Admin

- Lecture recordings: ZoomPro tab on Blackboard. I will record any time we don't have absolutely everyone here (which I'm assuming will be most every week).

- Solutions:
  - I will continue to upload solutions to most problems to https://github.com/PM520-Spring2022/Solutions

PM 520 - Lecture 4

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

Assignment1: Randomization tests - golf balls - due midnight 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?

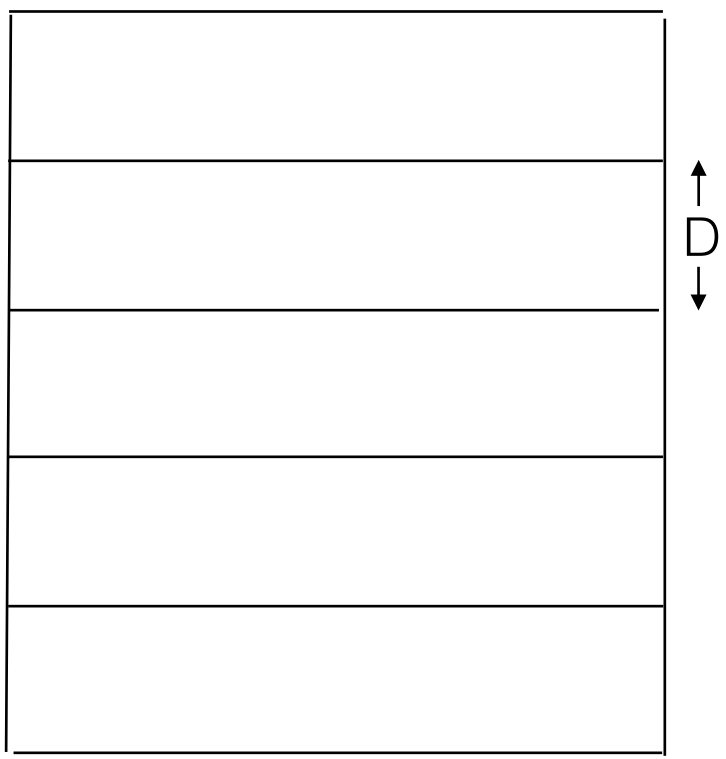
[Originally due to Allan Rossman - via Randall Pruim]

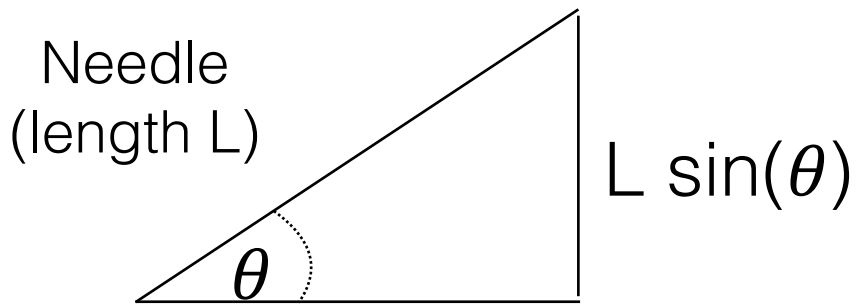
Assignment notes

- We cannot read your answers if you upload an html file (annoying fact of life with Github).
- So please knit your answers to either:
  - A pdf (ideal, but requires you to install al latex editor. I recommend tinytex)
  - A markdown (.md) file, but note that you also need to upload the folder that contains the graphics for any figures that are included.

- Thanks to those of you who have confirmed your (non-obvious) GitHub IDs. I think I know who everybody is now, but I'll confirm once we have graded the assignments.

Monte Carlo Simulation: Calculating π - Buffon's needle





So, P(cross line) = min(1,LSin(θ)/D)

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

C/N ~ $\int$ Lsin(θ)(1/π) dπ [integral goes from 0 to π in radians]

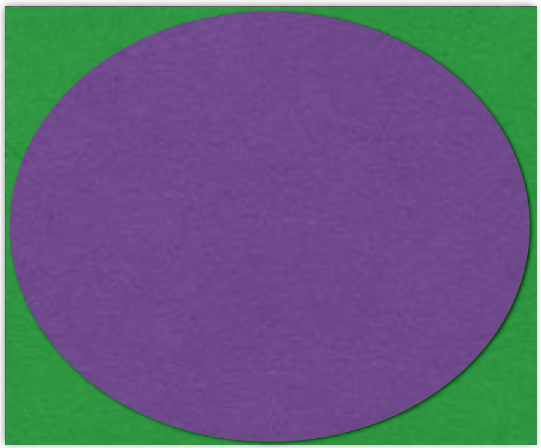
So C/N ~ L[-cos(π)-(-cos(0))]/π = 2L/π

So π ~ 2LN/C

Monte Carlo Simulation: Calculating π - 'dart board' approach

- Estimate (not calculate) π using Monte Carlo methods.

- So, you need to think of something you can simulate in which the probability of success depends upon π.



In-class exercise: Exponential task 1

- Generate 1000 Exp(λ) rvs. conditional on them each being greater than y, for some y (Try λ=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 λ=1, and compare it to 1000 Exp(1) rvs. [or superimpose the exponential density function using the command curve(λ*exp(-λ*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~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?

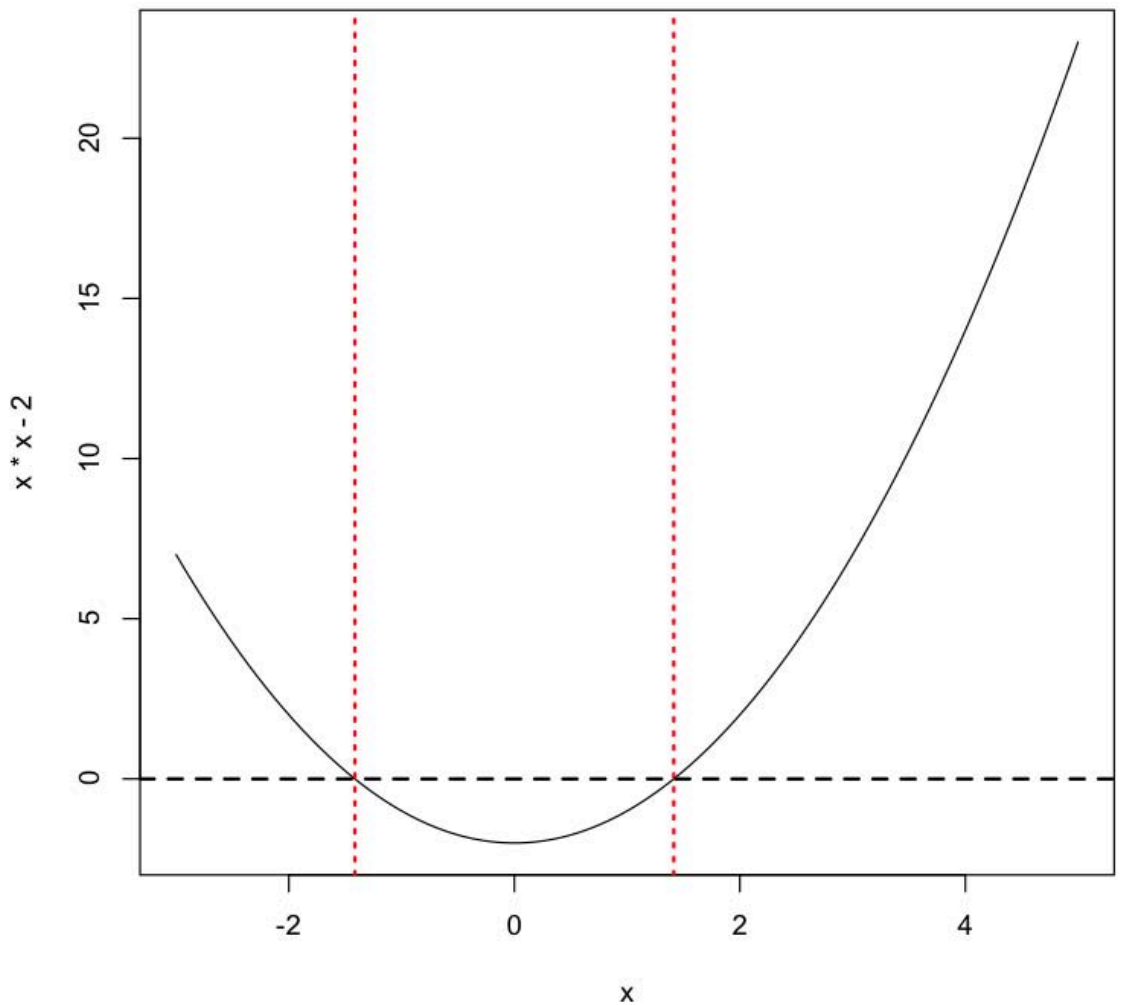
Note: the mean of an exp(λ) r.v. is 1/λ.

See 'Week2-BusWaitingTimesExercise on Github

Finding roots and fixed points

"Root-finding" Chapter 10 of Jones et al.

Finding roots and fixed points




```
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)
```

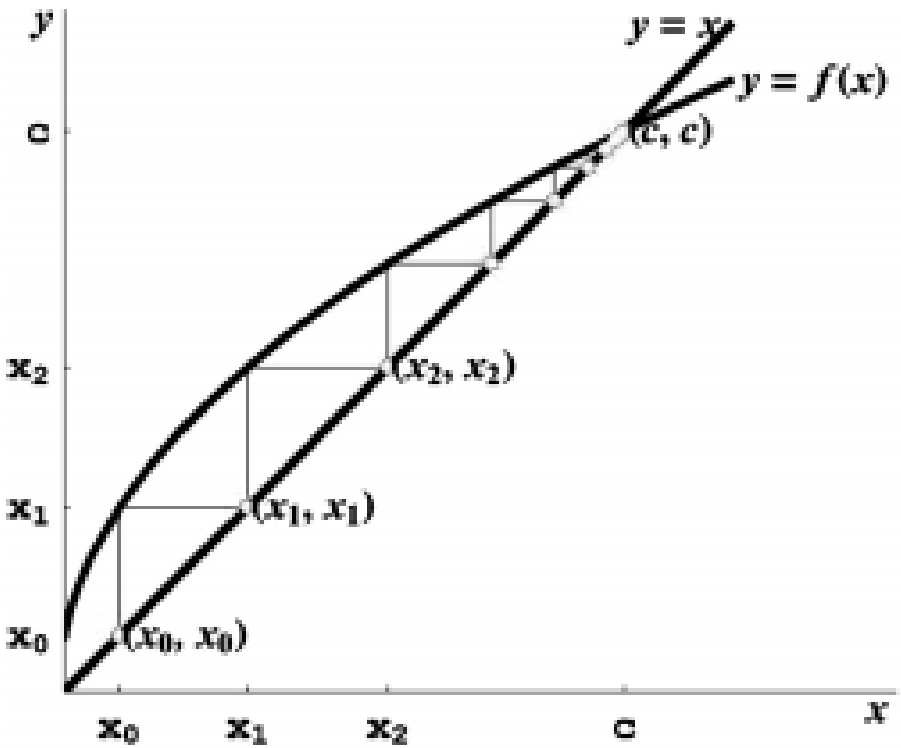
Roots are important

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).

