: Randomization tests - golf balls - due today

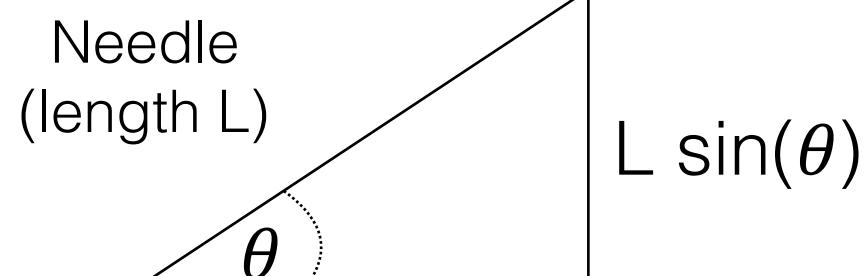
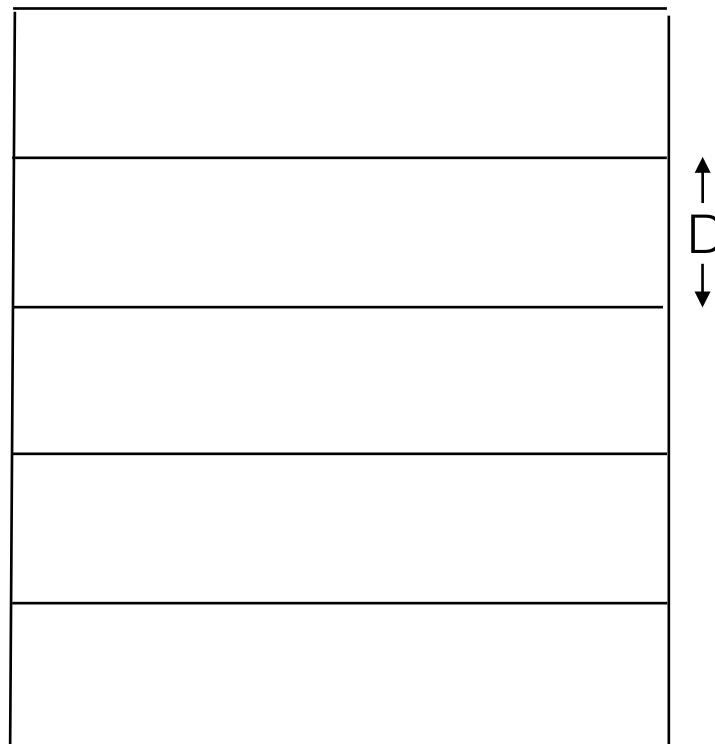
- Allan Rossman used to live along a golf course and collected the golf balls that landed in his yard. Most of these golf balls had a number on them.



- Question: What is the distribution of these numbers?
- In particular, are the numbers 1, 2, 3, and 4 equally likely?

[Originally due to Allan Rossman - via Randall Pruim]

# Monte Carlo Simulation: Calculating $\pi$ - Buffon's needle



$$\text{So, } P(\text{cross line}) = \min(1, LSin(\theta)/D)$$

If  $N$  = number of trials, lines are distance **D=1** apart,  
 $C$  = number of times needle crosses line,  
and  **$L \leq D$** , then:

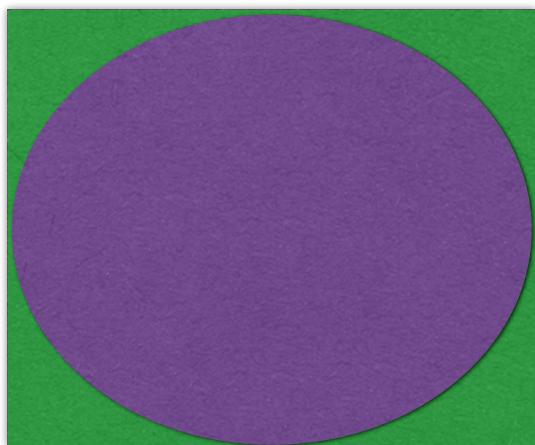
$$C/N \sim \int L\sin(\theta)(1/\pi) d\pi \quad [\text{integral goes from } 0 \text{ to } \pi \text{ in radians}]$$

$$\text{So } C/N \sim L[-\cos(\pi)-(-\cos(0))]/\pi = 2L/\pi$$
$$\text{So } \pi \sim 2LN/C$$

Table

# Monte Carlo Simulation: Calculating $\pi$ - ‘dart board’ approach

- Estimate (not calculate)  $\pi$  using Monte Carlo methods.
- So, you need to think of something you can simulate in which the probability of success depends upon  $\pi$ .



# In-class exercise: Exponential task 1

- Generate 1000  $\text{Exp}(\lambda)$  rvs. conditional on them each being greater than  $y$ , for some  $y$  (Try  $\lambda=1$ ,  $y=1$ , say). Let's call those r.v.s  $X$ .
- Plot a histogram showing the distribution of  $(x-y)$ , for  $y=1$  and  $\lambda=1$ , and compare it to 1000  $\text{Exp}(1)$  rvs. [or superimpose the exponential density function using the command `curve(lambda * exp(-lambda * x))`]
- How do we generate exponential rvs conditional on them being greater than  $y$ ?

# In-class exercise: Exponential task 2: Waiting for a bus

- Suppose times between bus arrivals are distributed as  $T \sim \text{exp}(1)$ .
1. Suppose we arrive at a bus-stop at some fixed time during the day (say after 10 hours). How long, on average, do we have to wait for a bus? [What if we arrive at a random time each day?]
  2. If we get off one bus and wait for the next one to arrive on the same route, how long, on average, do we have to wait?
  3. How long on average was the time between the arrival of the bus we caught and the one before it.
  4. What is the expected time between any two buses?

Note: the mean of an  $\text{exp}(\lambda)$  r.v. is  $1/\lambda$ .

See 'Week2-BusWaitingTimesExercise on Github

# The dinner party

Suppose you are at a dinner party. The host wants to give out a door prize that is wrapped in a box. Everyone (including the host) sits around a circular table and each person is given a fair coin. Initially the host is holding the box. He/she flips his coin. If it is heads, the box is passed to the right; if it is tails, it is passed to the left. The process is repeated by whichever guest is holding the box. (Heads, they pass right; tails, they pass left.) The game ends when the last person to receive the box finally gets it for the first time. That person gets to keep the box as the winner of the game.

How do you work out where should you sit in order to maximize your probability of winning?

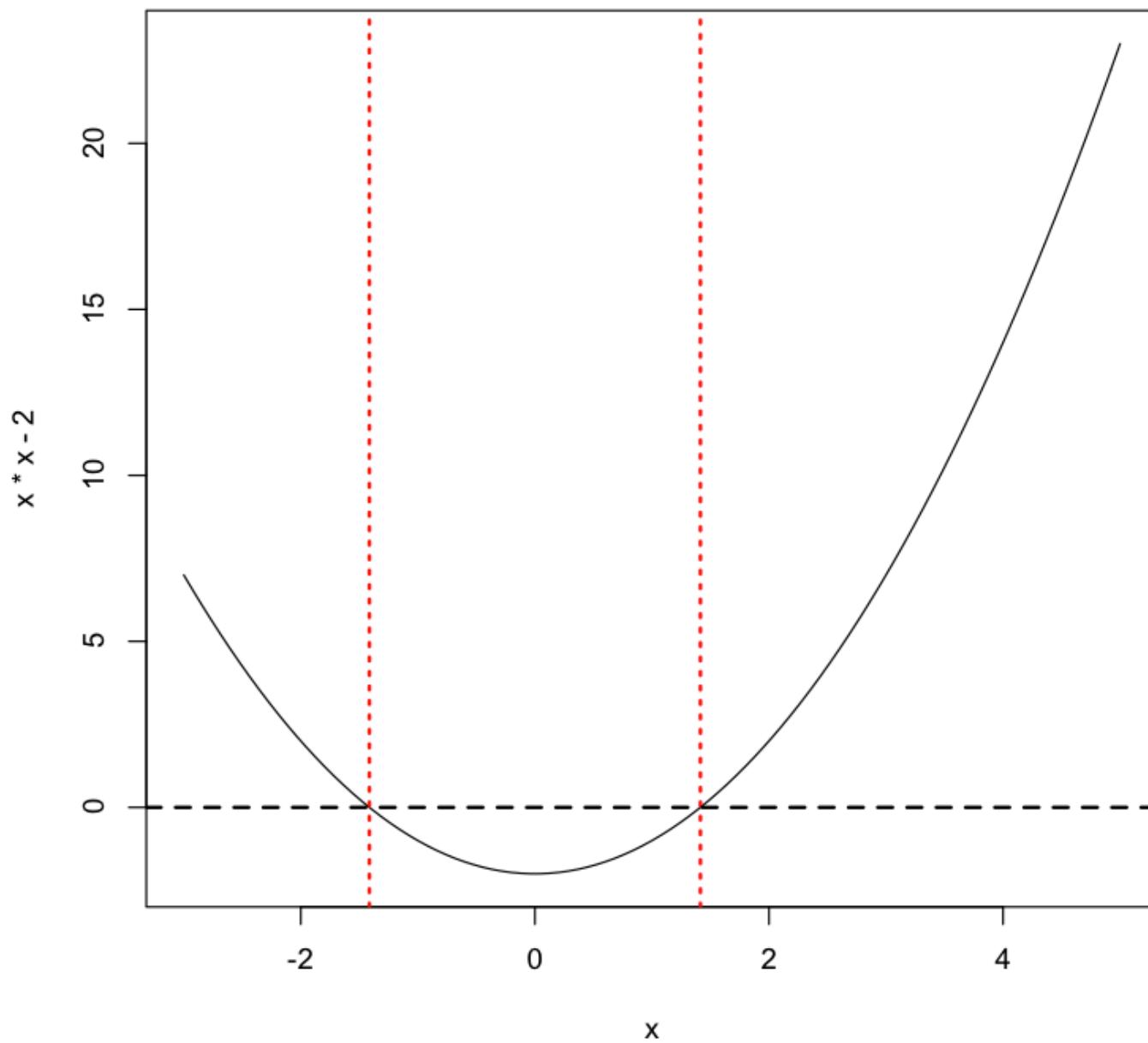
Pseudocode on blackboard: DinnerParty\_Pseudocode.R

# Function format

```
x <- function(args,...){  
  #put code here  
}  
  
# Let's calculate the log of n factorial [i.e. log(n x (n-1) x (n-2) x ... x 2 x 1)]  
logfactorial<-function(n){  
  logfac<-0  
  while (n>1){  
    logfac <- logfac+log(n)  
    n <- n-1  
  }  
  return (logfac)  
}  
  
> logfactorial(300)  
[1] 1414.906  
> logfactorial(300000)  
[1] 3483469
```

# Finding roots and fixed points

“Root-finding” Chapter 10 of Jones et al.



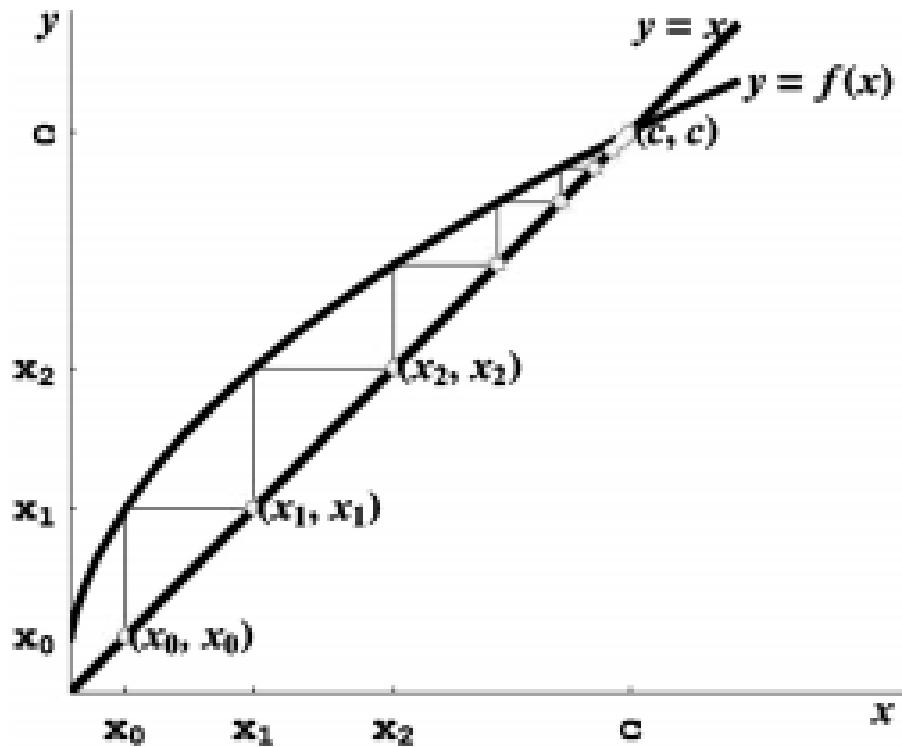
```
curve(x*x-2,-3,5)
abline(h=0,col=1,lwd=2,lty=2)
abline(v=sqrt(2),col=2,lwd=2,lty=3)
abline(v=-sqrt(2),col=2,lwd=2,lty=3)
```

# Roots are important

1. As part of optimization (derivative of a function is 0 at its optima).
2. For finding **fixed-points**, i.e., an  $x$  such that  $f(x)=x$  (because then,  $g(x)=f(x)-x=0$ ).

# Fixed point iteration

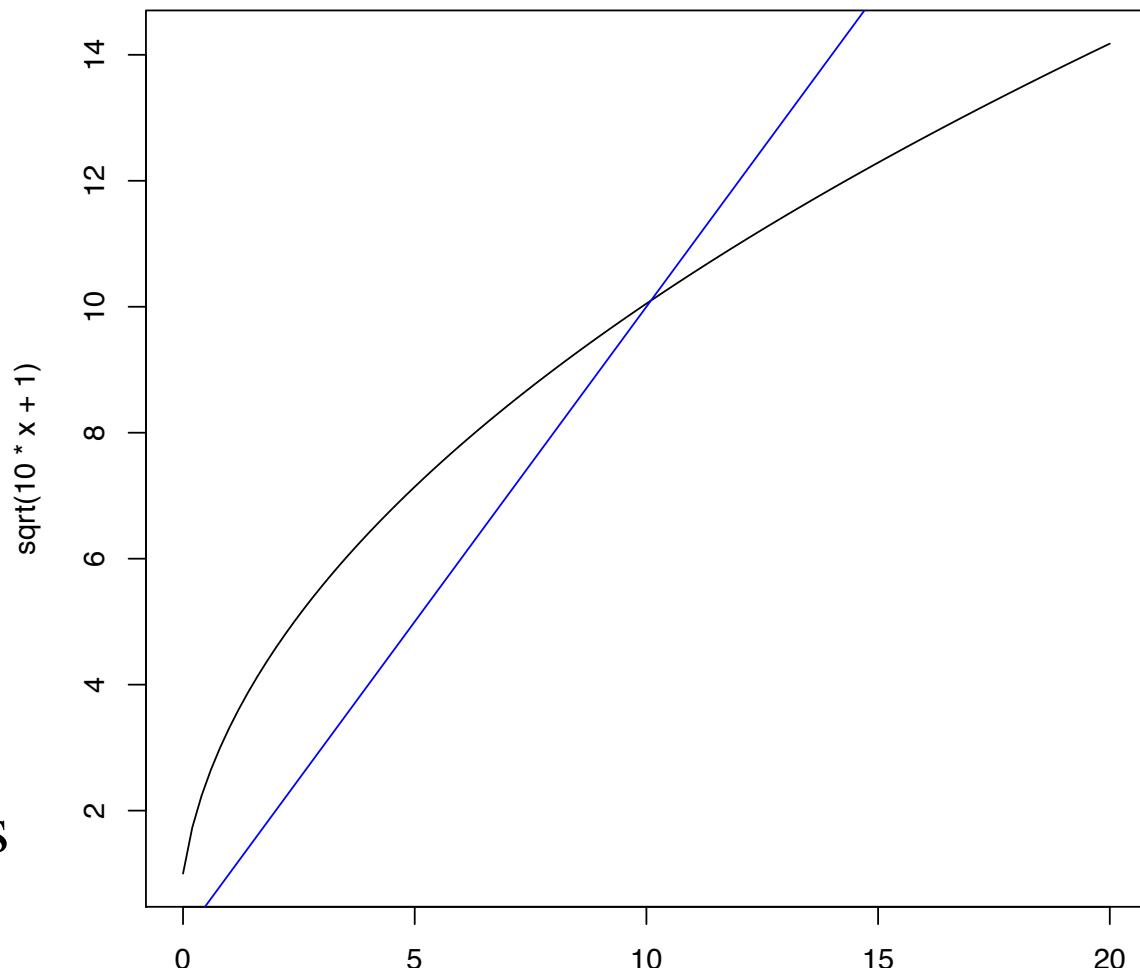
- The fixed point(s) are the points at which the graph of the function intercepts the line  $y=x$



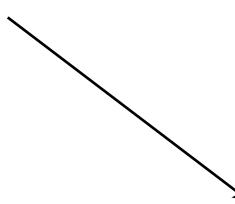
- Start at  $x_0$ .
- For  $i=1,2,3,\dots$ 
  - Set  $x_i = f(x_{i-1})$
- When should we stop?