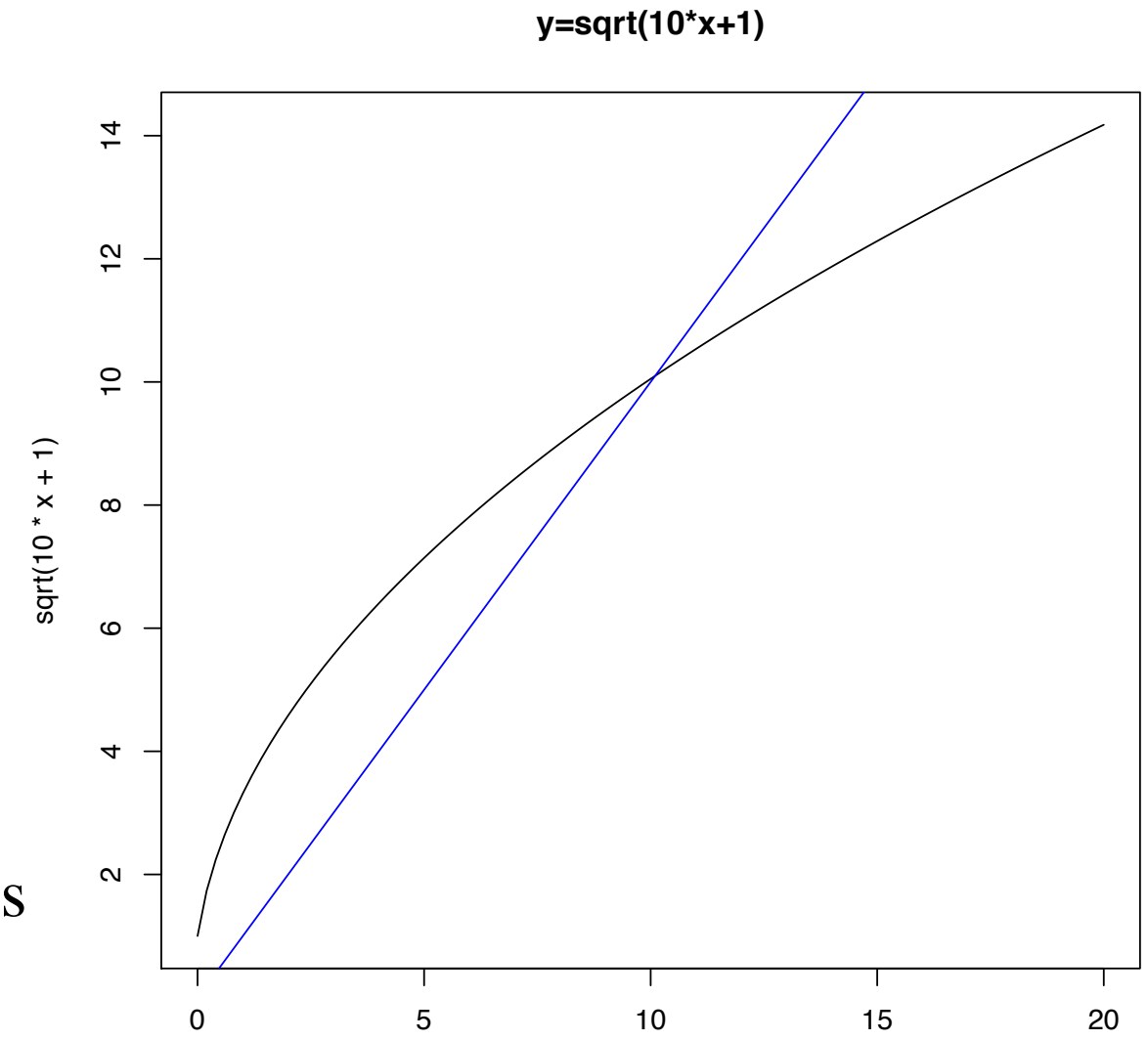
Fixed point iteration

- 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,...
  - Set $x_i = f(x_{i-1})$
- When should we stop?

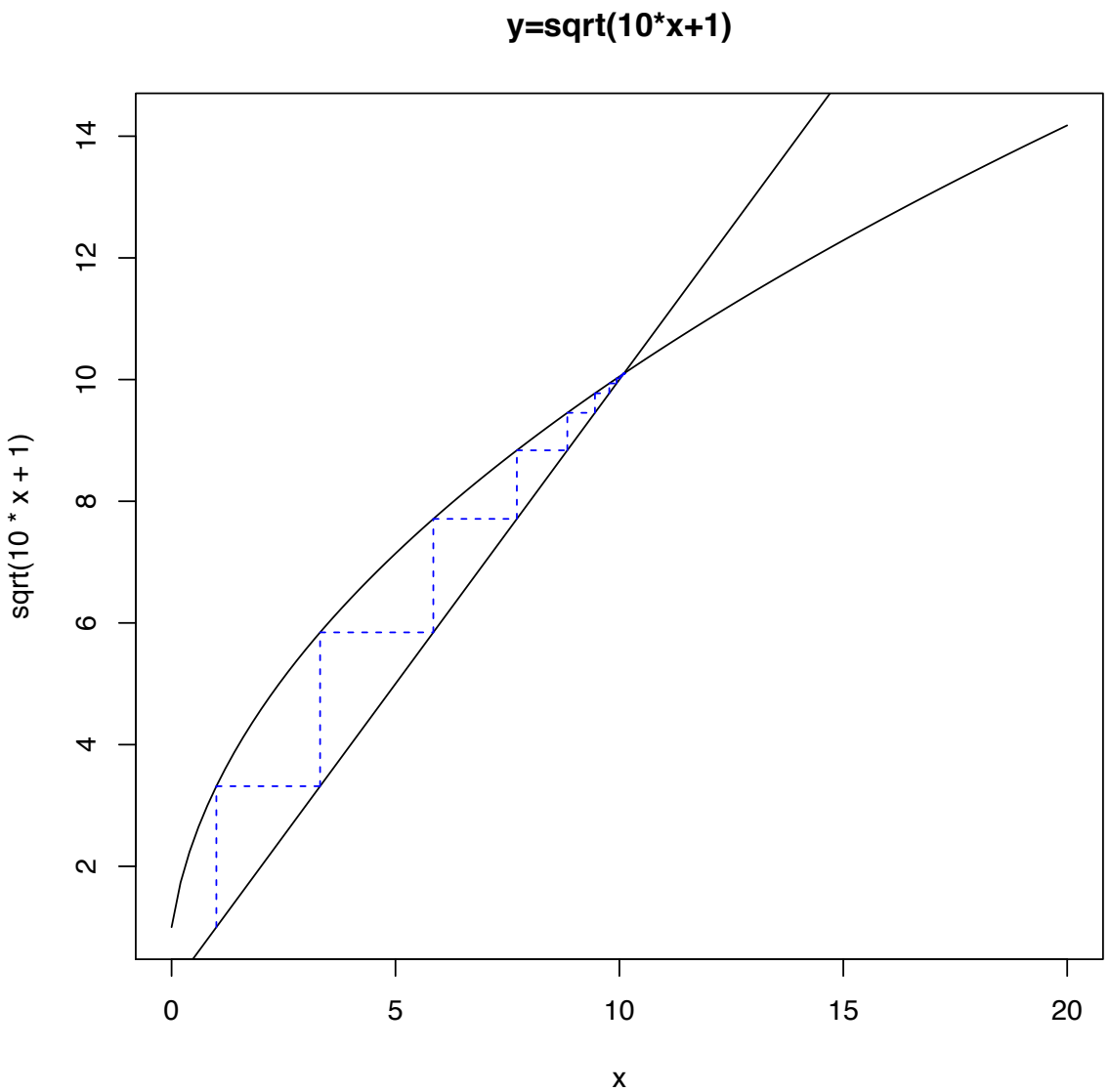
Saving plots to file



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()
```

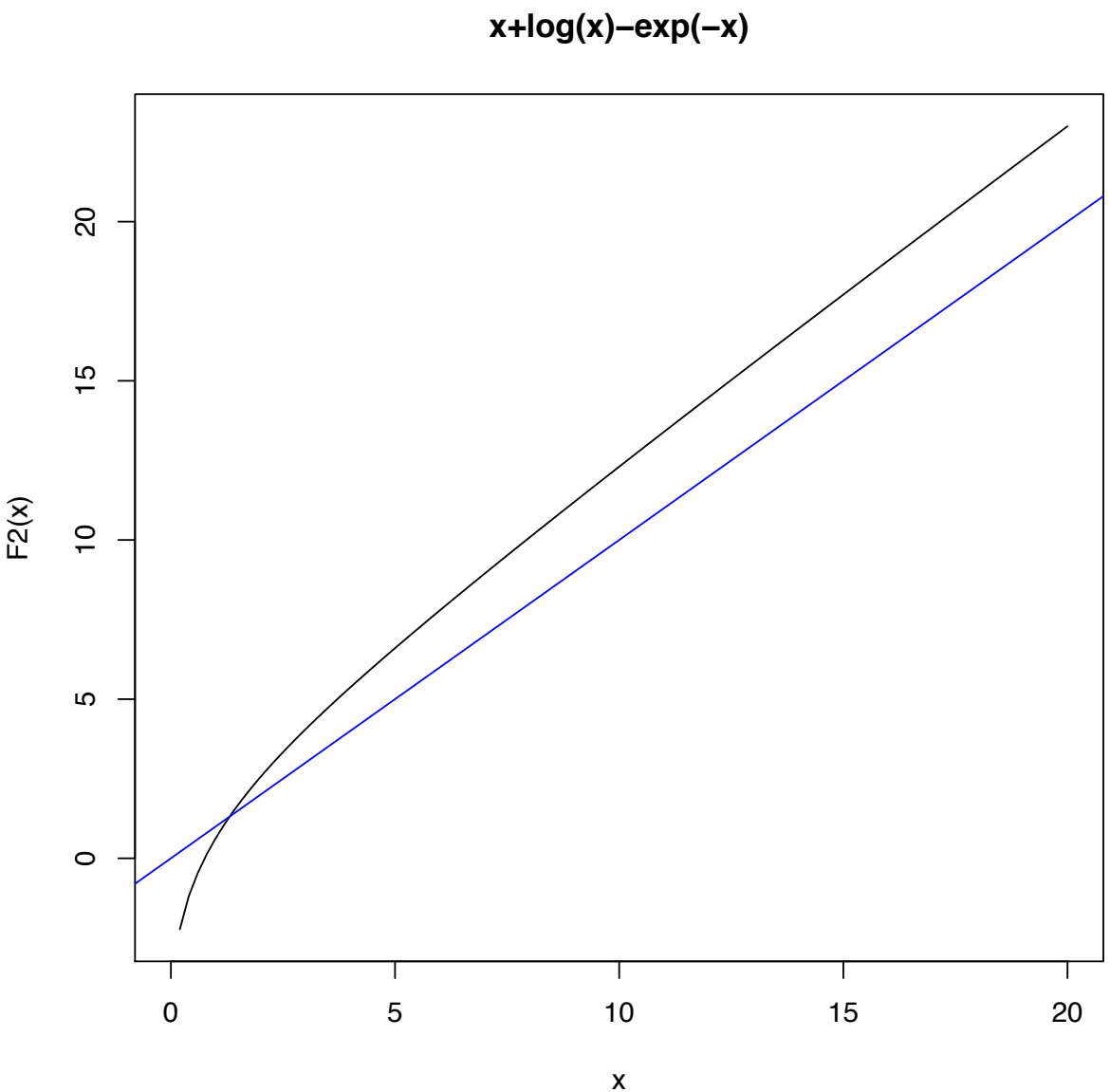
A version that plots progress



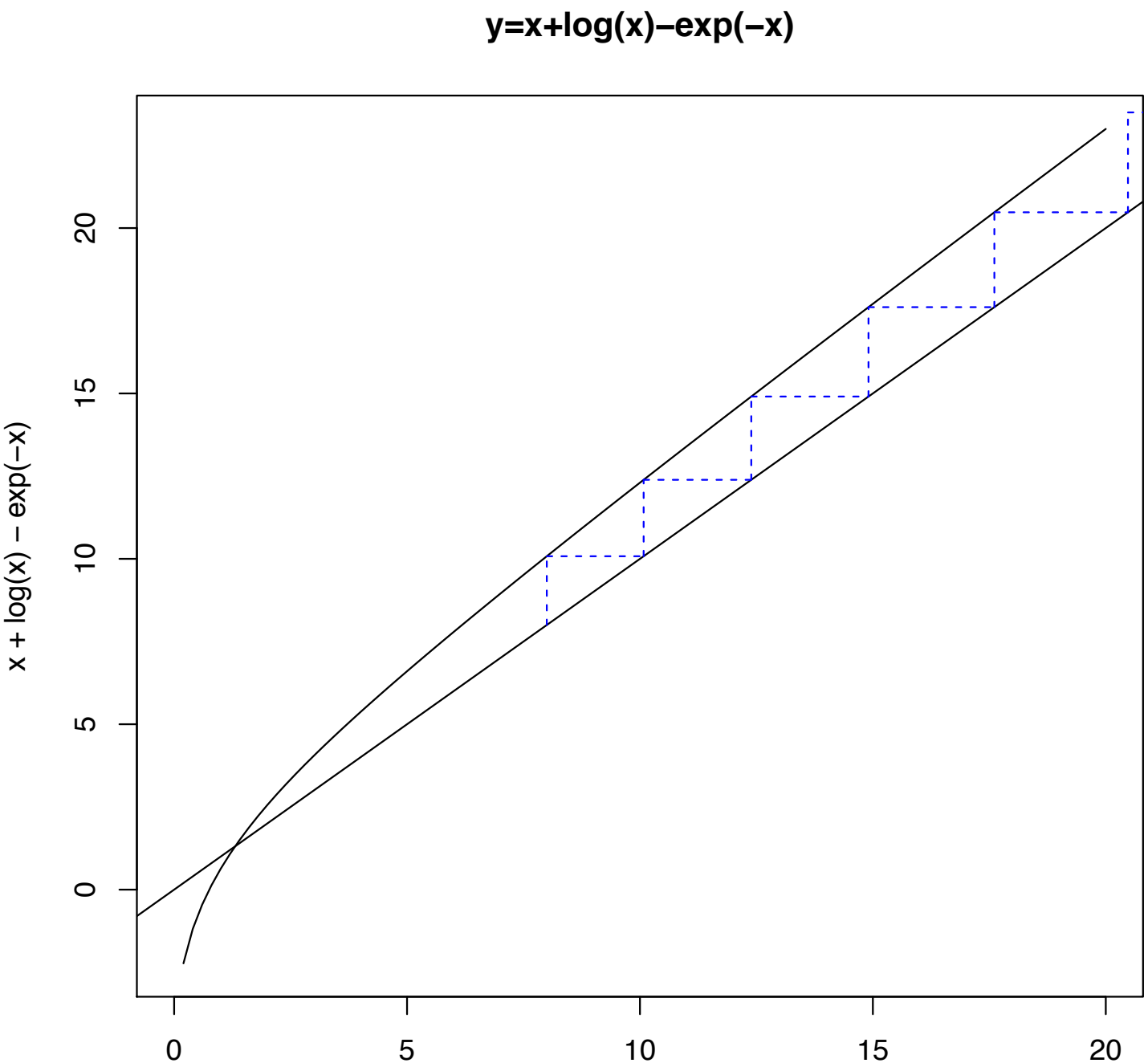