# Saving plots to file

$y = \sqrt{10x + 1}$

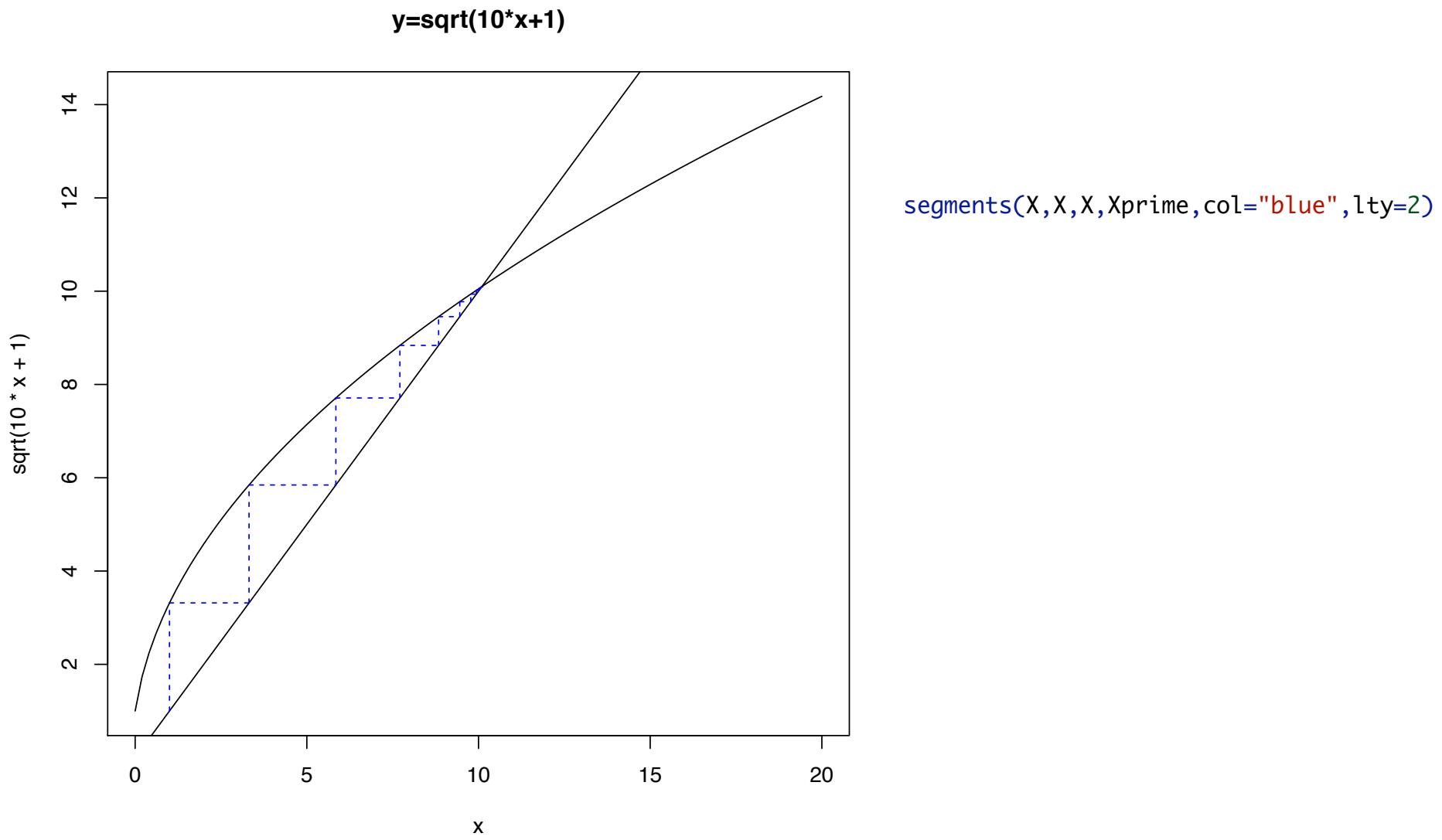


Plots functions



```
pdf("Fig2.pdf")
curve(sqrt(10*x+1), 0, 20, main="y=sqrt(10*x+1)")
abline(coef=c(0,1), col="blue")
dev.off()
```

# Fixed point iteration in action



# On debugging...

[FixedPoints - Attempts 1-7]

```
1 # write the function sqrt(10*x+1)
2 F1<-function(x){
3     return(sqrt(10*x+1))
4 }
5
6 pdf("Fig3.pdf")
7 curve(sqrt(10*x+1),0,20,main="y=sqrt(10*x+1)")
8 abline(coef=c(0,1),col=4)
9 dev.off()
10
11 # set initial parameters (maximum number of steps; accuracy required; start point)
12 StartingXValue<-0
13 Deviation<-1000 # an arbitrary big number
14 MaxNumberOfIterations<-100
15 Tolerance<-1e-5
16 i<-0
17 Xprime<-StartingXValue
18
19 #Set up a while loop until we hit the required target accuracy or the max. number of steps
20 while ((i<MaxNumberOfIterations)&&(Deviation>Tolerance))
21 {
22     # Set x'=f(x)
23     X<-Xprime # x is our current v-value
24
25     Xprime<-F1(X)
26
27     # calculate accuracy<- |f(x)-x|
28     Deviation<-mod(Xprime-X)
29     X<-Xprime
30 }
31
32 # output the result
33 if (Deviation<Tolerance){
34     cat(paste("Found the fixed point: ",X, "after ", i, "iterations"))
35 }
36 else{
37     cat(paste("Convergence failure. Deviation: ",Deviation, "after ", i, "iterations"))
38 }
```

Attempt 1 - 4 bugs

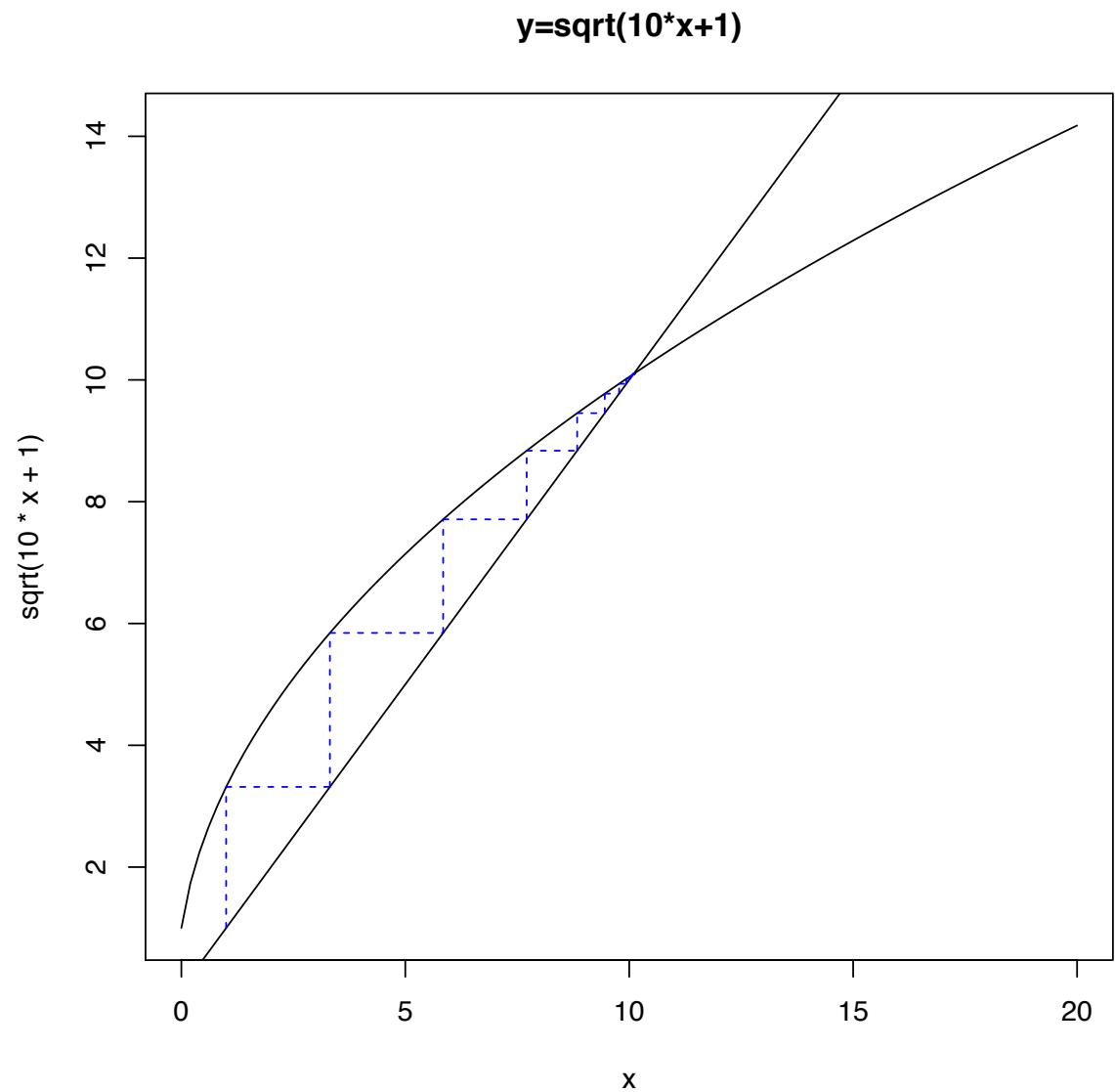
```
15 # function declaration
16 FixedPointFinder<-function(func,StartingValue,Tolerance,MaxNumberOfIterations){
17     # set initial parameters (maximum number of steps; accuracy required; start      point)
18     #StartingXValue<- StartValue
19     #MaxNumberOfIterations<-100
20     #Tolerance<-1e-5
21     Deviation<-1000  # an arbitrary big number
22     i<-0
23     Xprime<-StartingXValue
24 browser()
25     #Set up a while loop until we hit the required target accuracy or the max.  number of steps
26     while ((i<MaxNumberOfIterations)&&(Deviation>Tolerance))
27     {
28         # Set x'=f(x)
29         X<-Xprime # x is our current v-value
30
31         Xprime<-func(X)
32
33         # calculate accuracy<- |f(x)-x|
34         Deviation<-abs(Xprime-X)
35         X<-Xprime
36         i<-i+1
37     }
38
39     # output the result
40     if (Deviation<Tolerance){
41         cat(paste("Found the fixed point: ",X, "after ", i, "iterations"))
42     }else{
43         cat(paste("Convergence failure. Deviation: ",Deviation, "after ", i,      "iterations")))
44 }
```

Attempt 4 as a function  
(debugged) call as:  
`FixedPointFinder(F1,1,1e-5,1000)`  
(say)

So put in a domain check (attempts 5 and 6) ...

```
1 # Now we put a domain check in the function
2 # write the function sqrt(10*x+1)
3 F1<-function(x){
4   if (x<-1){
5     cat("Function not defined for x<-1. Exit.")
6     return("NaN") }
7   return(sqrt(10*x+1))
8 }
```

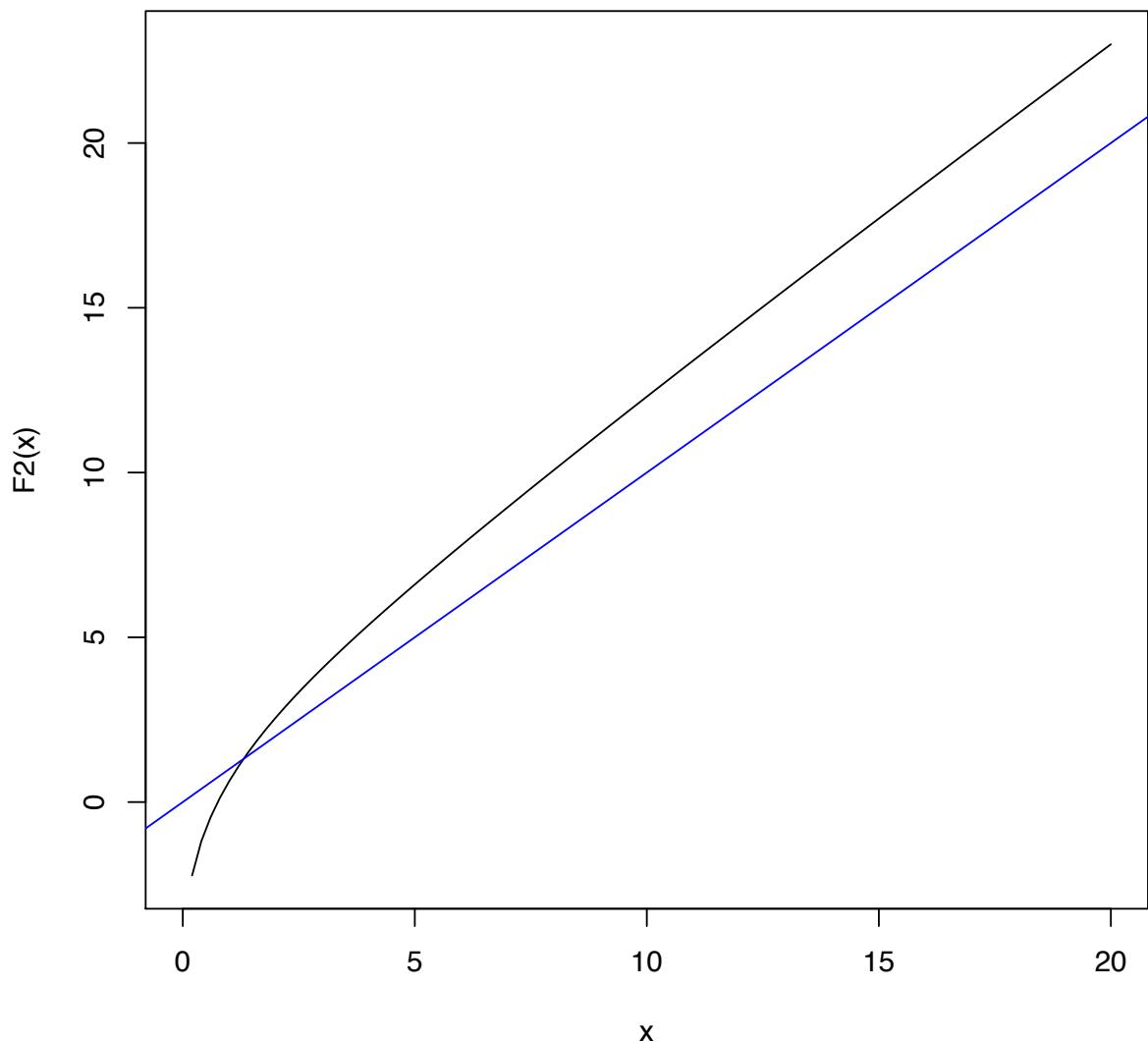
# A better version, that plots progress



On Github as  
“FixedPoints9.Rmd”

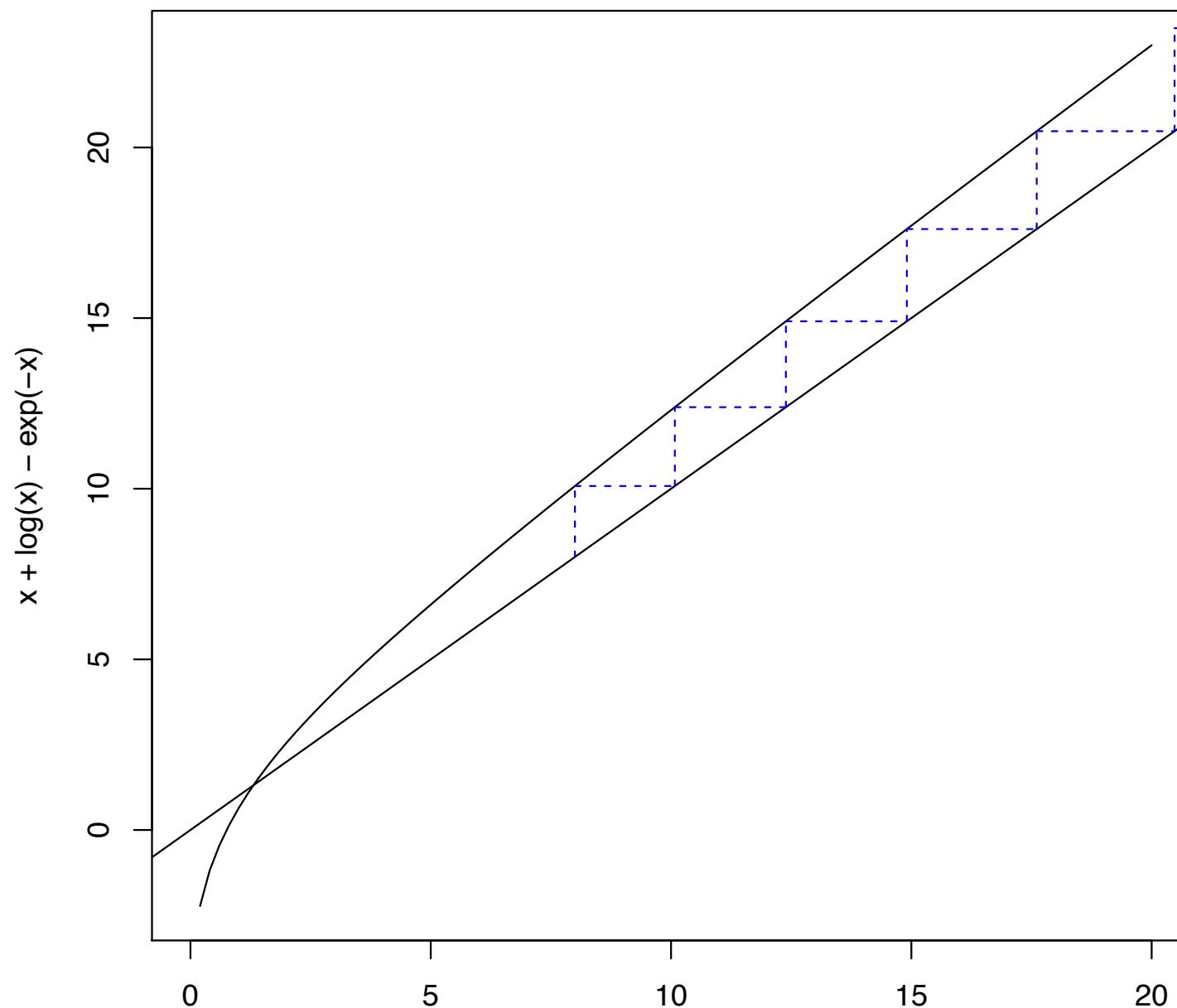
# Let's try a tougher function

$x+\log(x)-\exp(-x)$



```
F2<-function(x){  
  return(x+log(x)-exp(-x))  
}
```