```
segments(X,X,X,Xprime,col="blue",lty=2)
```

On Github in “FixedPoints” repo

Let's try a tougher function



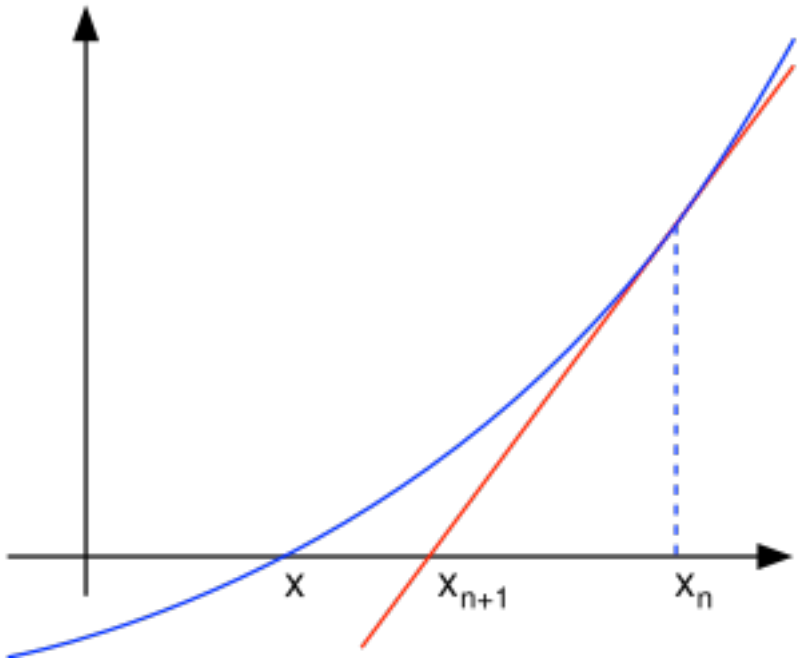

```
F2<-function(x){
        return(x+log(x)-exp(-x))
}
```


y=x+log(x)−exp(−x)
x + log(x) − exp(−x)
0
5
10
15
20
0
5
10
15
20

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

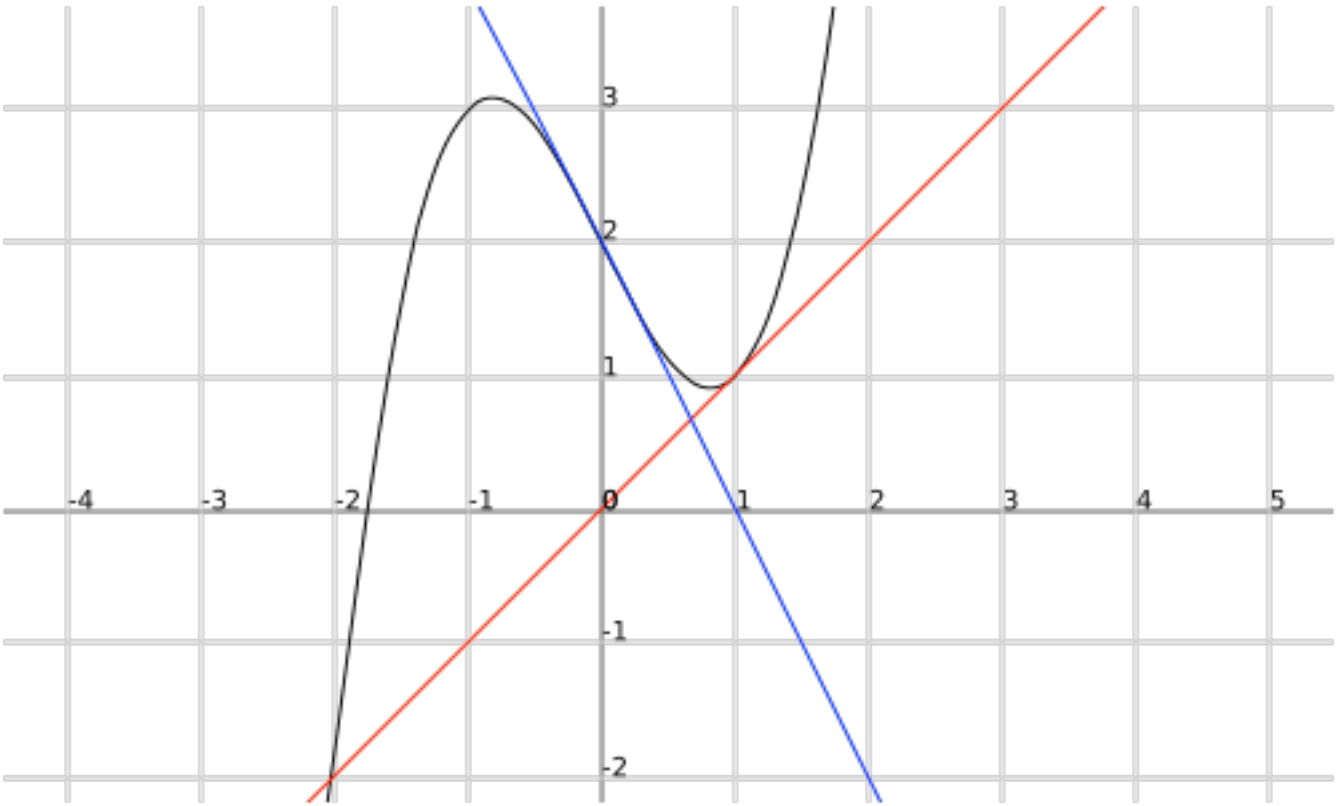


An illustration of one iteration of Newton's method (the function *f* is shown in blue and the tangent line is in red). We see that $xn+1$ is a better approximation than xn for the root x of the function f.

Informally

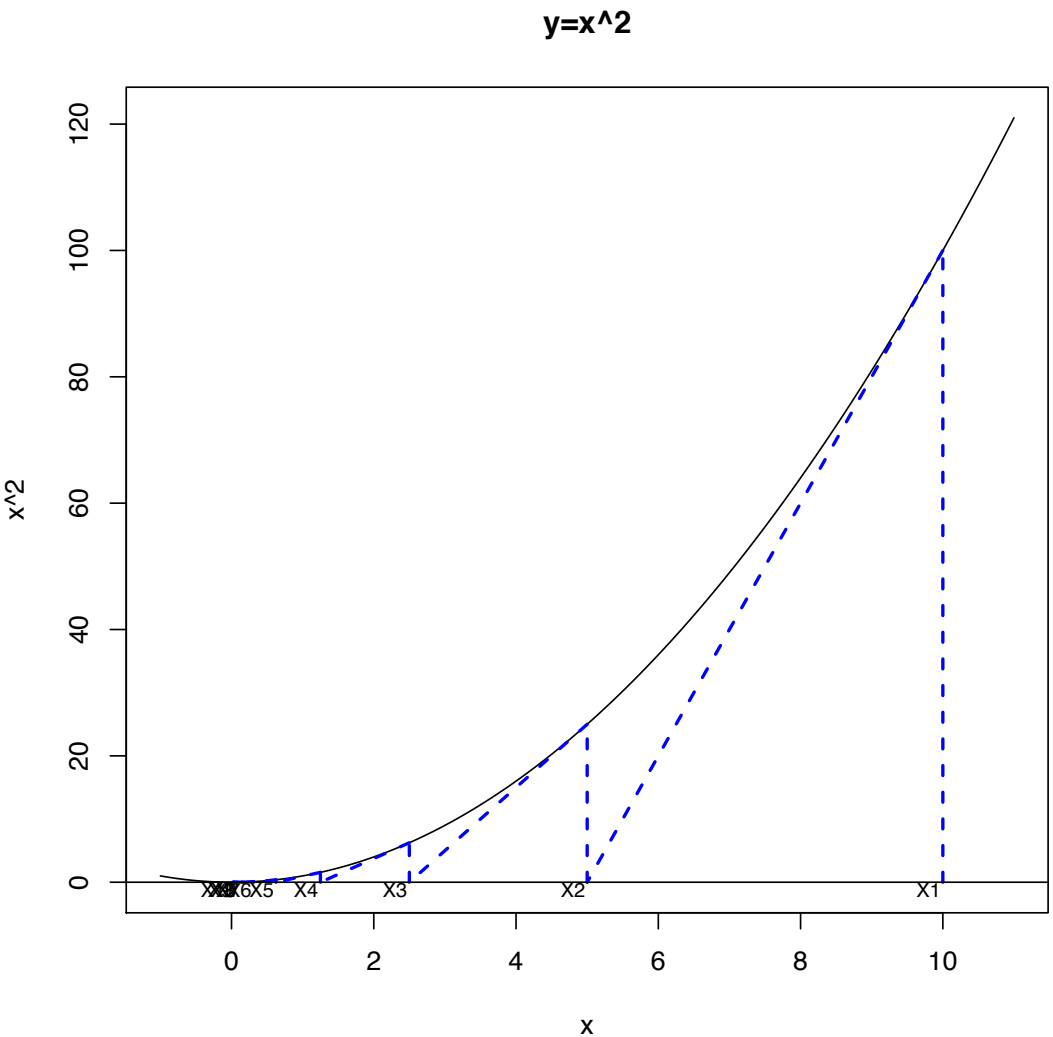
- If we are at x_i
  - Take the tangent to f(x) at $x = x_i$
  - Find where the tangent intercepts the x-axis. Set this to be x_{i+1}
  - Iterate

Which is not to say that it always works…



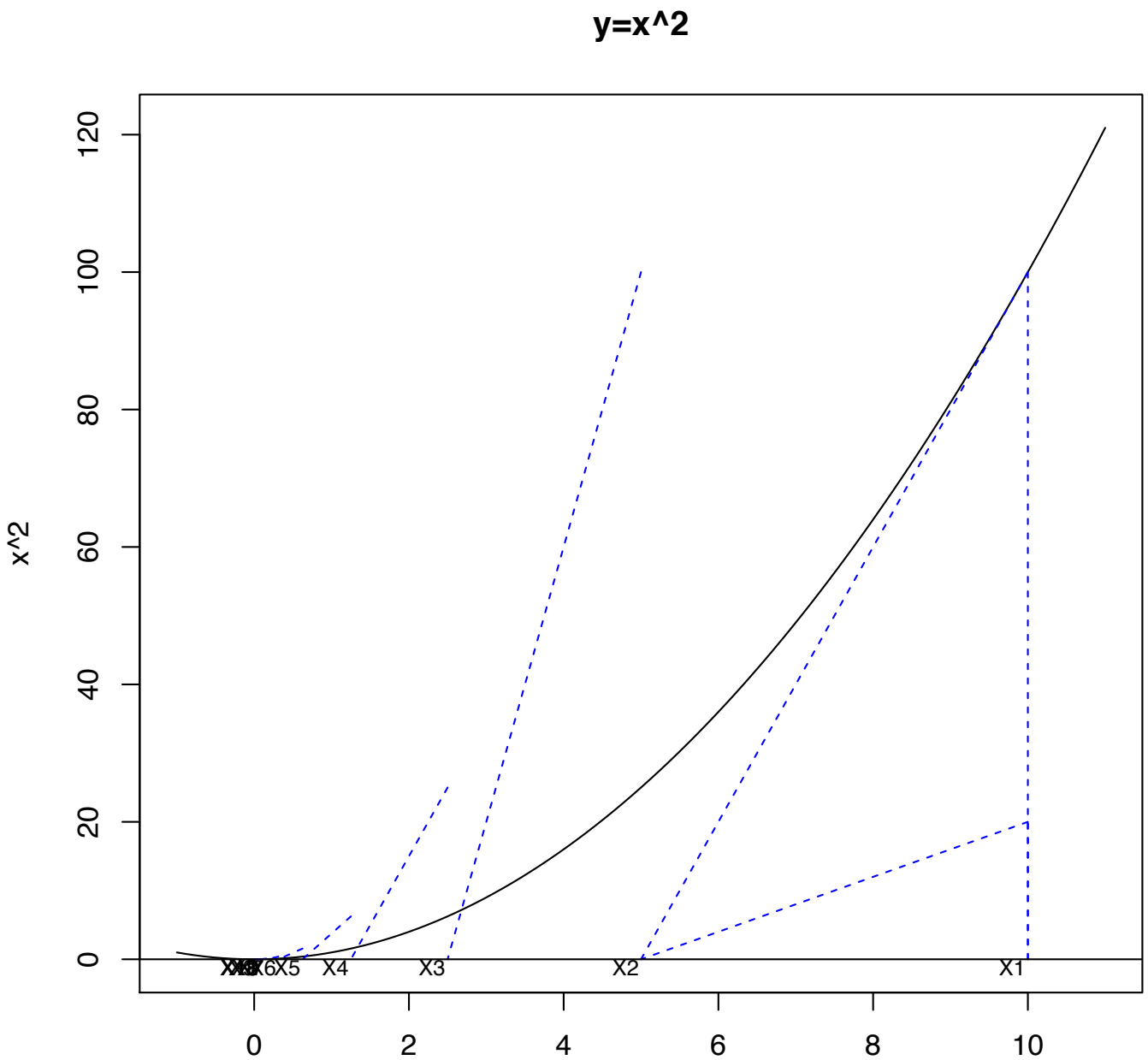
The tangent lines of $x^3 - 2x + 2$ at 0 and 1 intersect the x-axis at 1 and 0 respectively, illustrating why Newton's method oscillates between these values for some starting points.

Example




```
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
```

My first attempt looked like this…



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,MaxNumberOfIterations){
    #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<MaxNumberOfIterations)&&(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 it's 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()
```

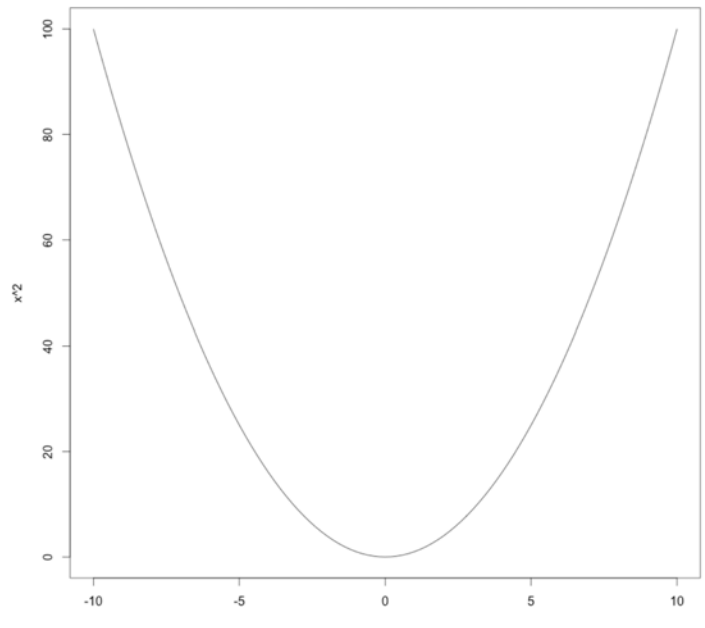
‘Week4-NewtonRaphson’ on Github

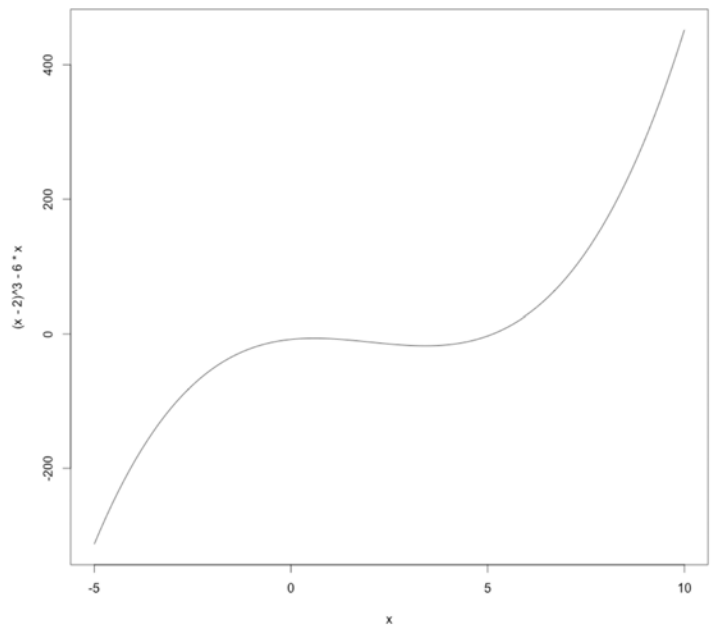
Lab task (~30 minutes)

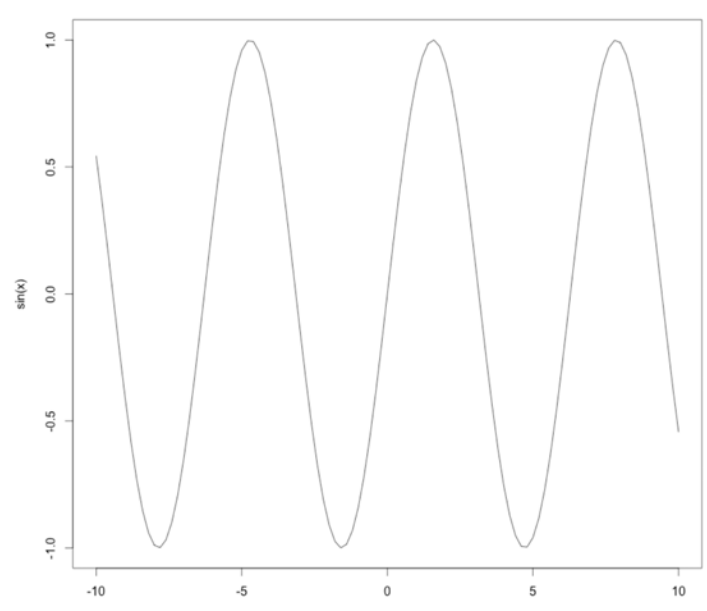
- Implement Newton-Raphson in R

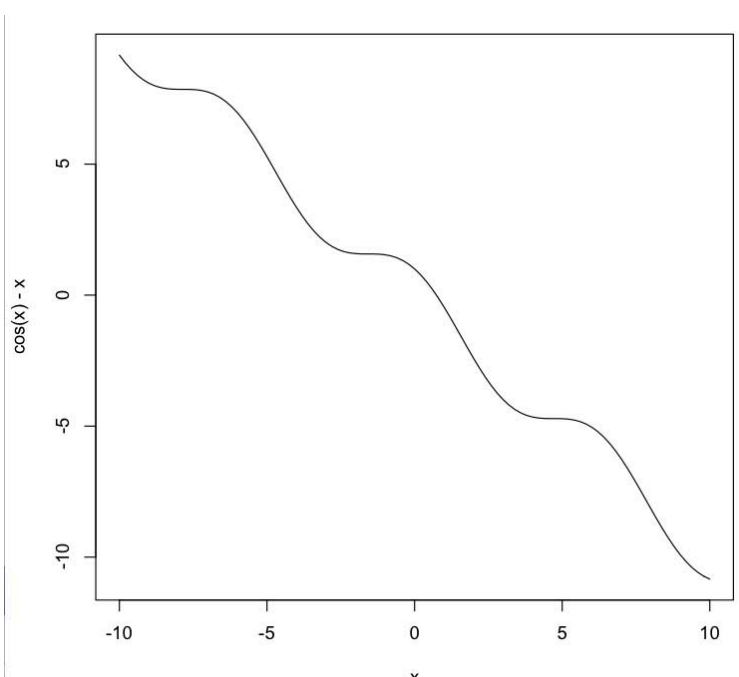
- Test it on the following functions:
  - $f(x)=x^2$
  - $f(x)=(x-2)^3-6x$
  - $f(x)=\sin(x)$
  - $f(x)=\cos(x)-x$

- 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.