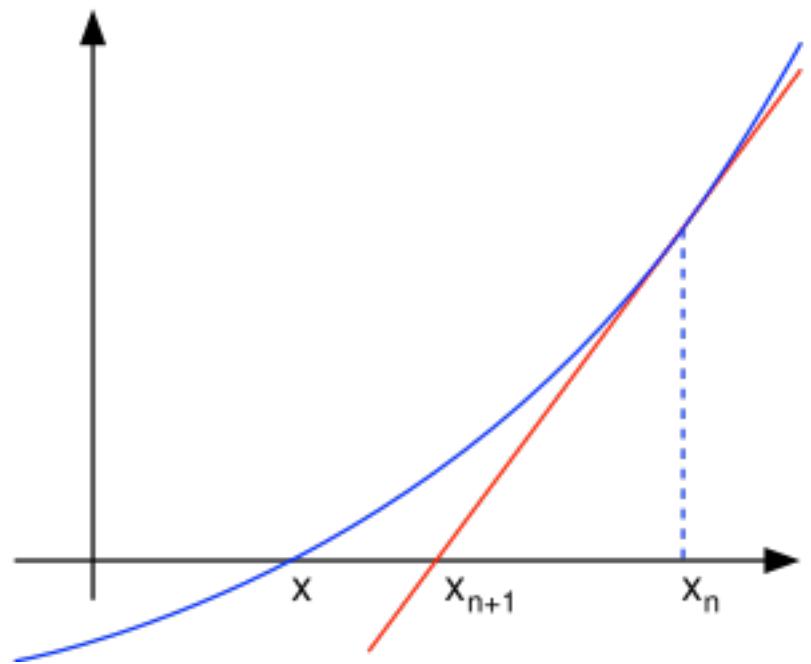
$$y = x + \log(x) - \exp(-x)$$



# Root finding: The Newton-Raphson method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Newton-Raphson to find roots
- Start at an arbitrary value
- Sequence  $\{x_n\}$  converges to a root

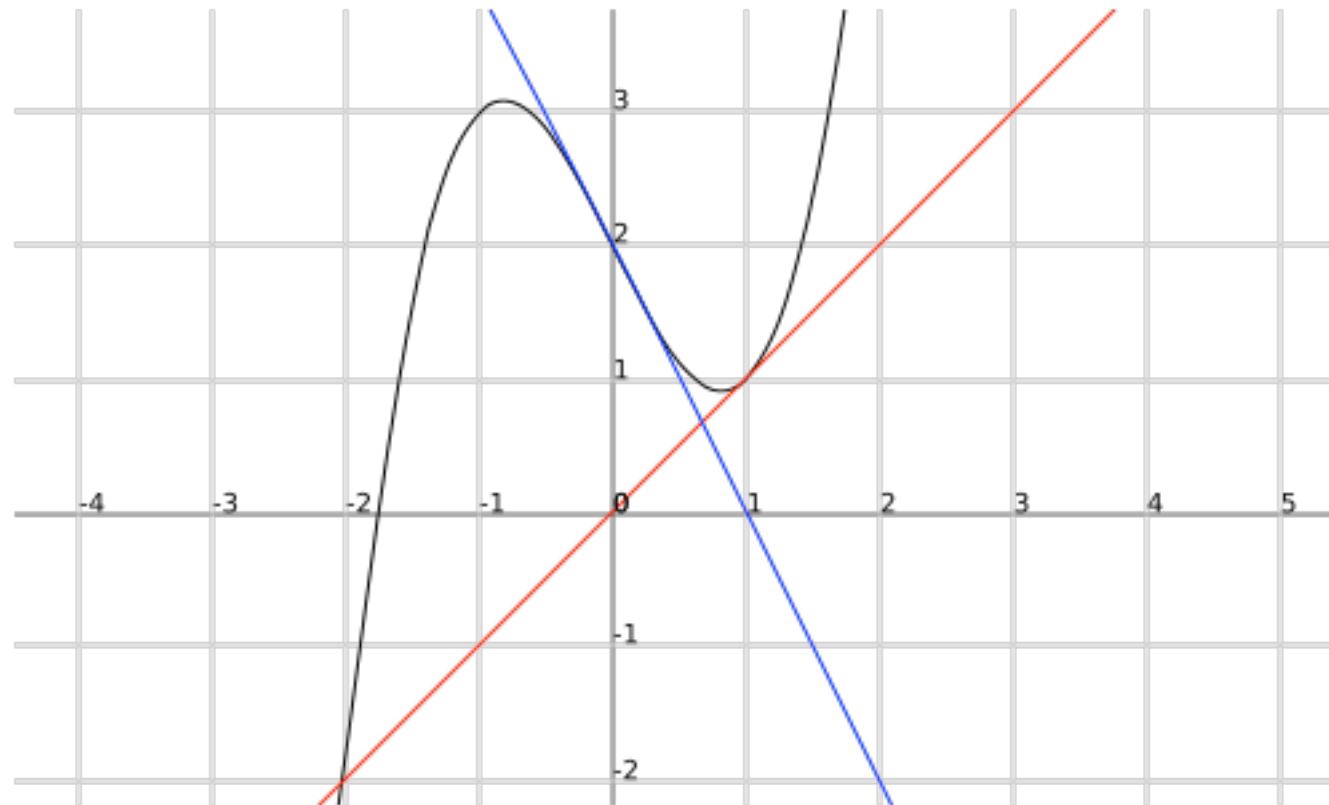


An illustration of one iteration of Newton's method (the function  $f$  is shown in blue and the tangent line is in red). We see that  $x_{n+1}$  is a better approximation than  $x_n$  for the root  $x$  of the function  $f$ .

# Informally

- If we are at  $x_i$ 
  - Take the tangent to  $f(x)$  at  $x= x_i$
  - Find where the tangent intercepts the x-axis. Set this to be  $x_{i+1}$
  - Iterate

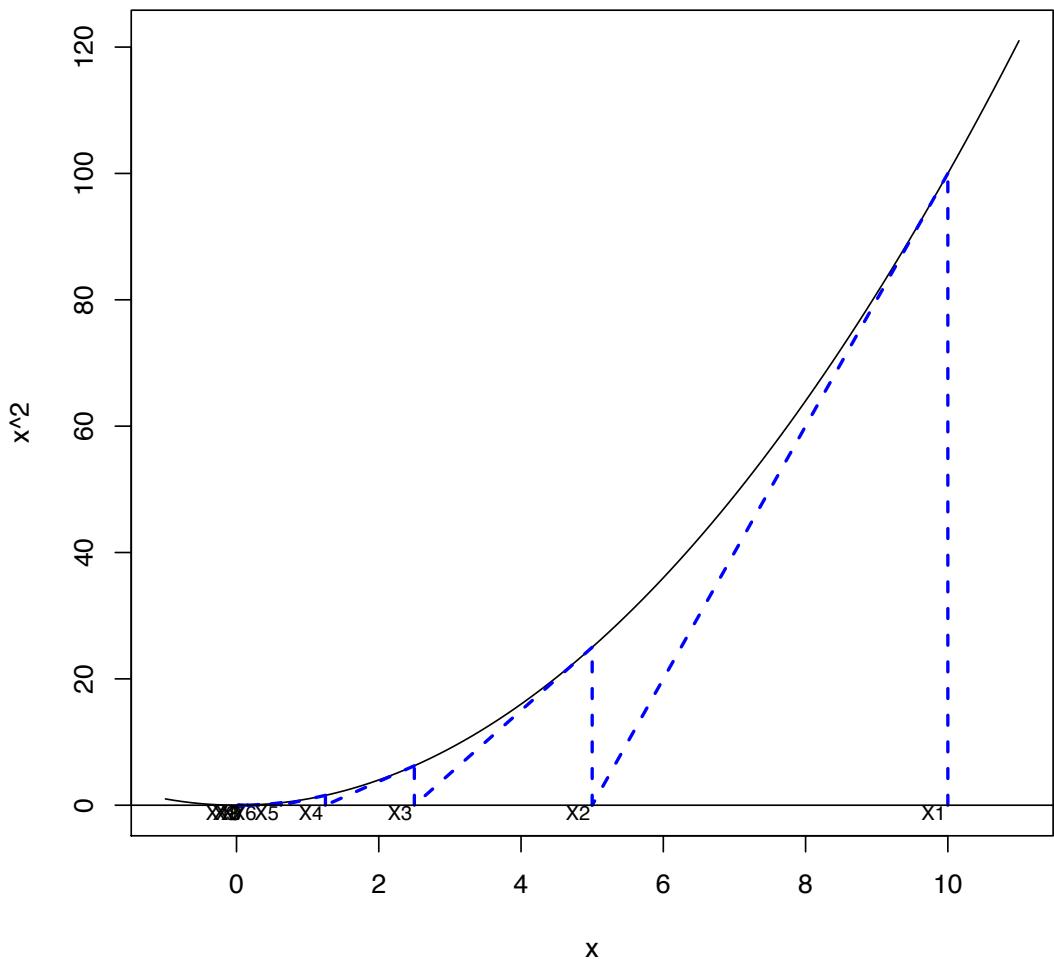
# Which is not to say that it always works...



The tangent lines of  $x^3 - 2x + 2$  at 0 and 1 intersect the  $x$ -axis at 1 and 0 respectively, illustrating why Newton's method oscillates between these values for some starting points.

# Example

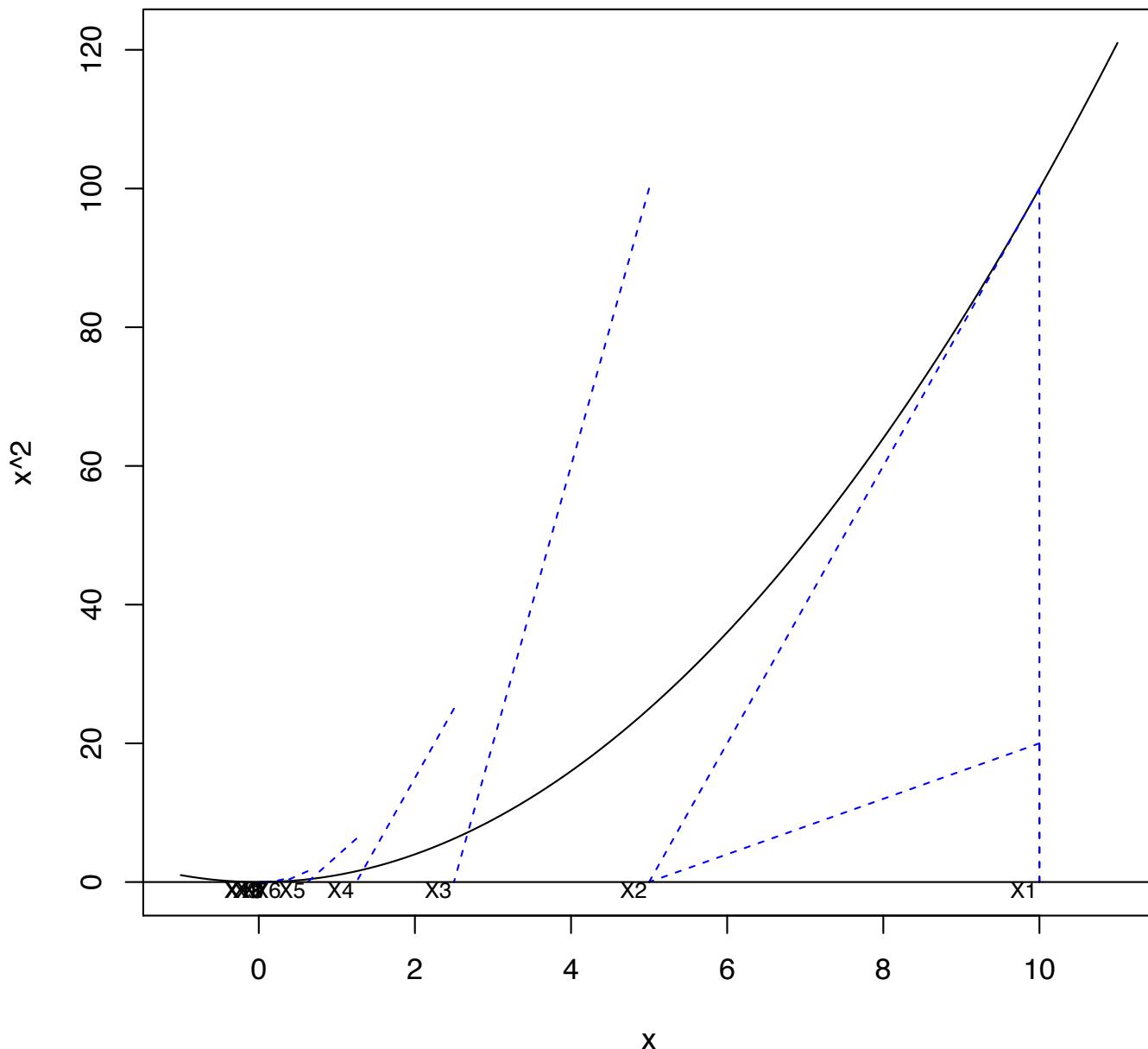
$y=x^2$



Iteration 1 : X= 5 Y= 25  
Iteration 2 : X= 2.5 Y= 6.25  
Iteration 3 : X= 1.25 Y= 1.5625  
Iteration 4 : X= 0.625 Y= 0.390625  
Iteration 5 : X= 0.3125 Y= 0.09765625  
Iteration 6 : X= 0.15625 Y= 0.0244140625  
Iteration 7 : X= 0.078125 Y= 0.006103515625  
Iteration 8 : X= 0.0390625 Y= 0.00152587890625  
Iteration 9 : X= 0.01953125 Y= 0.0003814697265625  
Found the root point: 0.01953125 after 9 iterations

# First attempt

$$y=x^2$$



# Newton-Raphson Pseudocode

```
# a function we will work with
F1<-function(x){
    return(c(x^2,2*x)) # note that the function returns two numbers. The first is f(x); the second is the derivative, f'(x)
}

#define a function F2(x)=sin(x)
#define F3(x)=(x-2)^3-6*x
#define F4(x)=cos(x)-x#
# (All functions need to return f(x) and f'(x))

#Define your Newton-Raphson function ...
```

# Newton-Raphson Pseudocode

```
# Define your Newton-Raphson function
```

```
NewtonRaphson<-function(func,StartingValue,Tolerance,MaxNumberOflterations){
```

```
    #initialize a variable, Deviation (say), to record |f(x)| so that you know how far away you are from 0.
```

```
    #(So initialize it to some arbitrary large number)
```

```
    #Set up a counter, i, to record how many iterations you have performed. Set it equal to 0
```

```
    # Initialize the values of x and f(x)
```

```
    #Set up a while loop until we hit the required target accuracy or the max. number of steps
```

```
    while ((i<MaxNumberOflterations)&&(Deviation>Tolerance))
```

```
{
```

```
    # Record the value of f(x) and f'(x), for the current x value.
```

```
    # I put them in a variable Z. Z[1]<-x; Z[2]<-f(x)
```

```
    # To be safe, check that the function and its derivative are defined at X (either could be NaN if you are unlucky)
```

```
    if ((Z[1]==“NaN”)|| (Z[2]==“NaN”)){
```

```
        cat(“Function or derivative not defined error.\n”)
```

```
        break
```

```
}
```

```
    #Find the next X-value using Newton-Raphson's formula. Let's call that value X
```

```
    # calculate Deviation<- |f(x)-0|
```

```
    # increase the value of your iteration counter
```

```
i<-i+1
```

```
    # if you like, have the program write out how it is getting on
```

```
    cat(paste(“\nIteration ”,i,”: X=”,X,” Y=”,Y))
```

```
}
```

```
    # output the result
```

```
if (Deviation<Tolerance){
```

```
    cat(paste(“\nFound the root point: ”,X, “after ”, i, “iterations”))
```

```
}else{
```

```
    cat(paste(“\nConvergence failure. Deviation: ”,Deviation, “after ”, i, “iterations”)))
```

```
    # have the function return the answer
```

```
return(X)
```

```
}
```

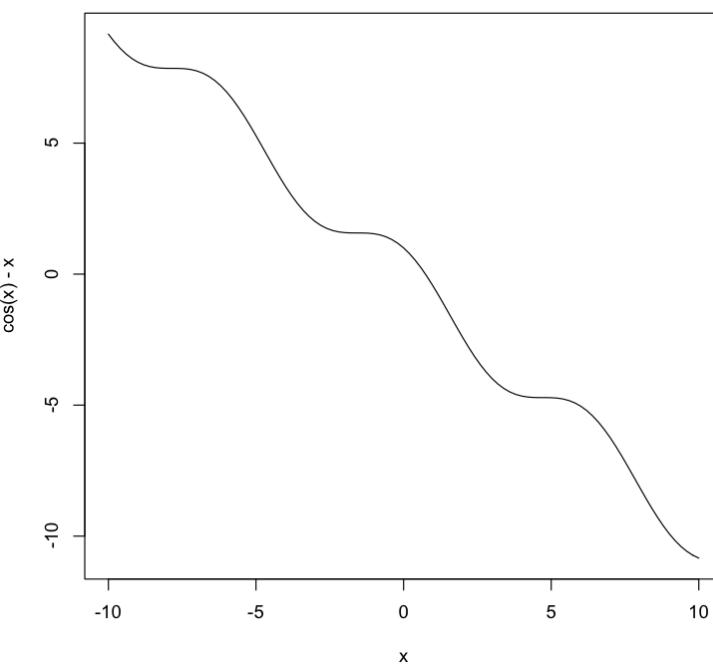
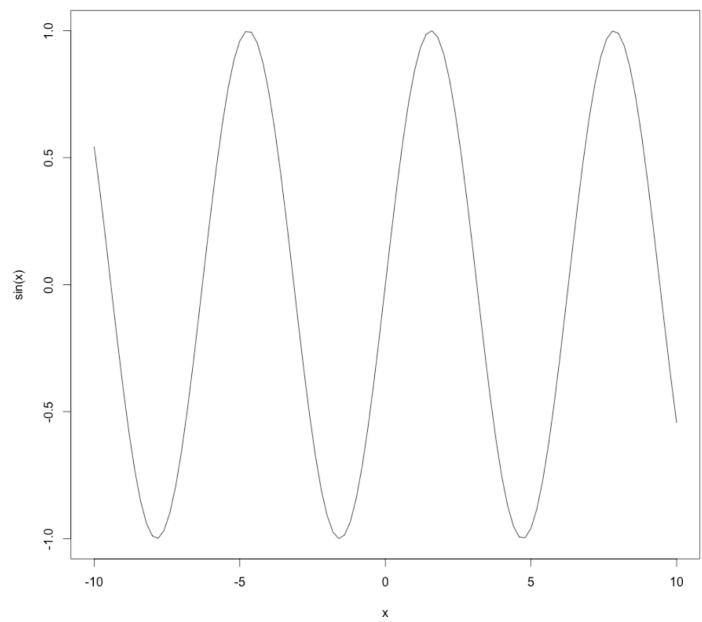
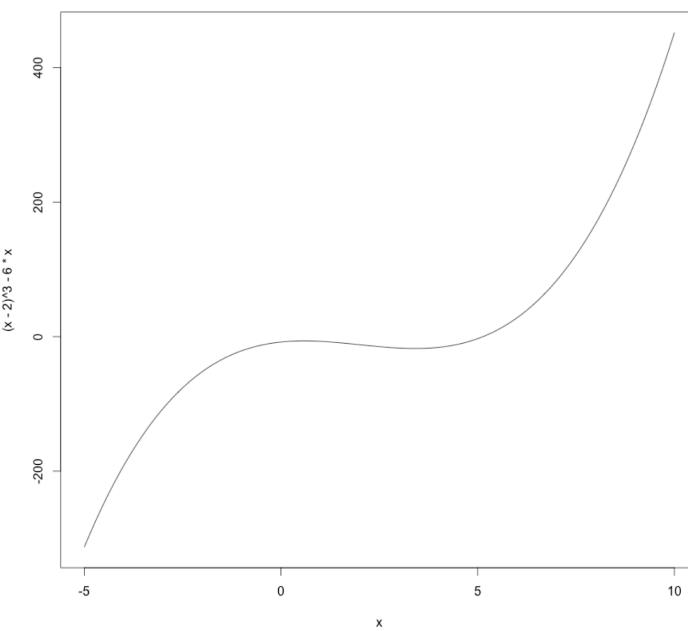
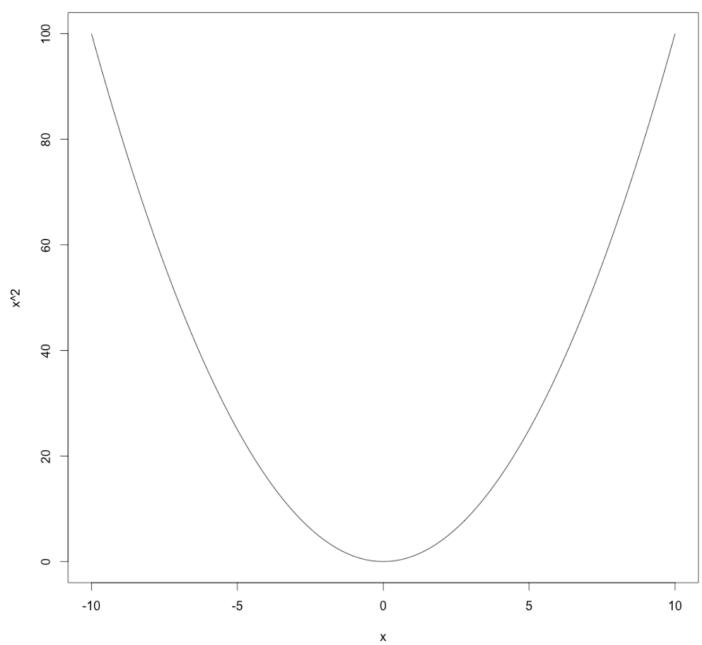
# How to use the function:

```
pdf("Fig6.pdf")
curve(x^2,-1,11,main="y=x^2")
NewtonRaphson(F1,10,1e-3,40)
abline(h=0)
dev.off()
```

‘Week3-NewtonRaphson’ on Github

# Lab task (30 minutes)

- Implement Newton-Raphson in R
- Test it on the following functions:
  - $f(x)=x^2$
  - $f(x)=(x-2)^3-6x$
  - $f(x)=\sin(x)$
  - $f(x)=\cos(x)-x$
- Draw plots showing progress of the algorithm
- When you have something that works, or that is giving you problems, push it to GitHub with `@pmarjora` in your commit message.



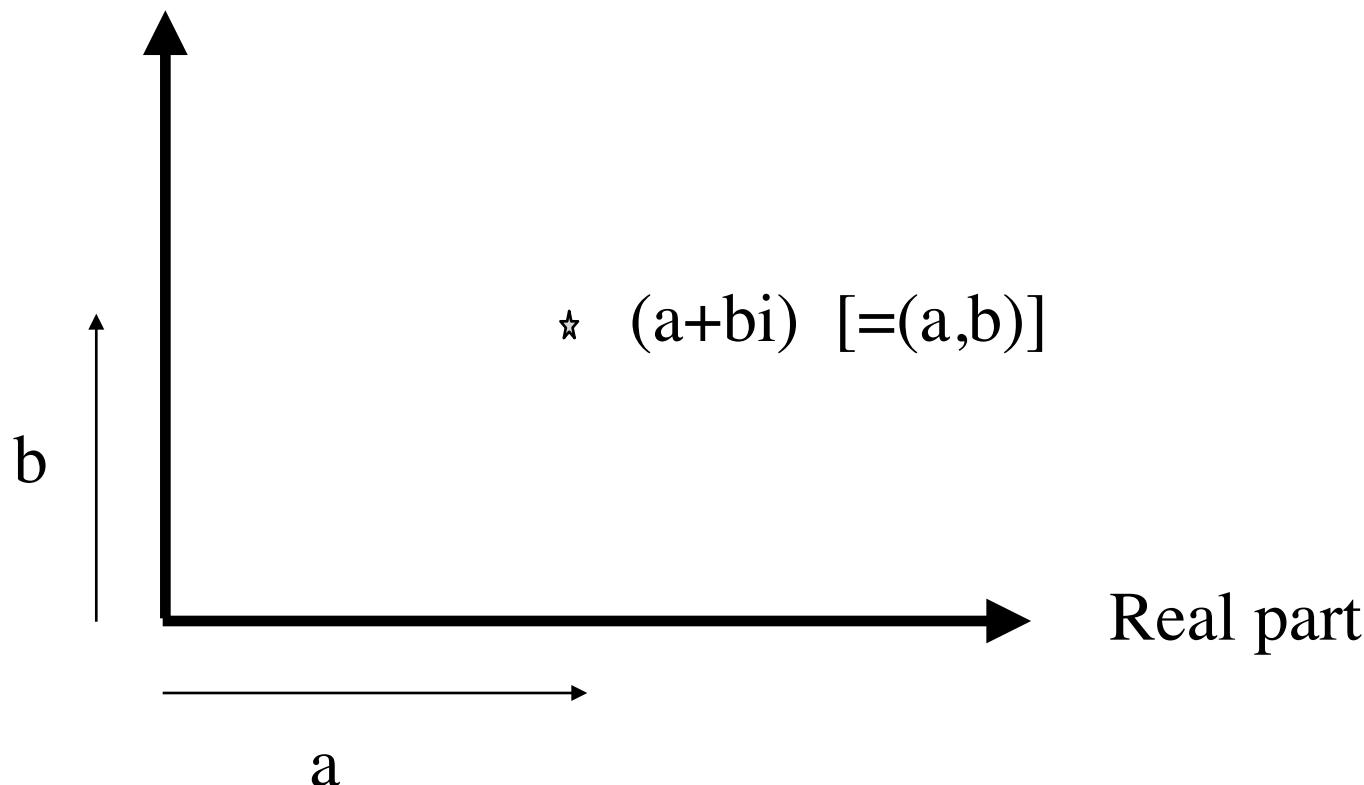
# Newton-Raphson (continued)

- We have seen Newton-Raphson on 1-dimensional space
- It works on higher-dimensional space.
  - e.g. Complex numbers
- Complex numbers:
  - $(a+bi)$
  - $(c+di)$
  - $(a+bi)(c+di) = ac + bdi^2 + adi + bci = (ac - bd) + (ad + bc)i$

Recall  $i^2 = -1$

# Complex numbers on the plane

- $a+bi$ :
- Imaginary part



# Complex math

- $(a+bi)(c+di) = (ac-bd)+(ad+bc)i$
- $(a+bi)/(c+di) = ?$

$$\frac{a + bi}{c + di} = \left( \frac{a + bi}{c + di} \right) \left( \frac{c - di}{c - di} \right) = \frac{(a + bi)(c - di)}{c^2 + d^2}$$

- Derivatives work the same way as for real numbers:
  - e.g. If  $z$  is a complex number, then the derivative of  $cz^n$  is  $ncz^{n-1}$ .

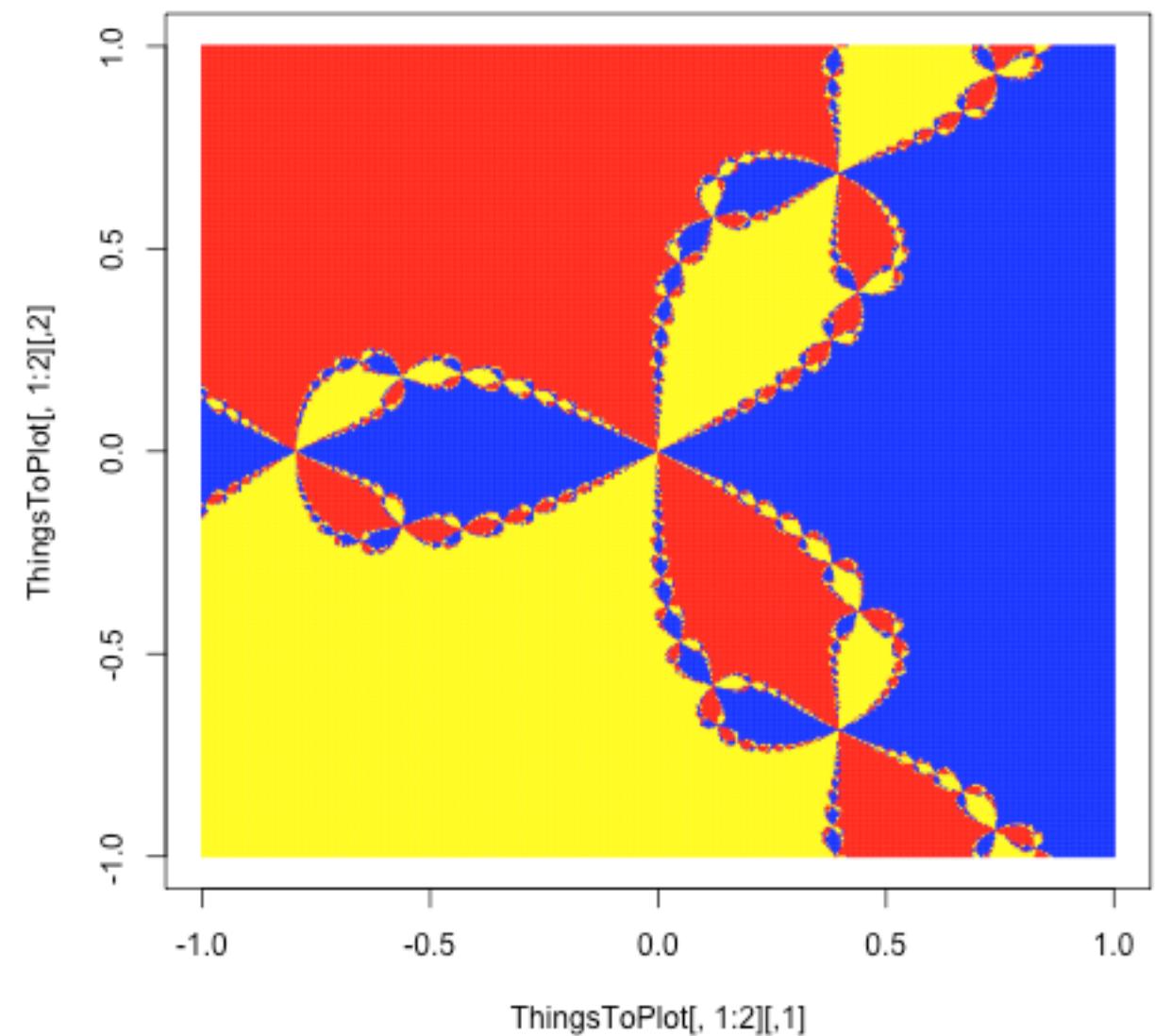
# Newton-Raphson on complex plane

- Works the same way, but using complex derivatives. Complex derivatives much in the same way as the derivatives you are familiar with.
- Try the function  $z^3-1$  (It has three roots).
- Try  $z^4-1$ .
-