x^2
x
0
20
40
60
80
100
-10
-5
0
5
10

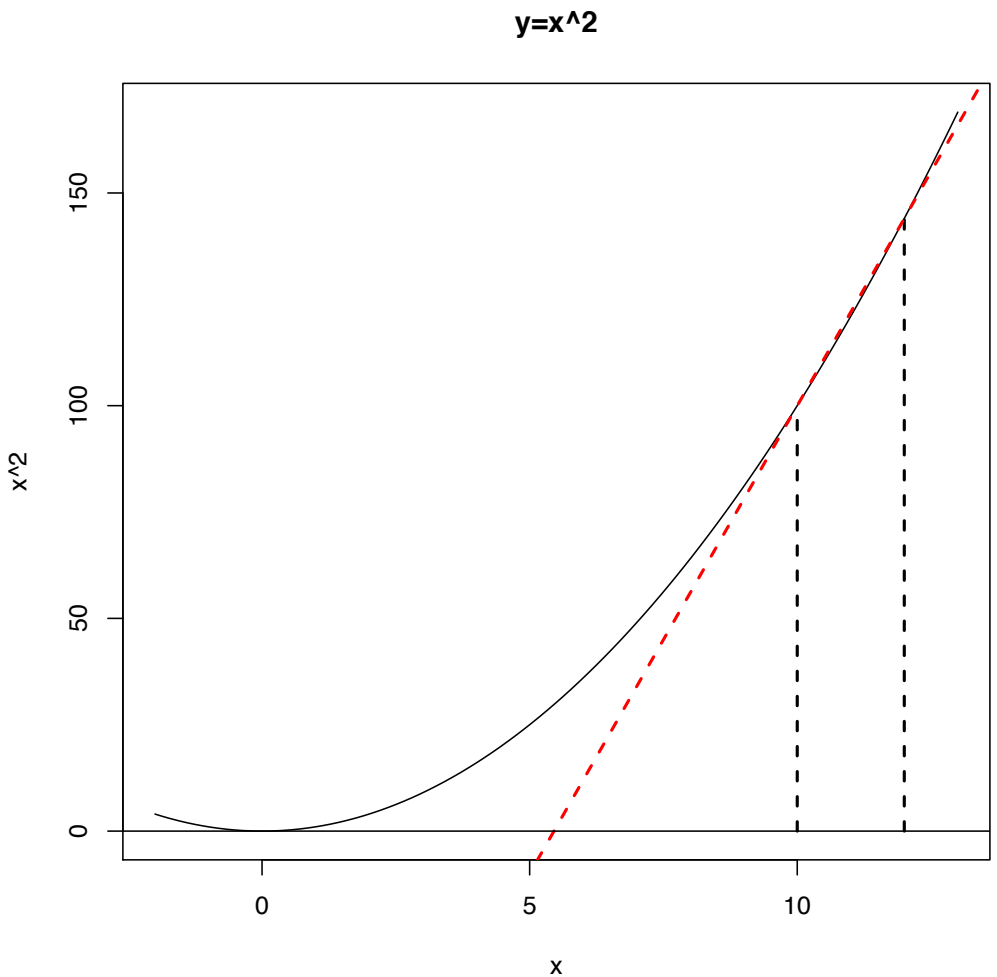

(x - 2)^3 - 6 * x
x
-200
0
200
400
-5
0
5
10


sin(x)
x
-1.0
-0.5
0.0
0.5
1.0
-10
-5
0
5
10


cos(x) - x
x
-10
-5
0
5
-10
-5
0
5
10

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()
```

Examinable Assignment 2a

- To be turned in by **midnight**, THREE week's from now (i.e. Friday Feb. 25th).
- (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, along with a compiled pdf or .md file (along with figures folder).

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?

Optional non-examinable lab task

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

Netwon-Raphson (continued)

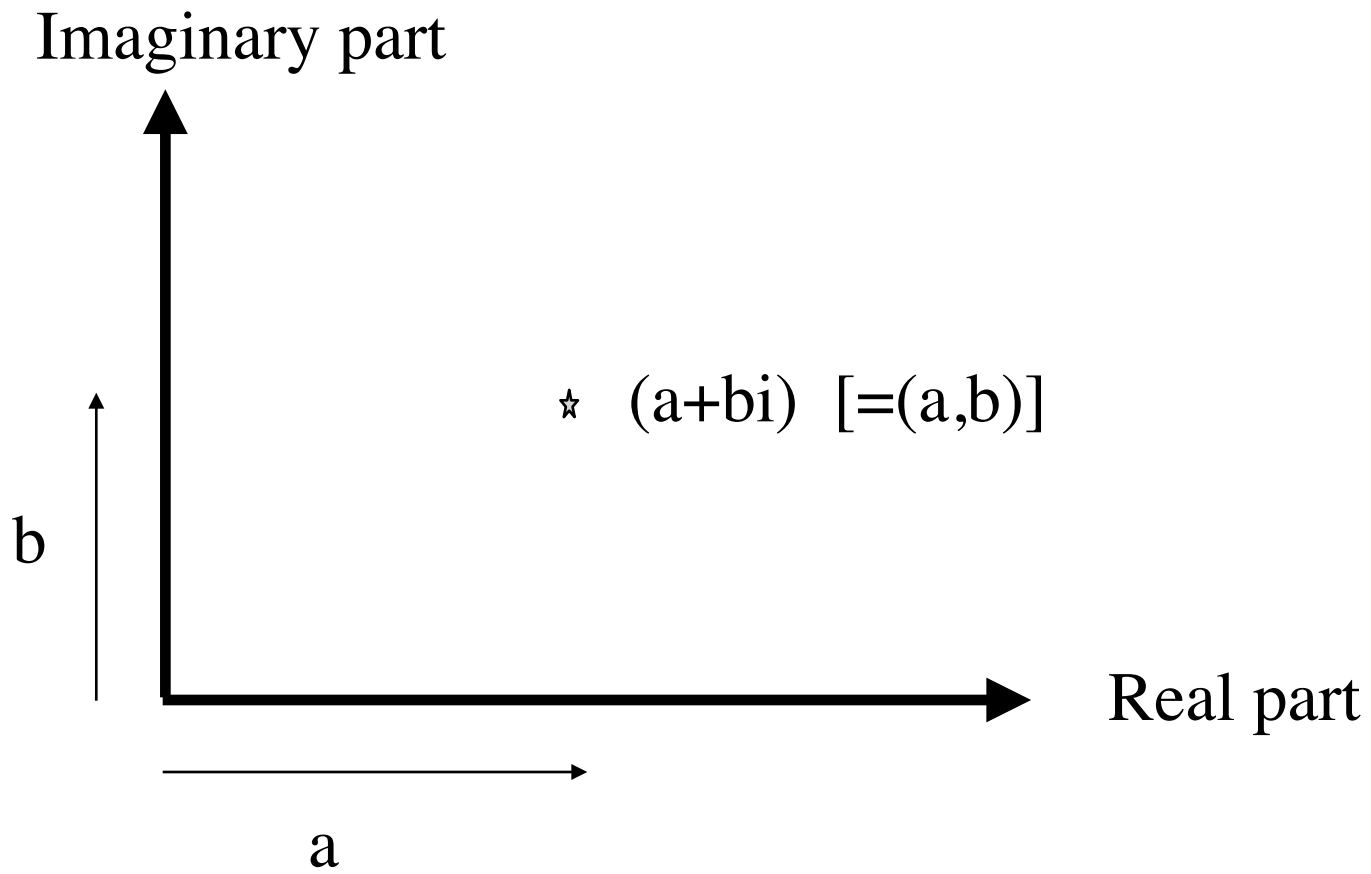
- We have seen Newton-Raphson on 1-dimensional space
- It works on higher-dimensional space.
  - e.g. Complex numbers

- 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:
-



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 work in much the same way as the derivatives you are familiar with.

- Try the function z^3-1 (It has three roots).
- Try z^4-1.

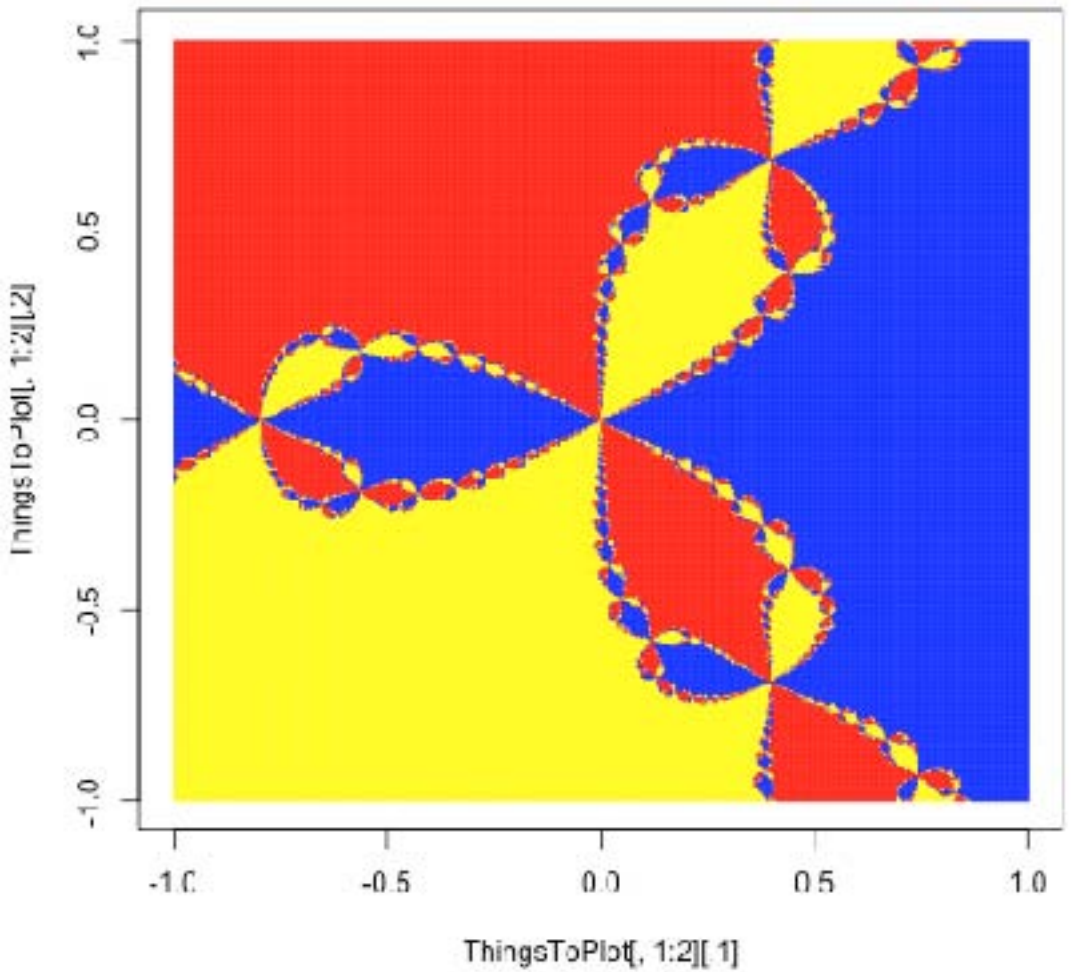
-

Newton-Raphson task: Newton-Raphson on complex plane

- Construct a plot in which you sample start values for Newton-Raphson on a 2D grid. eg., let x go from -1 to +1, and y go from -1 to +1
- Color each start point by either:
  - **A. How many iterations are needed to reach a root starting from that point.**
  - B. Which root you reach from that start point.

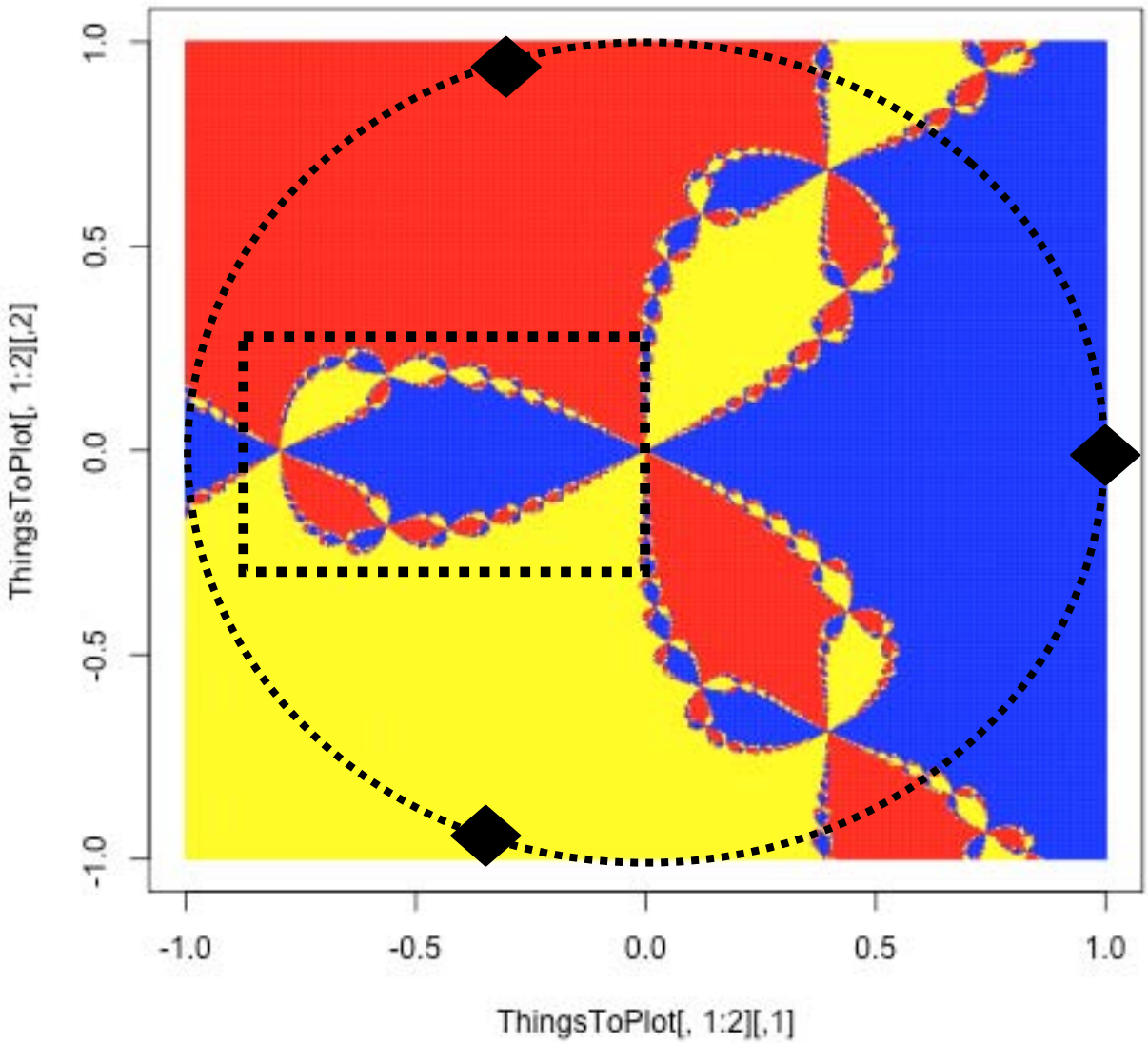
- What does the resulting plot look like?
- R has built-in support of complex numbers:
  - *complex(length.out = 0, real = numeric(), imaginary = numeric(),modulus = 1, argument = 0)*
  - *> x<-complex(1,3,4)*
  - *> y<-complex(1,3,-4)*
  - *> x+y*
  - *[1] 6+0i*
  - *> x*y*
  - *[1] 25+0i*

Example: $F(z)=z^3-1$, coloring according to the root you reach

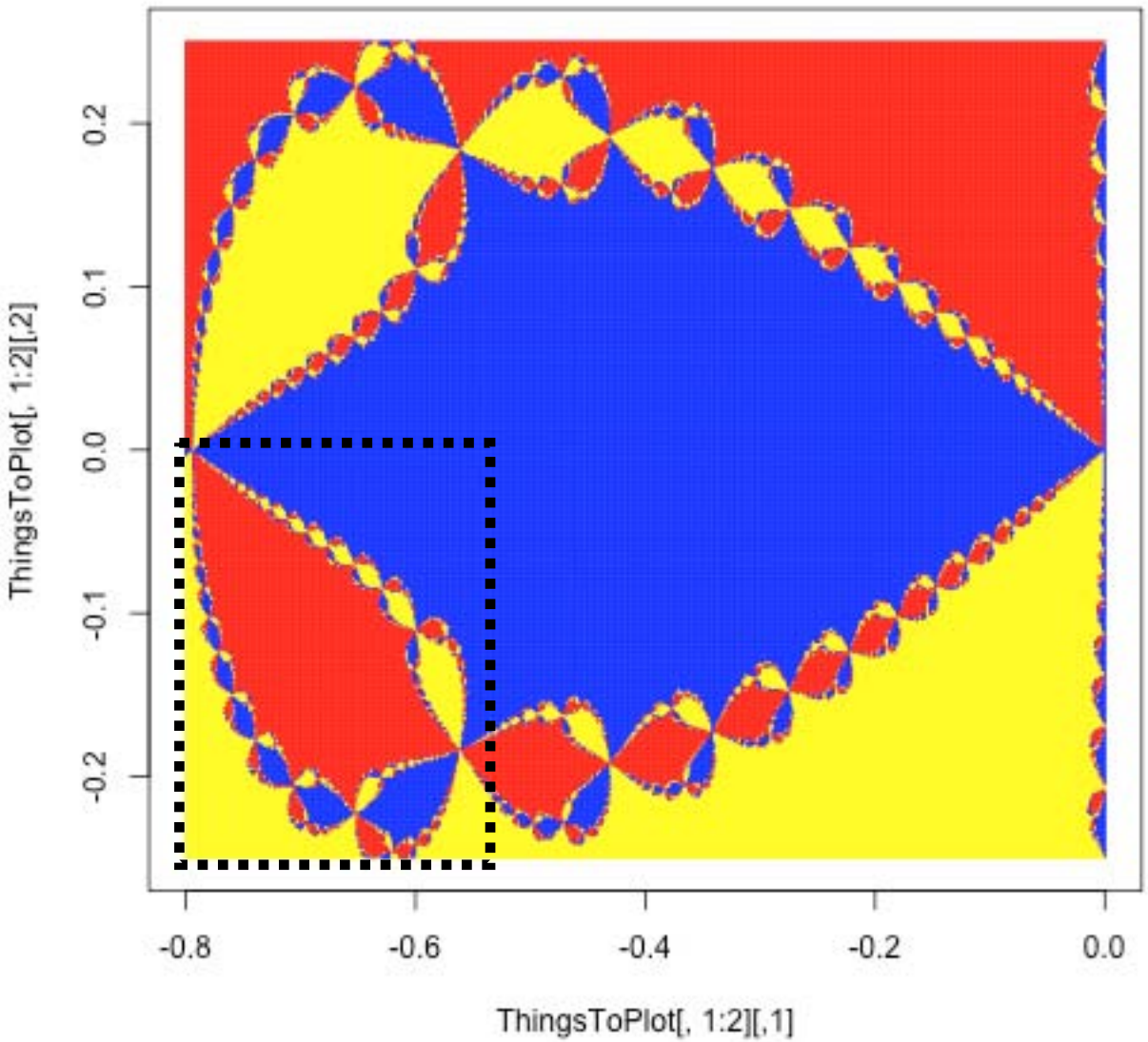