# Newton-Raphson task: Newton-Raphson on complex plane

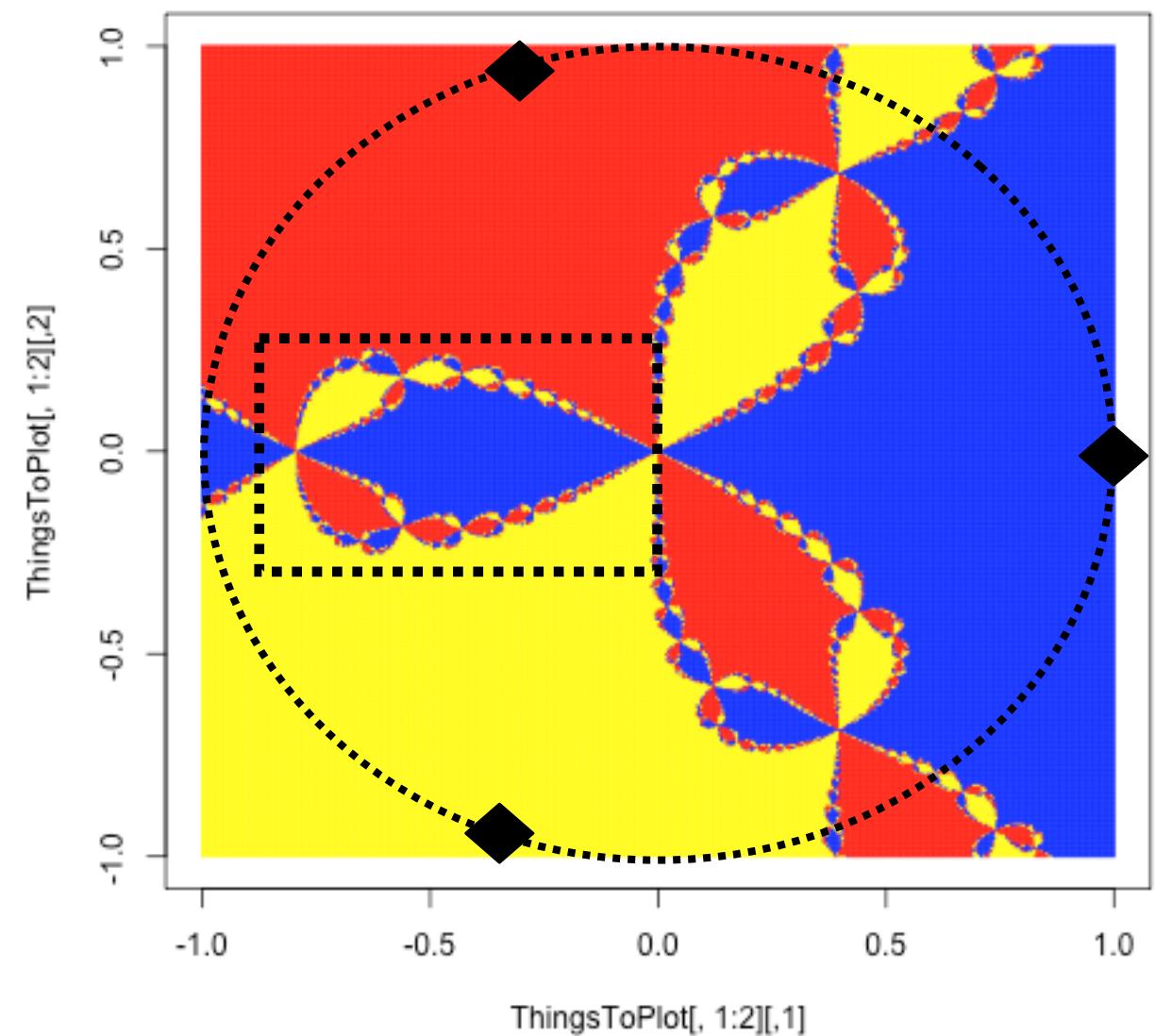
- Construct a plot in which you sample start values for Newton-Raphson on a 2D grid. Let  $x$  go from -1 to +1, and  $y$  go from -1 to +1
- Color each start point by either:
  - a) Which root you reach from that start point
  - b) How many iterations are needed to reach the root you eventually reach from that point.
- What does the resulting plot look like?
- R has built-in support of complex numbers:
  - `complex(length.out = 0, real = numeric(), imaginary = numeric(), modulus = 1, argument = 0)`
  - `> x<-complex(1,3,4)`
  - `> y<-complex(1,3,-4)`
  - `> x+y`
  - `[1] 6+0i`
  - `> x*y`
  - `[1] 25+0i`

$$F(z) = z^3 - 1$$

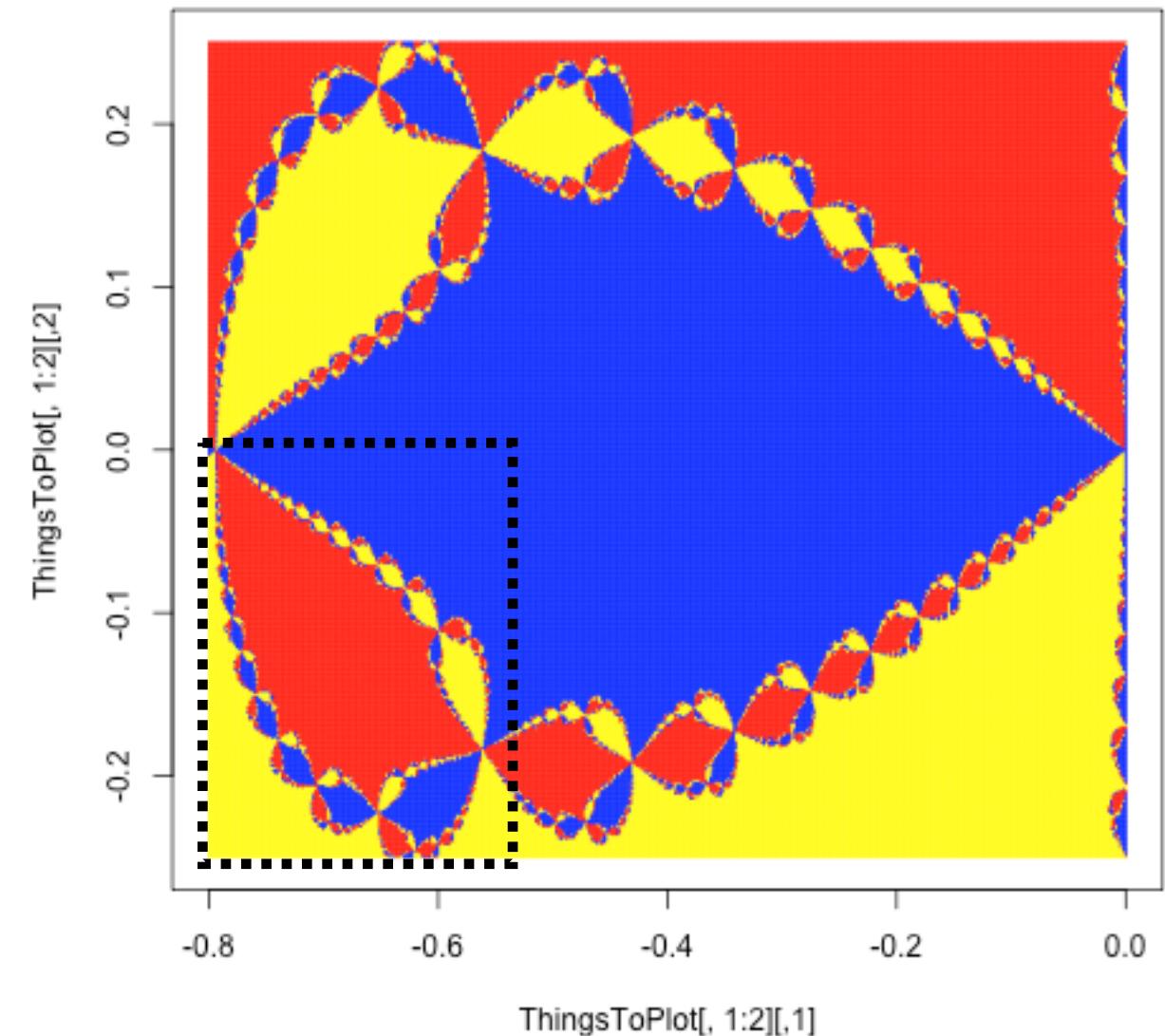


```
png("FigFrac1.png")
A<-RootPlotter(F6,-1,1,500,-1,1,500,0.2)
dev.off()
```

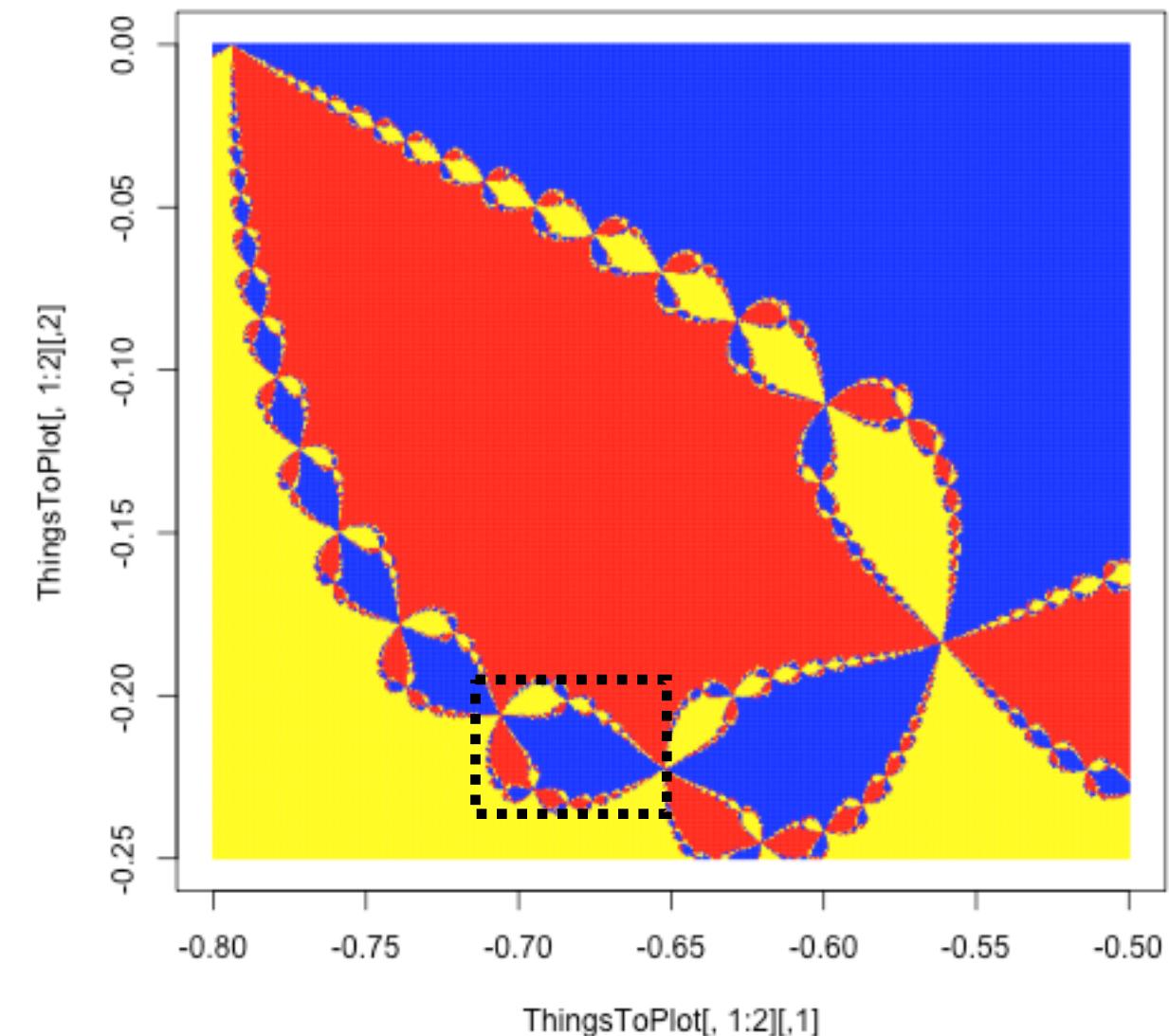
$$F(z) = z^3 - 1$$



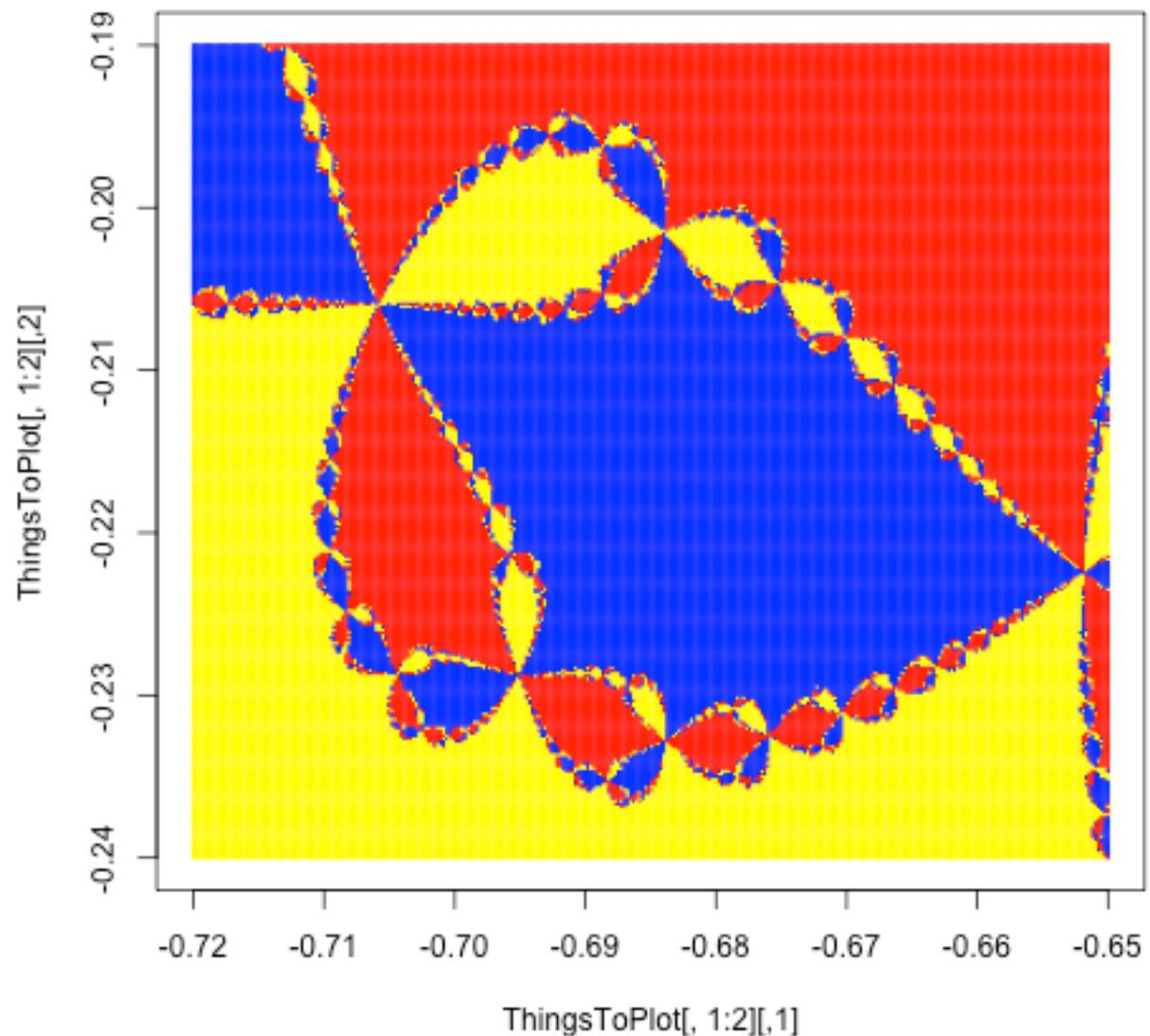
```
png("FigFrac1.png")
A<-RootPlotter(F6,-1,1,500,-1,1,500,0.2)
dev.off()
```



```
png("FigFrac2b.png")
A<-RootPlotter(F6,-0.8,0,500,-0.25,0.25,500,0.2)
dev.off()
```



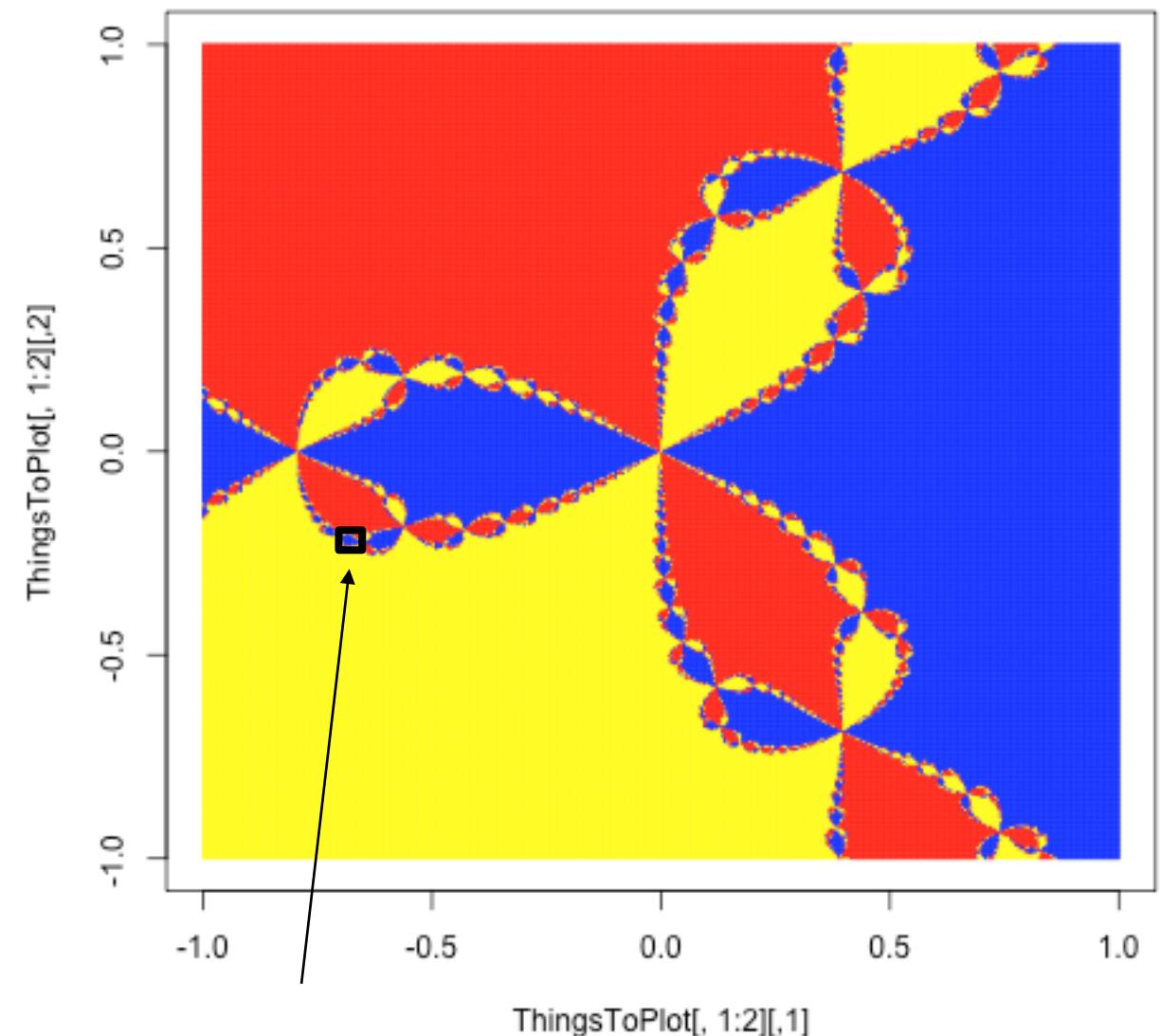
```
png("FigFrac3.png")
A<-RootPlotter(F6,-0.8,-0.5,500,-0.25,0,500,0.2)
dev.off()
```



```
png("FigFrac5.png")
```

```
A<-RootPlotter(F6,-0.72,-0.65,300,-0.24,-0.19,300,0.3)  
dev.off()
```

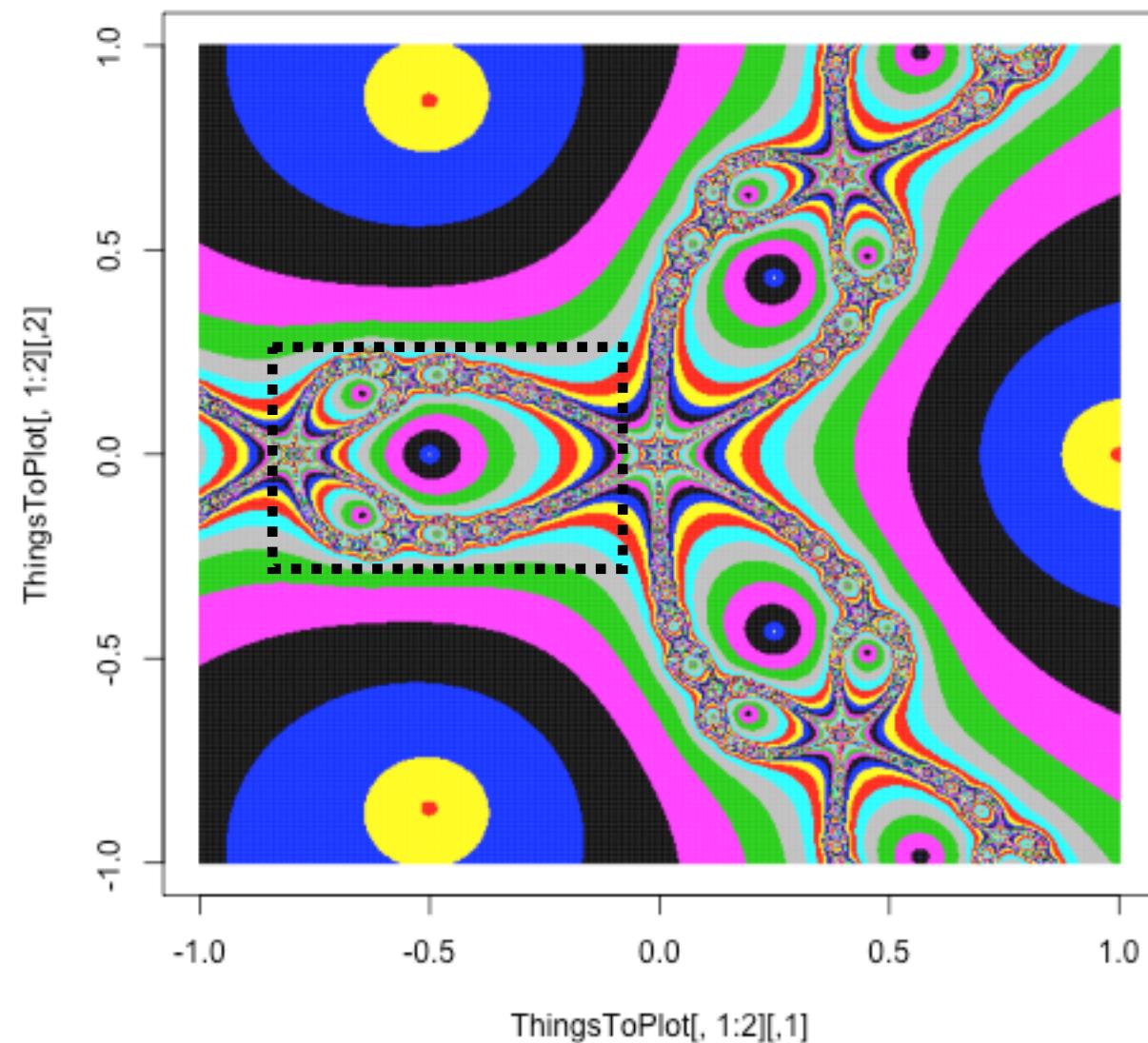
$$F(z) = z^3 - 1$$

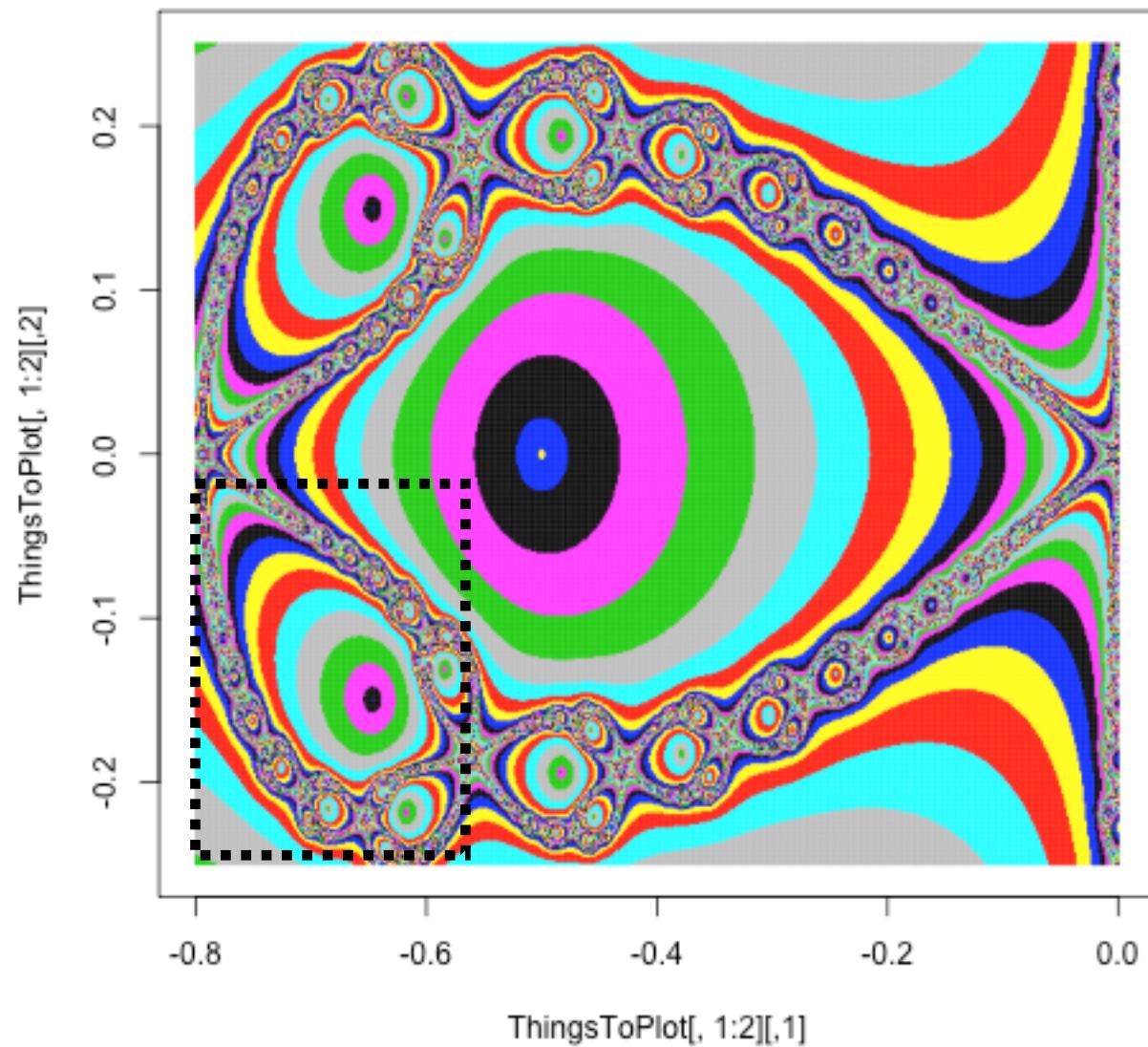


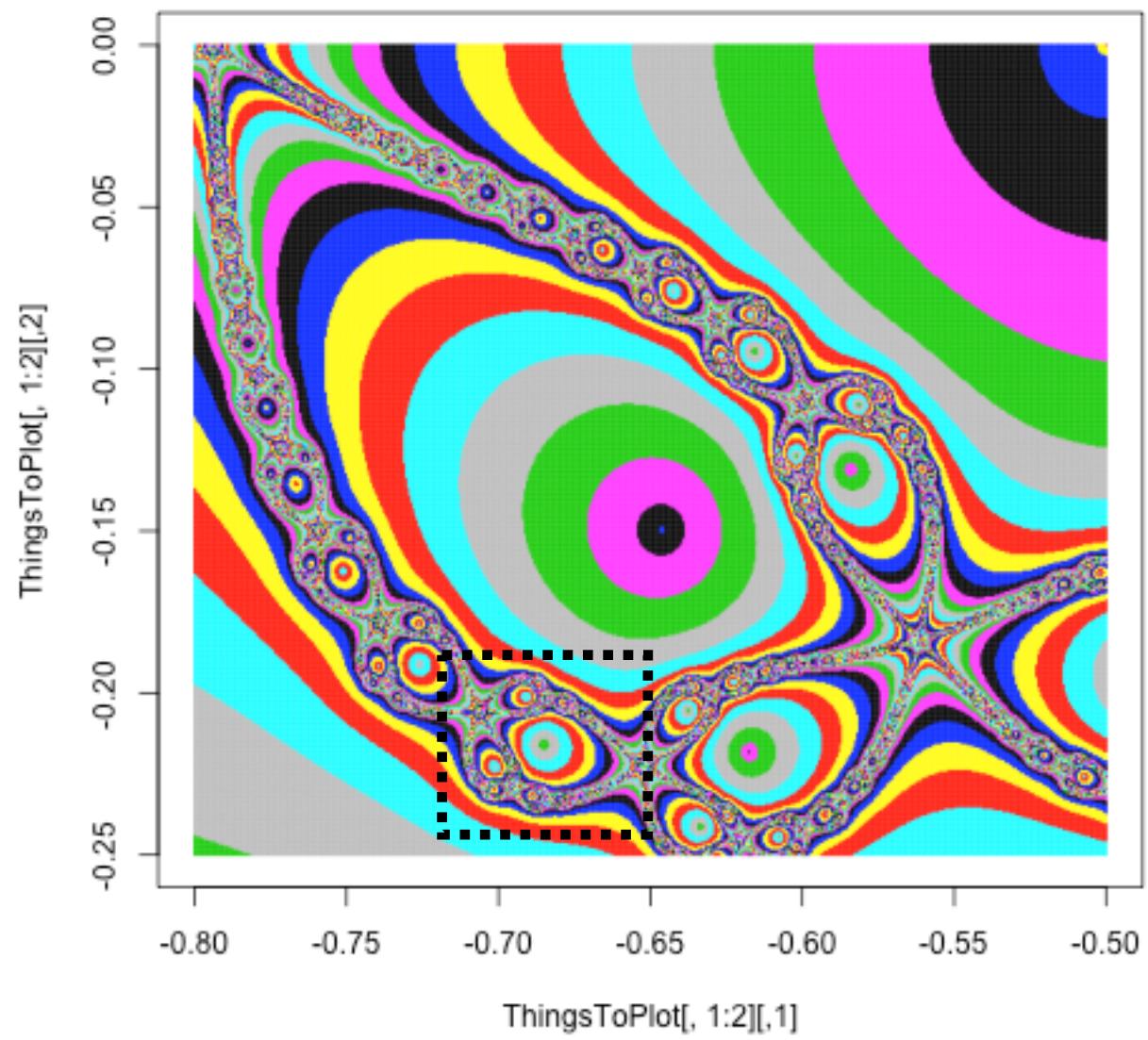
```
png("FigFrac1.png")
A<-RootPlotter(F6,-1,1,500,-1,1,500,0.2)
dev.off()
```

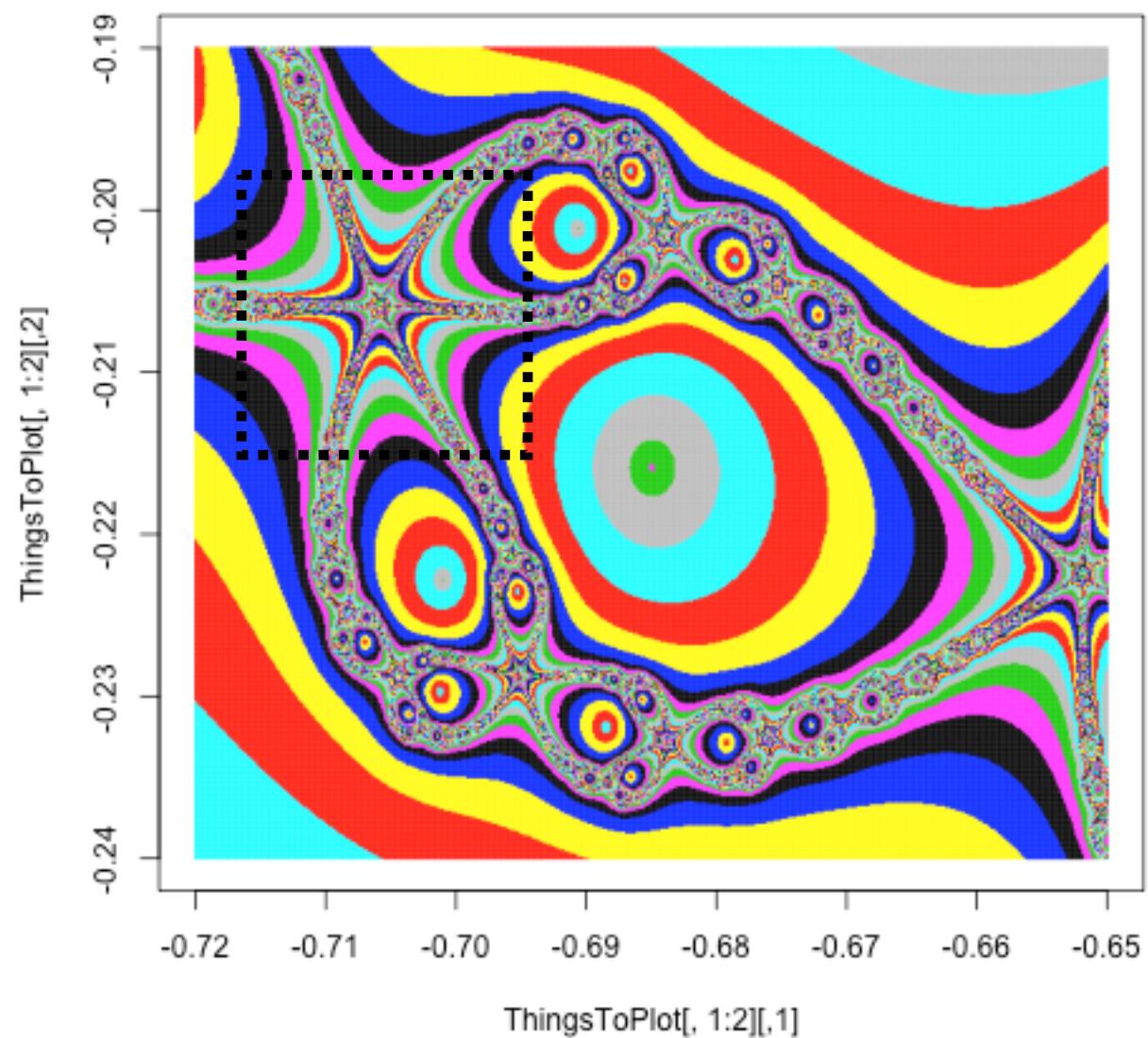
# Color by “number of iterations needed”