```
png("FigFrac1.png")
A<-RootPlotter(F6,-1,1,500,-1,1,500,0.2)
dev.off()
```

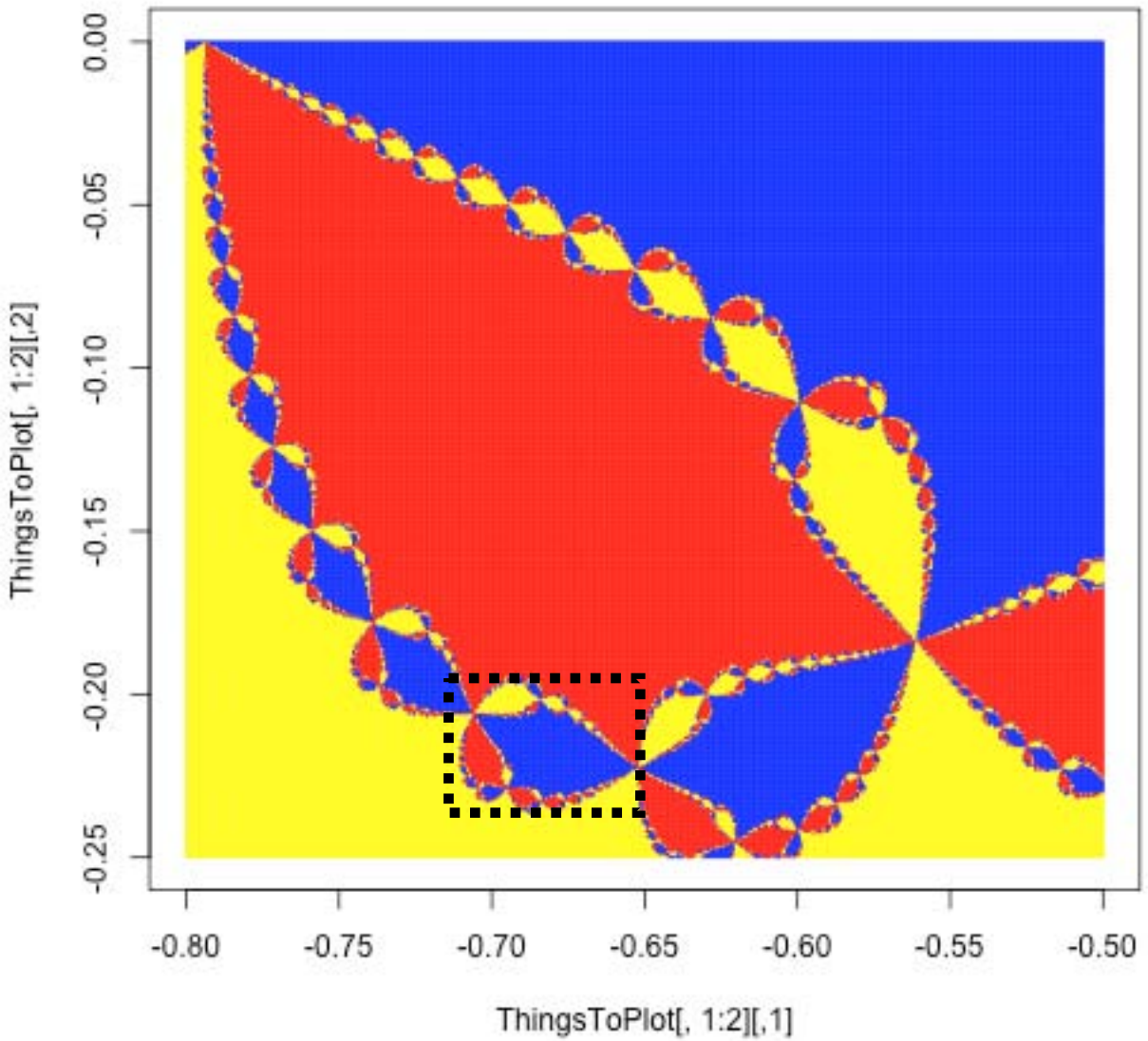
$F(z)=z^3-1$



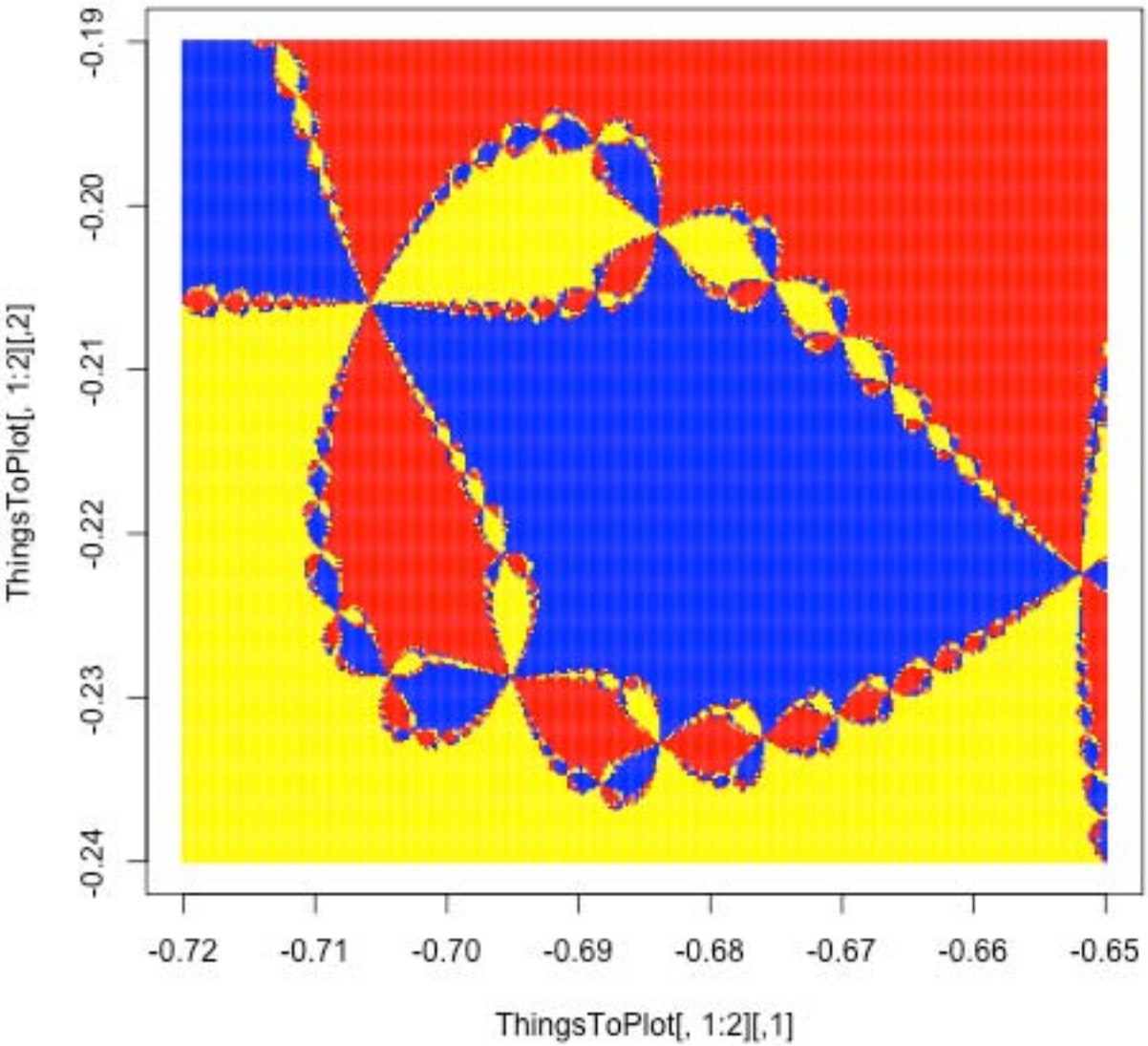

```
png("FigFrac1.png")
A<-RootPlotter(F6,-1,1,500,-1,1,500,0.2)
dev.off()
```




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

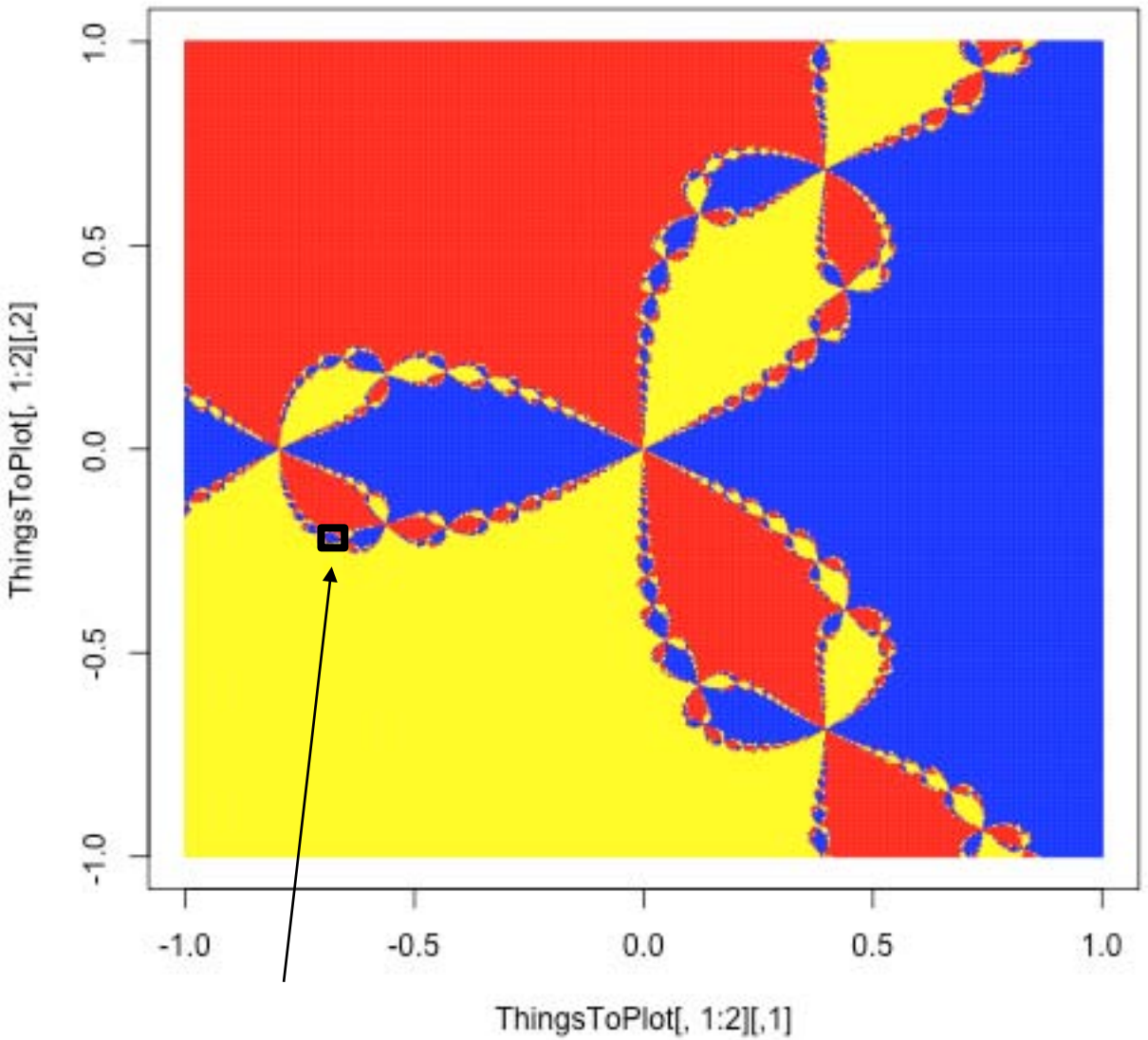