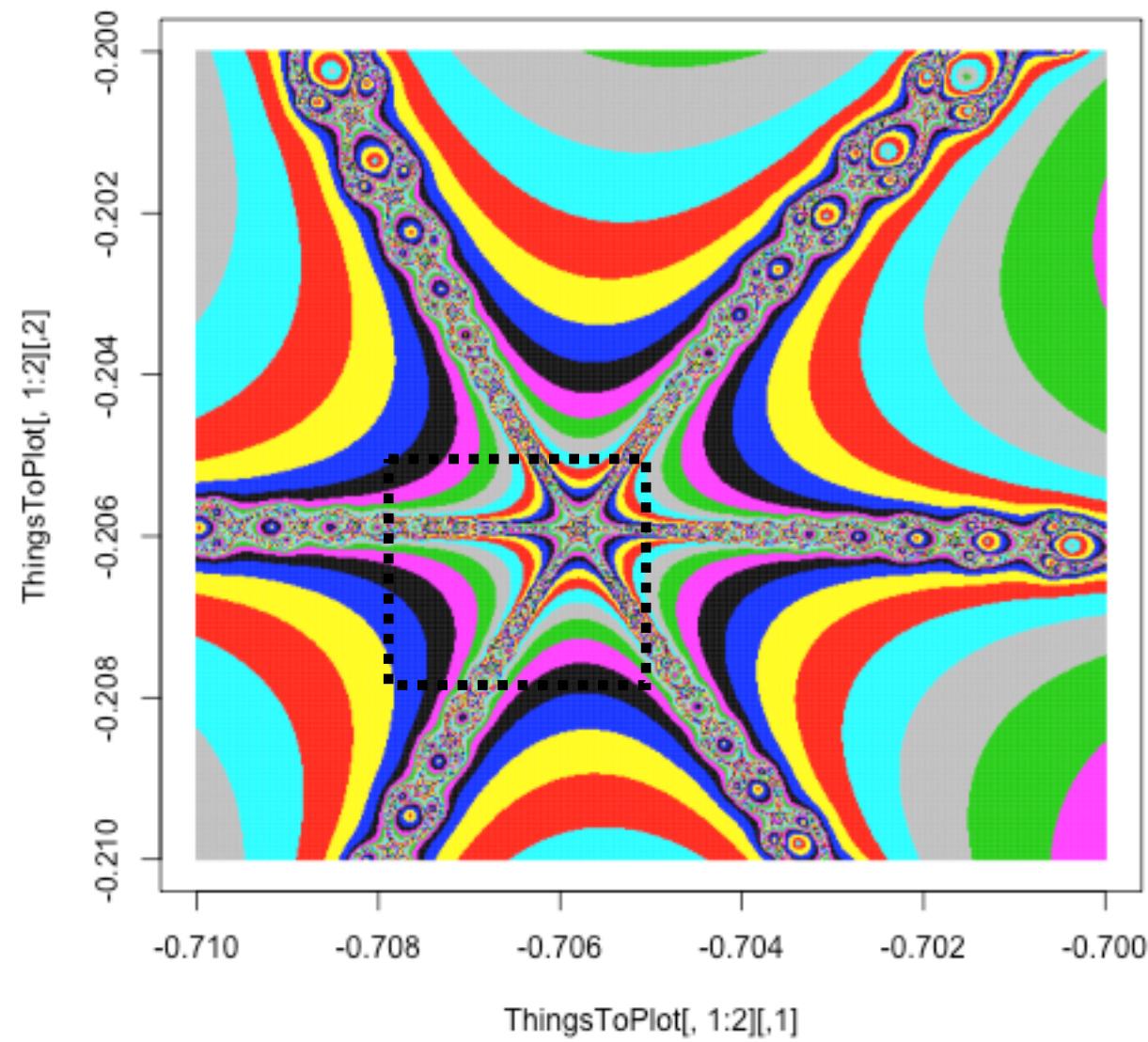
$$F(z) = z^3 - 1$$

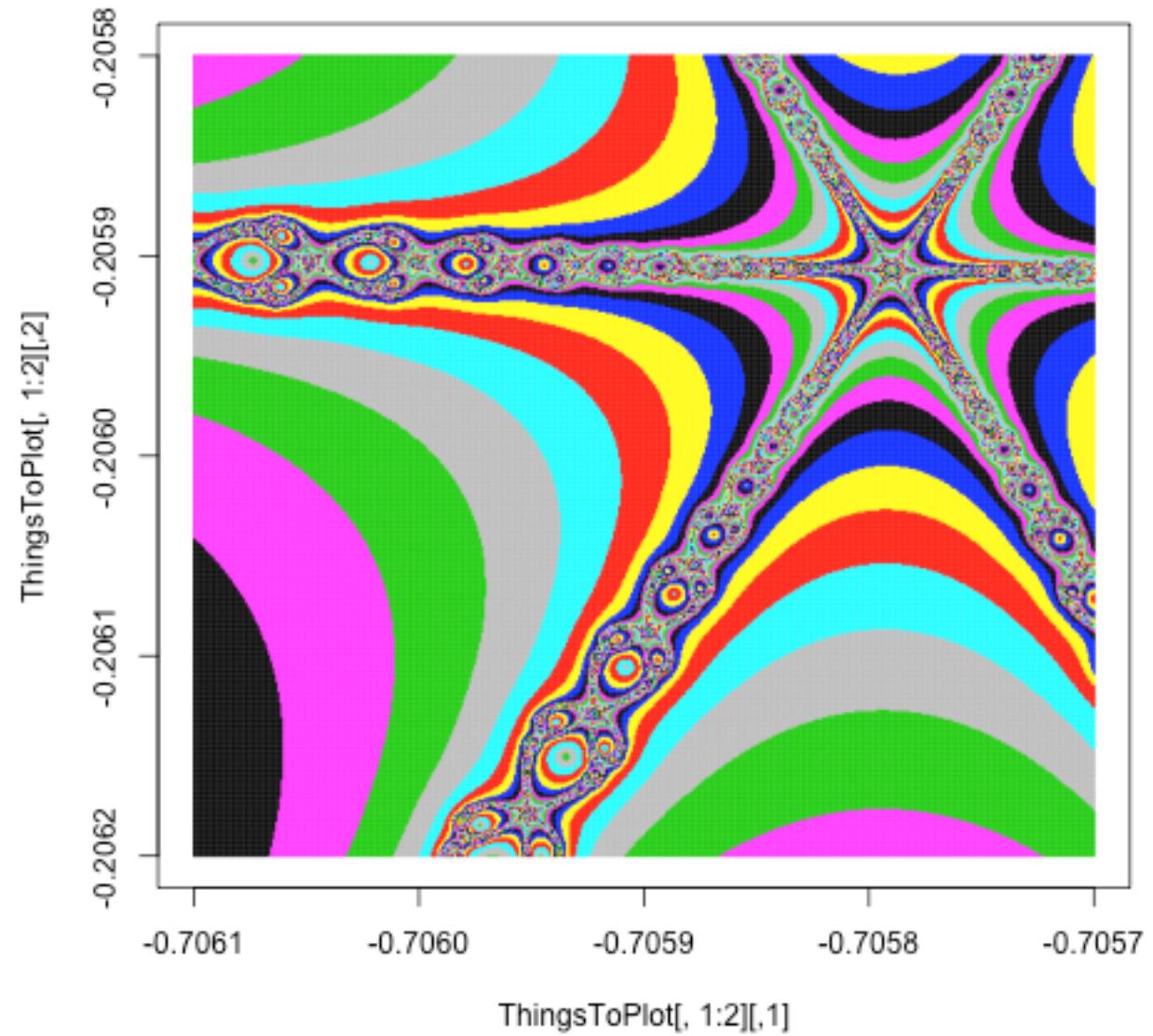




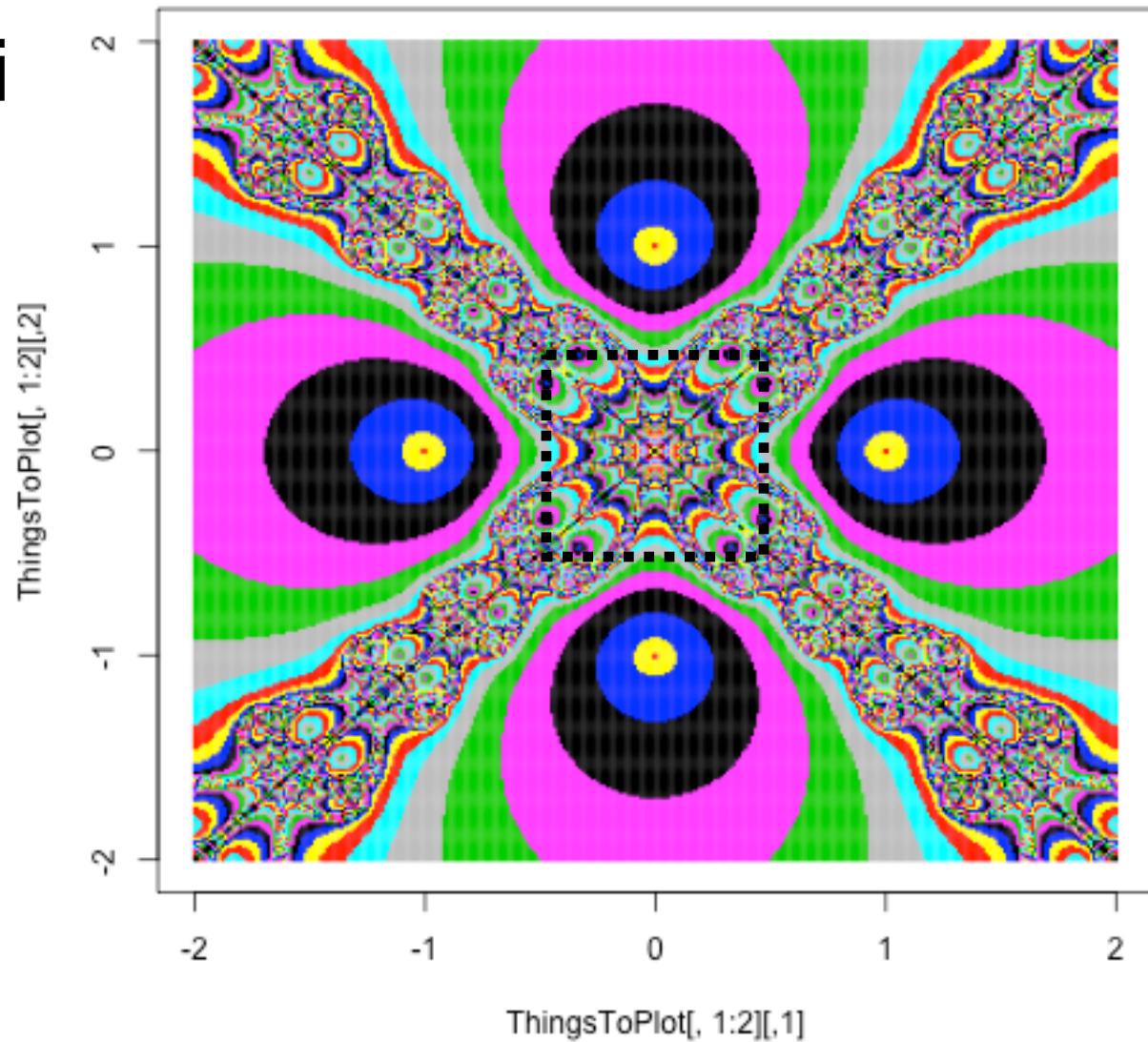






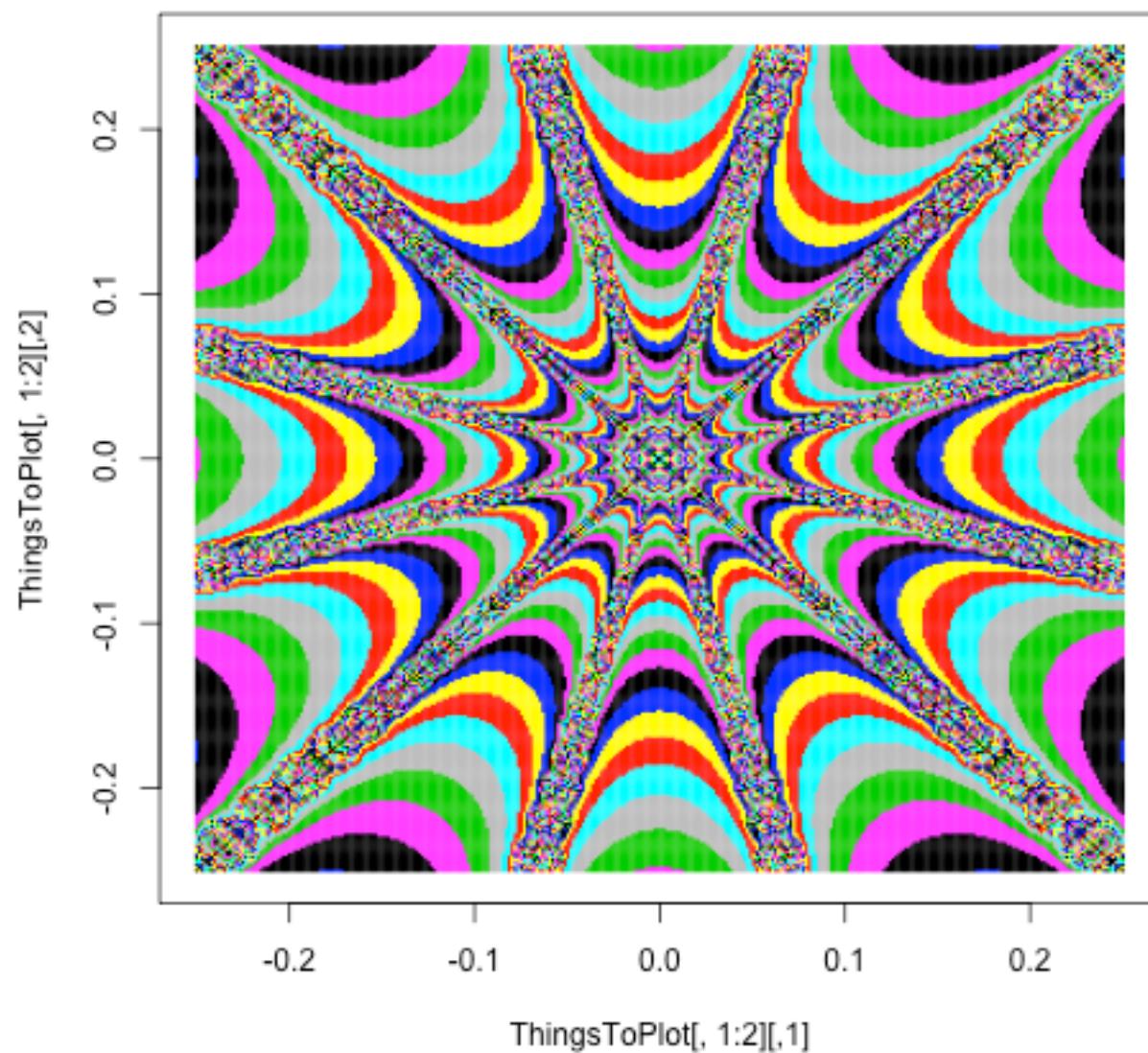


- $F_i$

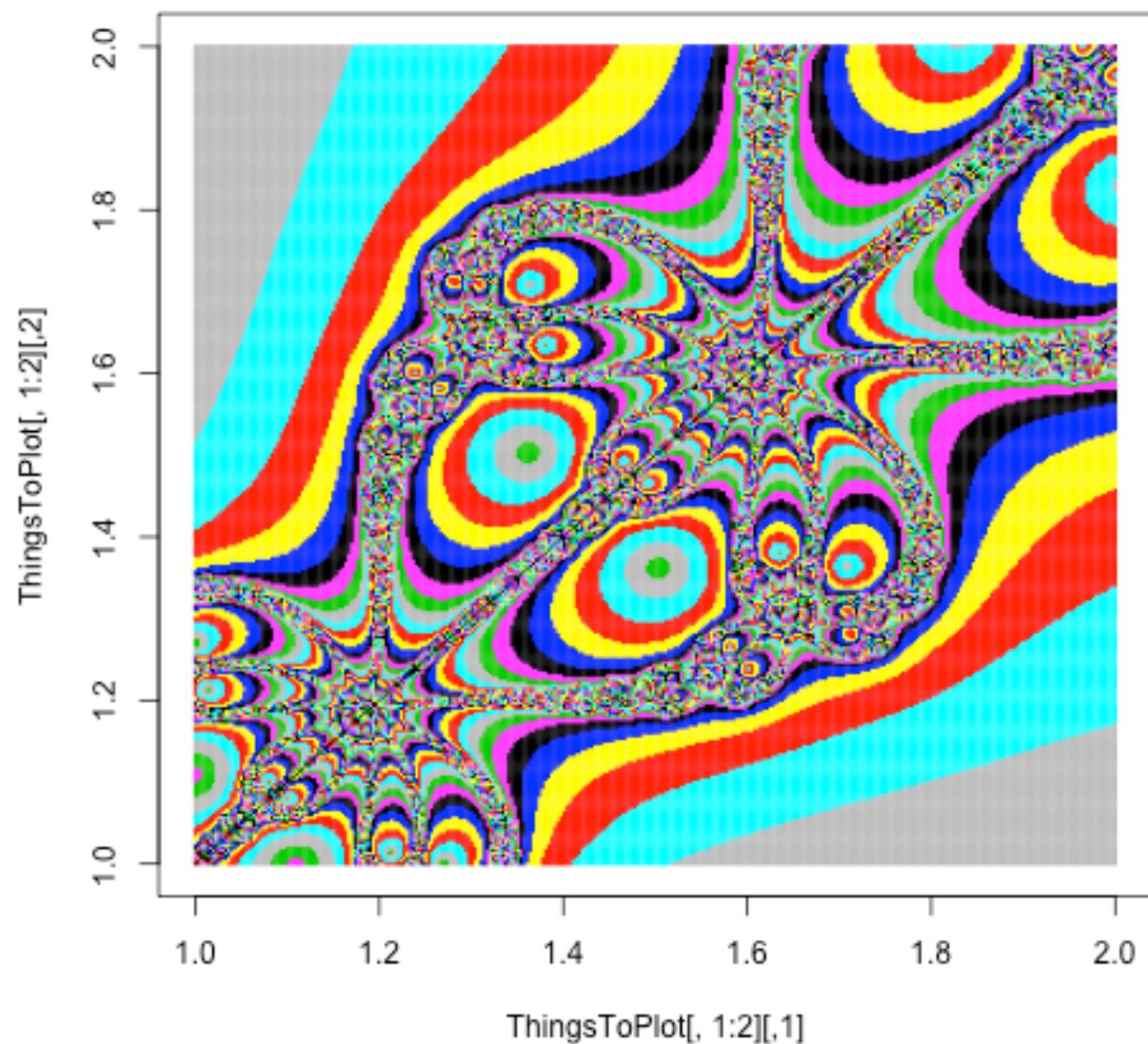


$$F(z) = z^4 - 1$$

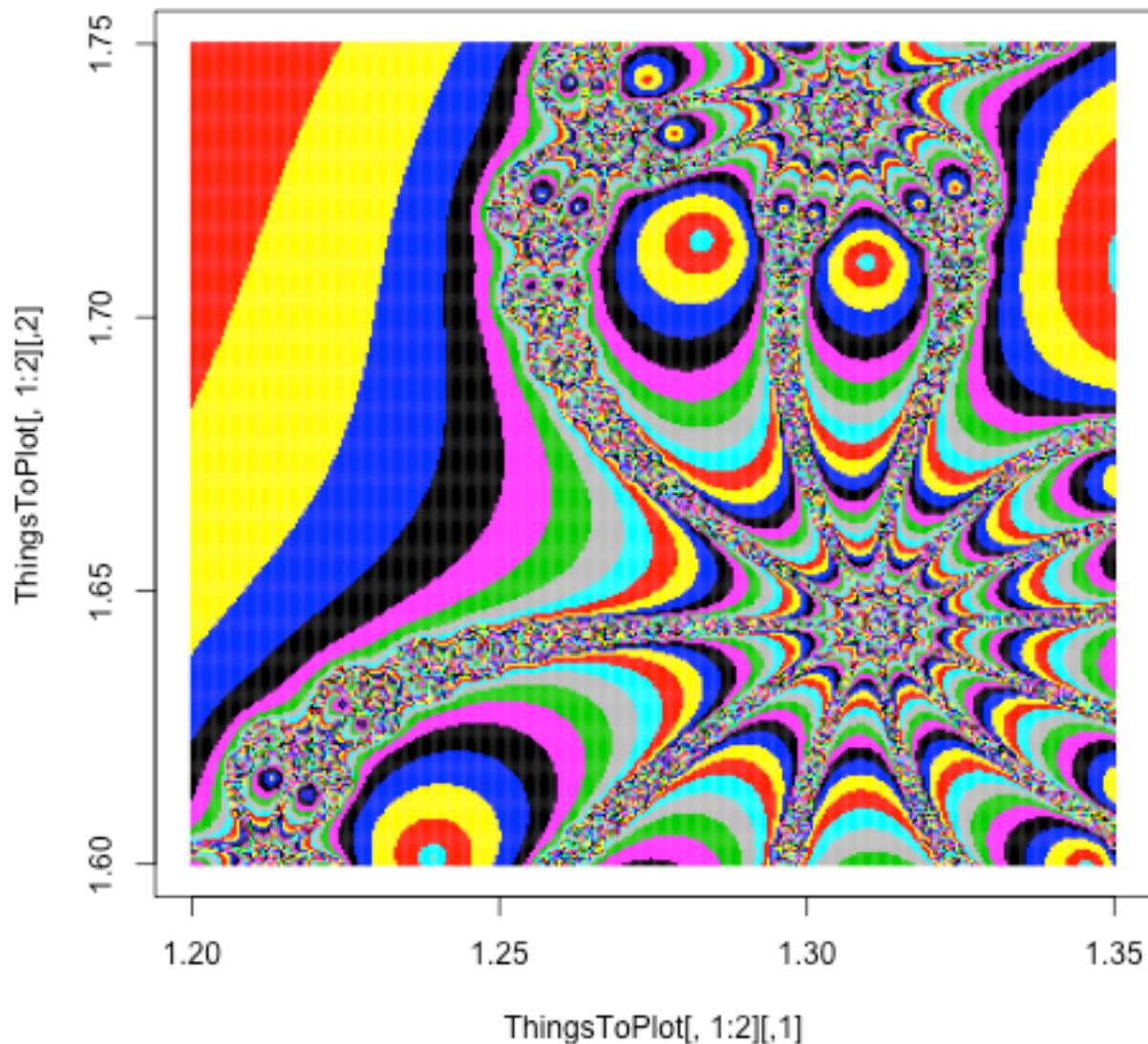
$$F(z) = z^4 - 1$$

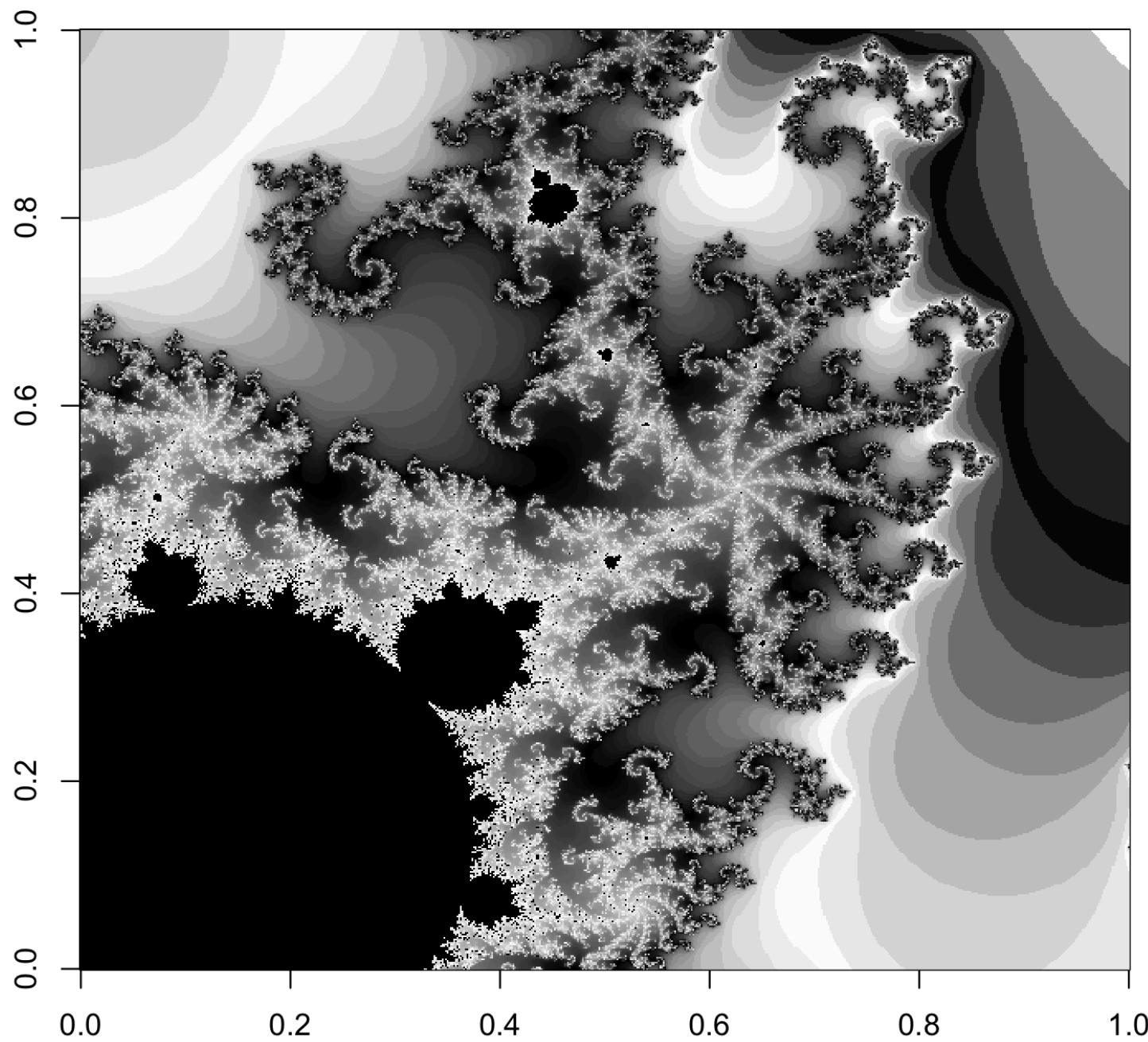


$$F(z) = z^4 - 1$$



$$F(z) = z^4 - 1$$

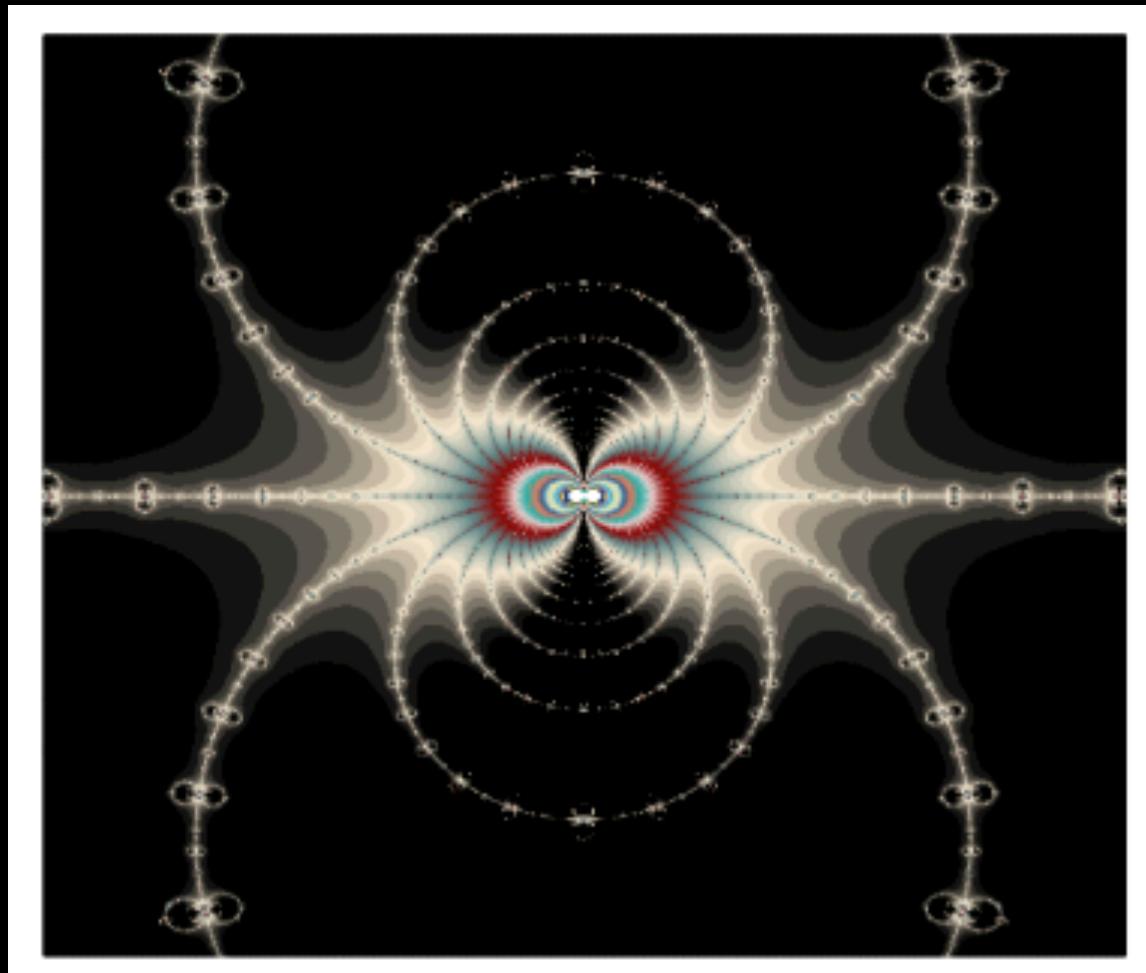




# Examinable assignment 2a: The Art Show

- Take the code for Newton-Raphson on the complex plain from Github:
  - Basic [slow] code: <https://github.com/PM520-Spring-2020/Week3-NewtonRaphsonFractals>. [has pretty graphics].
  - Better [faster] code: <https://github.com/PM520-Spring-2020/Week3-FasterFractals> [**I will walk you through this in a minute**]
- Use it to draw some fractals for some functions **other than those we saw in today's class**. [e.g., Google “Julia Sets”]
- Write a report in Rmarkdown including your R code, and describing the functions you tried, but also including some pretty pictures for display at the start of class in **four week's** time. **They must be drawn by your R code.** (R's gg-plot package is your friend here.)
- Each person should also commit one picture. You are encouraged to give it an pretentious/artsy name. We will vote on who has the best (and I will provide some sort of prize for the best picture).

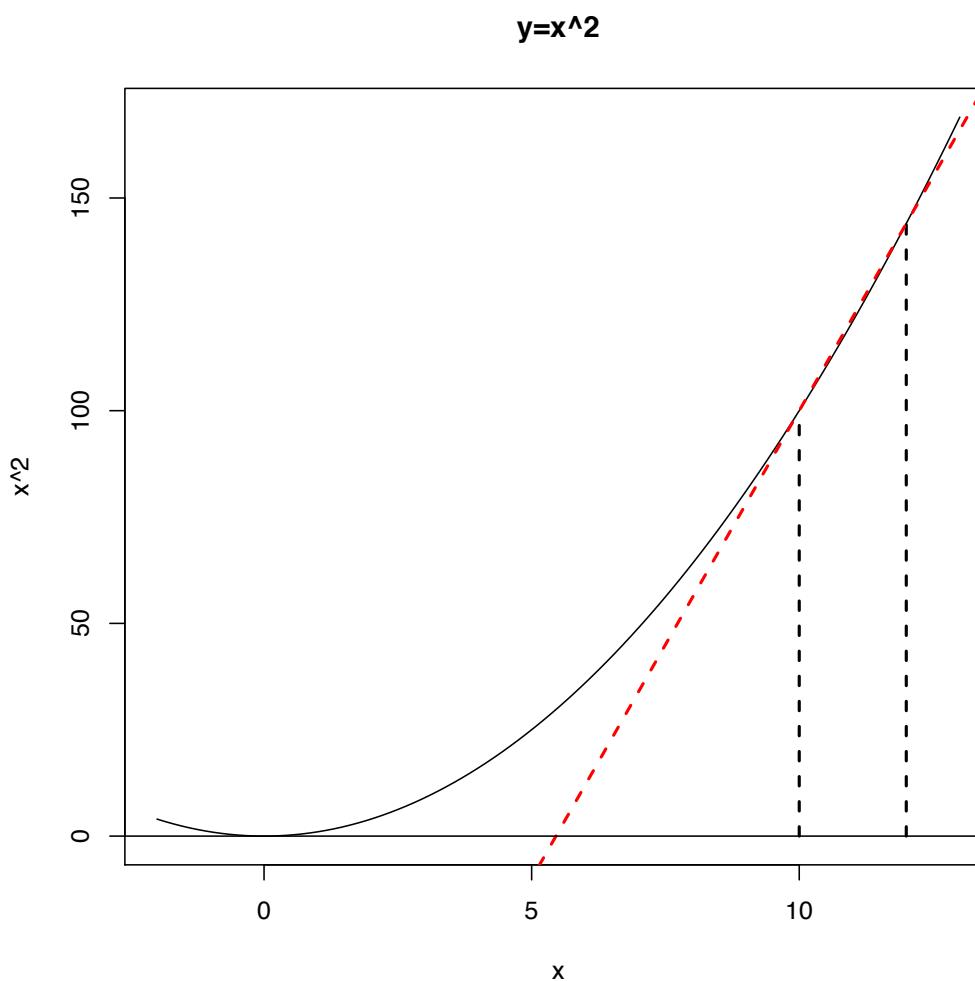
For examples from previous years, see:  
[https://github.com/PM520-Spring-2020/Week3-  
ArtShowPreviousYears](https://github.com/PM520-Spring-2020/Week3-ArtShowPreviousYears)