```
png("FigFrac3.png")
A<-RootPlotter(F6,-0.8,-0.5,500,-0.25,0,500,0.2)
dev.off()
```




```
png("FigFrac5.png")
A<-RootPlotter(F6,-0.72,-0.65,300,-0.24,-0.19,300,0.3)
dev.off()
```

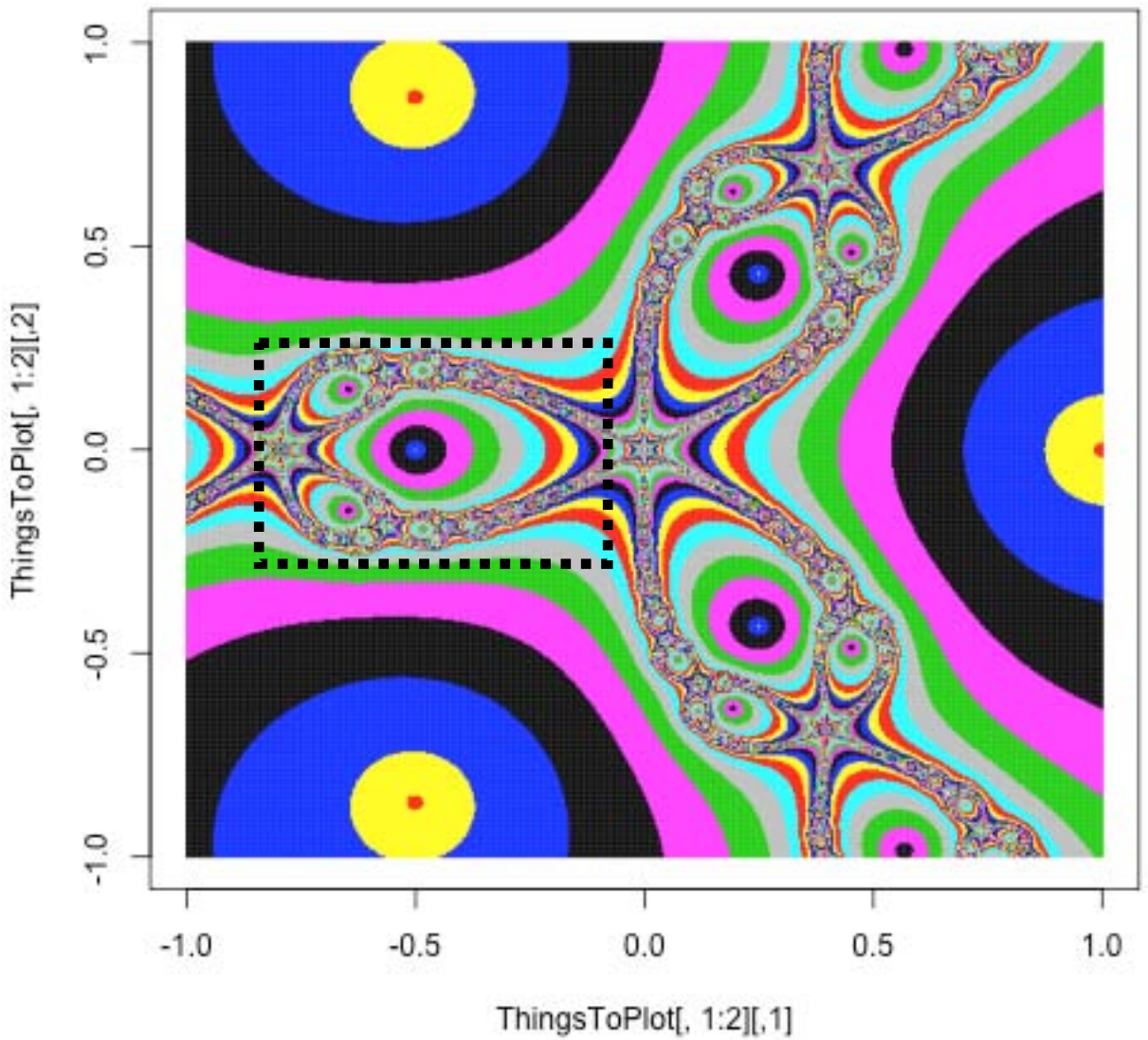
$F(z)=z^3-1$

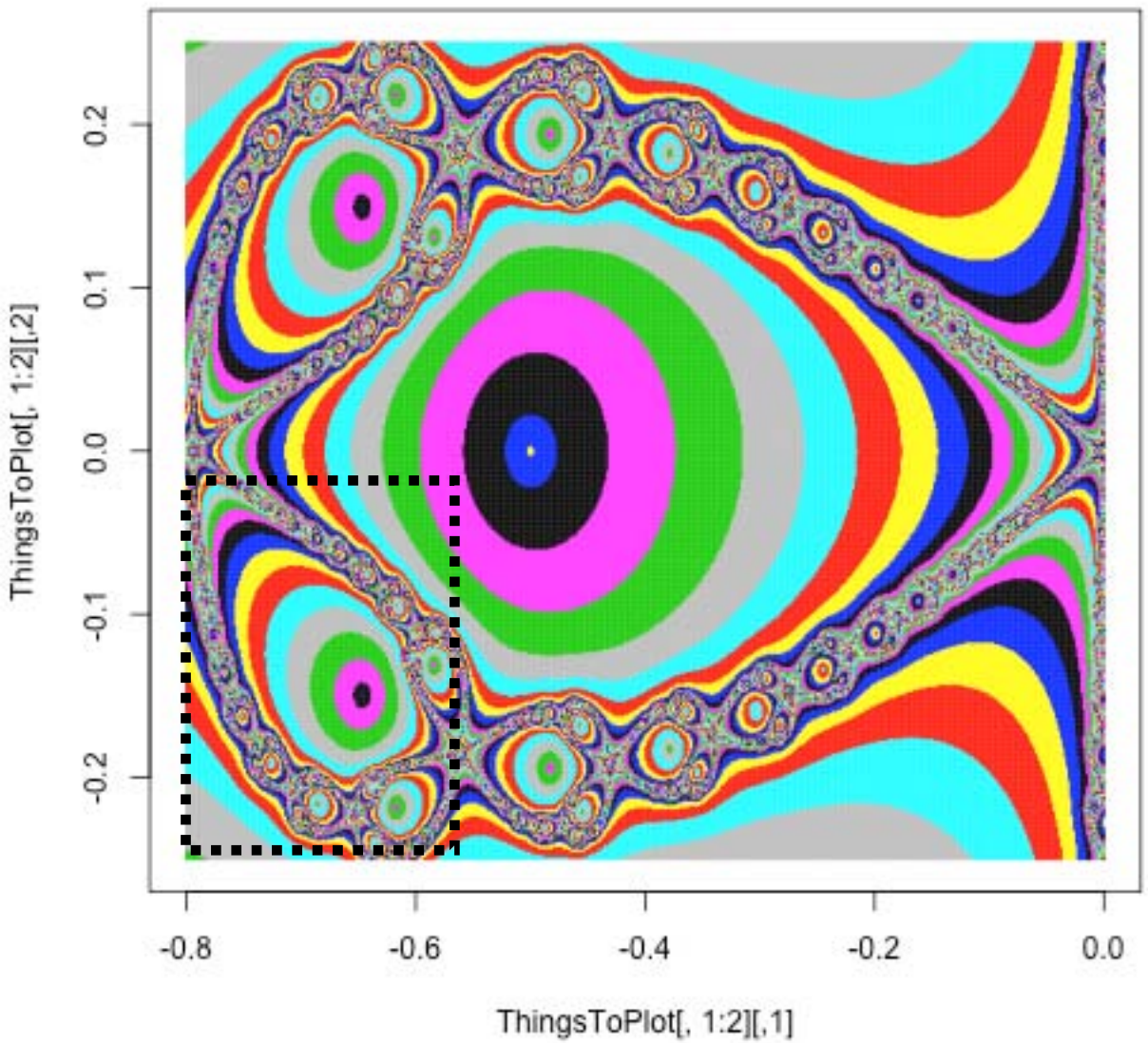


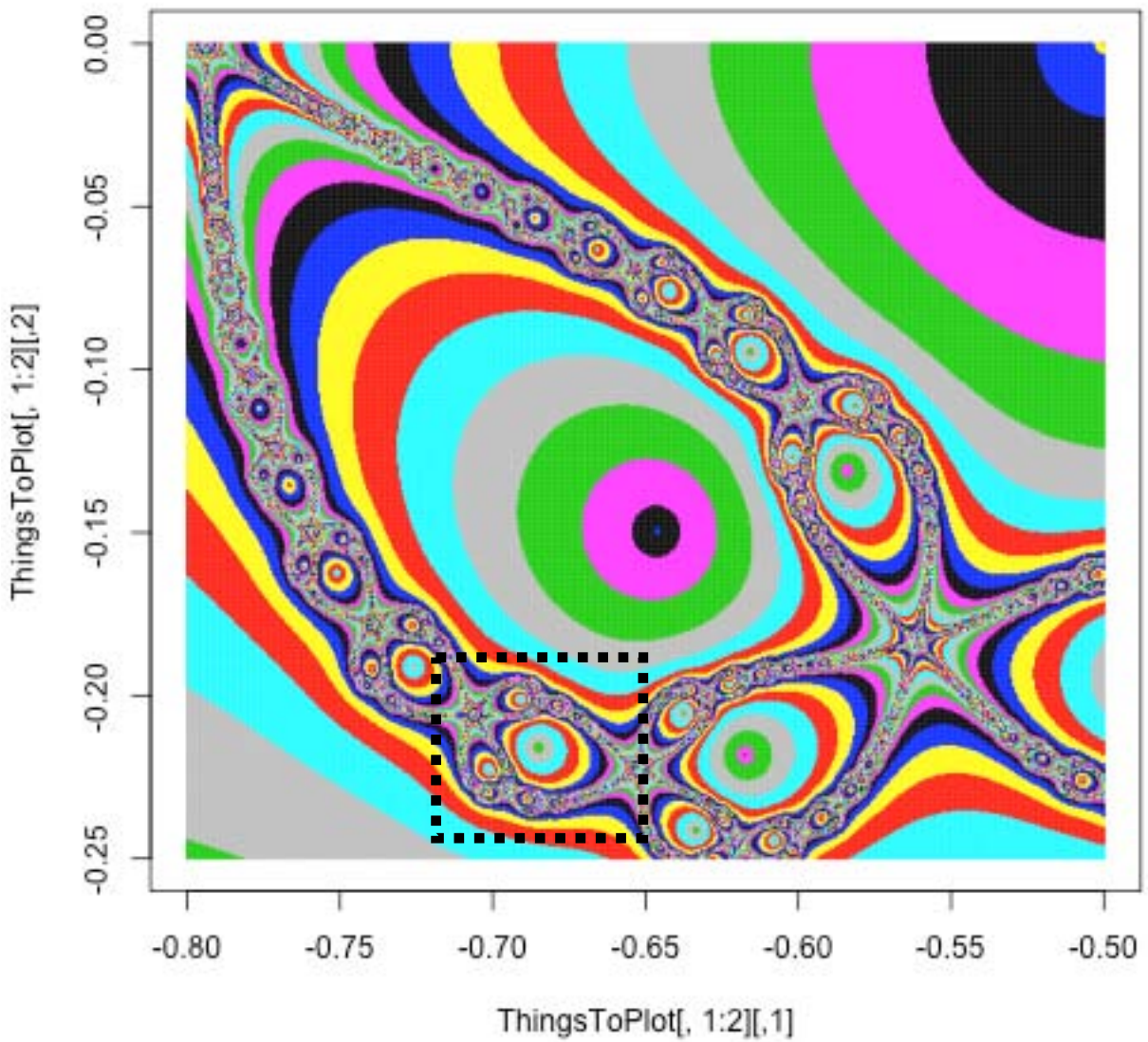