The One Who Bit Spiderman

# Secant method for root finding

- Newton-Raphson needs the derivative  $f'$ .
- What if  $f'$  is impossible to compute?
- Use an approximation to the derivative formed by a straight line put through the last two ‘guesses’



```
pdf("Fig5.pdf")
curve(x^2,-2,13,main="y=x^2")
segments(10,0,10,100,lty=2,col="black",lwd=2)
segments(12,0,12,144,lty=2,col="black",lwd=2)
abline(-120,22,col="red",lty=2,lwd=2)
abline(h=0)
dev.off()
```

# The Bisection method

- Previous methods don't always work.
- Bisection method is more reliable, but slower.
- Pseudo-code:

Start with  $X_L$ ,  $X_R$  such that  $f(X_L)f(X_R) < 0$

```
while ( |  $X_L - X_R$  | > tolerance){
```

```
     $X_M = (X_L + X_R)/2$ 
```

```
    if  $f(X_L)f(X_M) < 0$  then set  $X_R = X_M$ , else set  $X_L = X_M$ 
```

```
}
```

What happens if there are 3 roots between  $X_L$  and  $x_R$ ?

# Non-examinable lab task

- Implement the bisection method
- Test it on the following functions:
  - $f(x)=x^3$
  - $f(x)=x^3-2x^2+x$
  - $f(x)=\sin(x)$

# Examinable Assignment 2b

- To be turned in by 1pm, four week's from now.
- (Exercise 6, from Root-finding Chapter of Jones et al.)
  - See following slide for details.

1. Write a program to implement the Secant method.

2. Test it, using  $x_0=1$  and  $x_1=2$  on:

$$1. \cos(x)-x$$

$$2. \log(x)-\exp(-x)$$

Compare it with the performance of Newton-Raphson on the same functions.

3. Write a function to show a plot of the iterations of the algorithm.

4. Turn it in as an Rmd document.

**END**