```
png("FigFrac1.png")
A<-RootPlotter(F6,-1,1,500,-1,1,500,0.2)
dev.off()
```

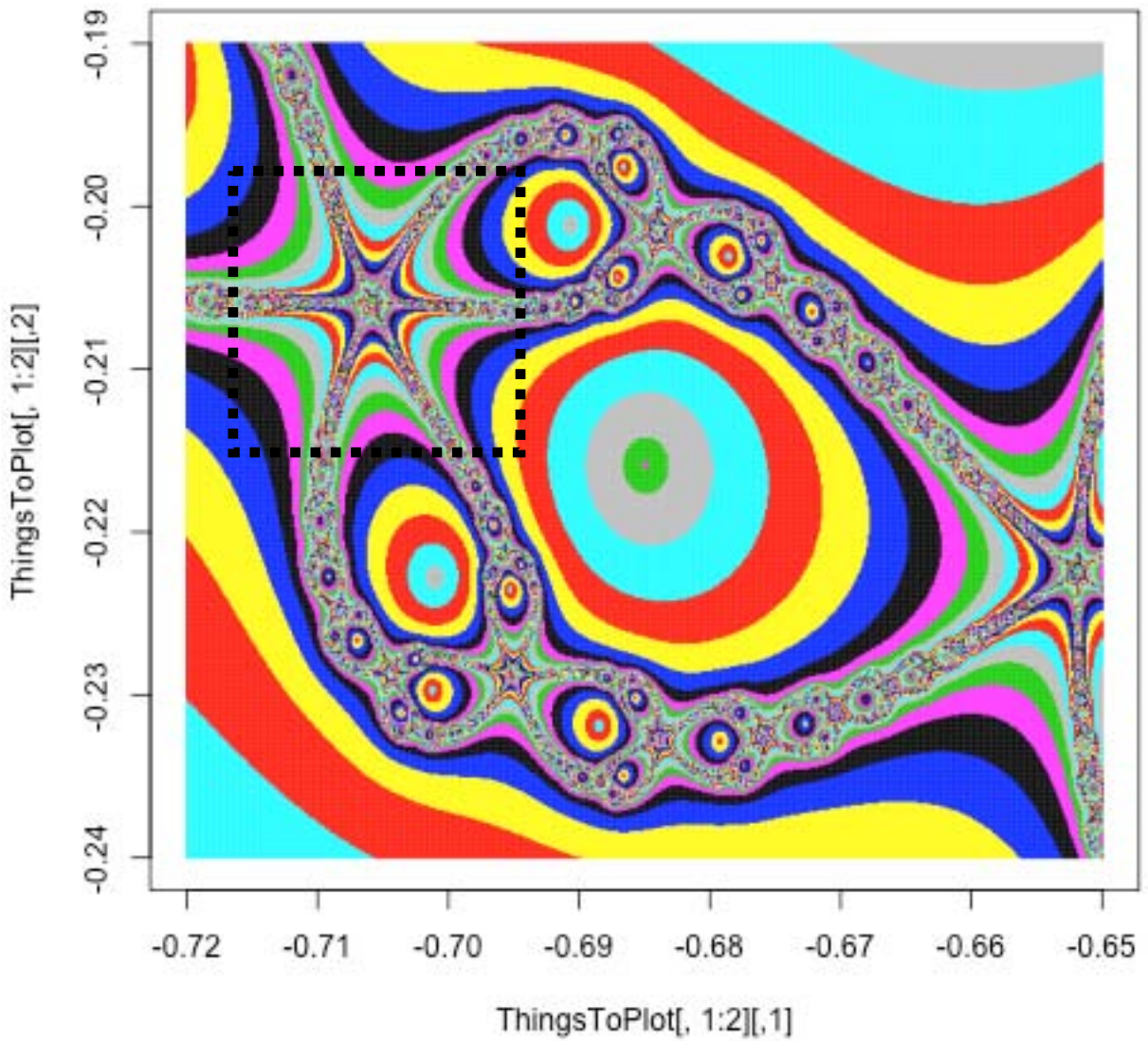
Example: Color by “number of iterations needed”

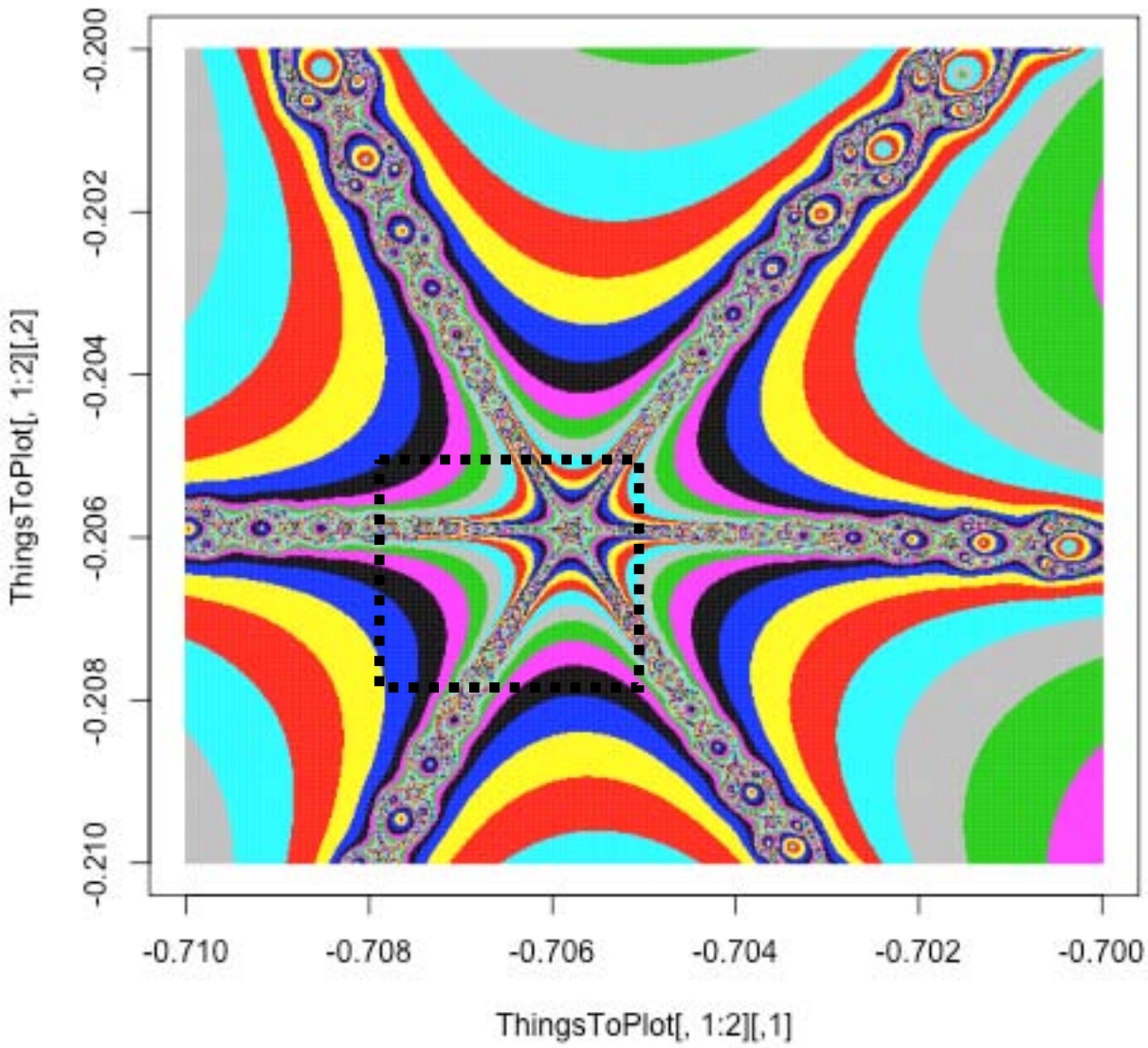
$F(z)=z^3-1$

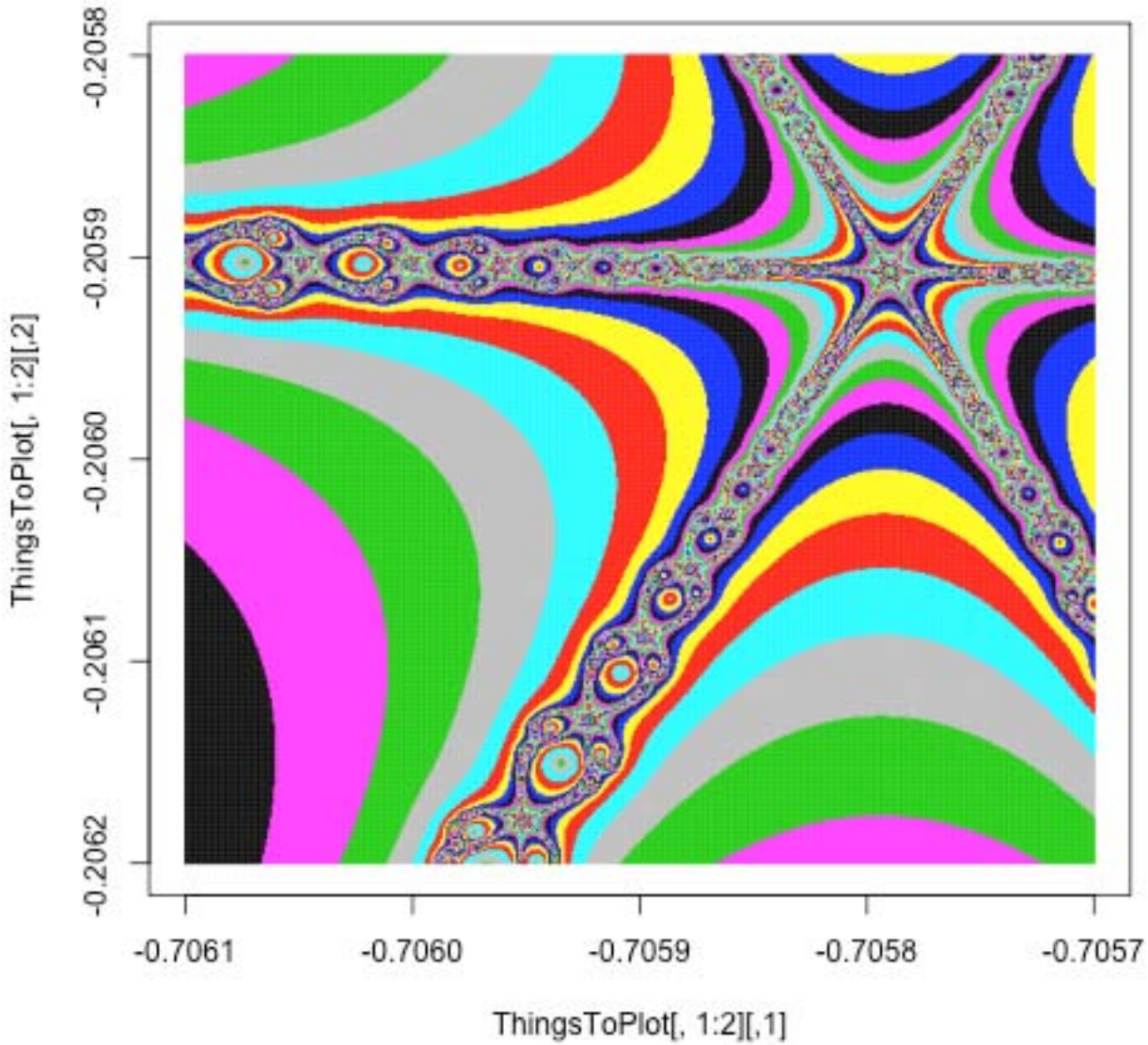



ThingsToPlot[, 1:2][,2]
0.2
0.1
0.0
-0.1
-0.2
-0.8
-0.6
-0.4
-0.2
0.0
ThingsToPlot[, 1:2][,1]

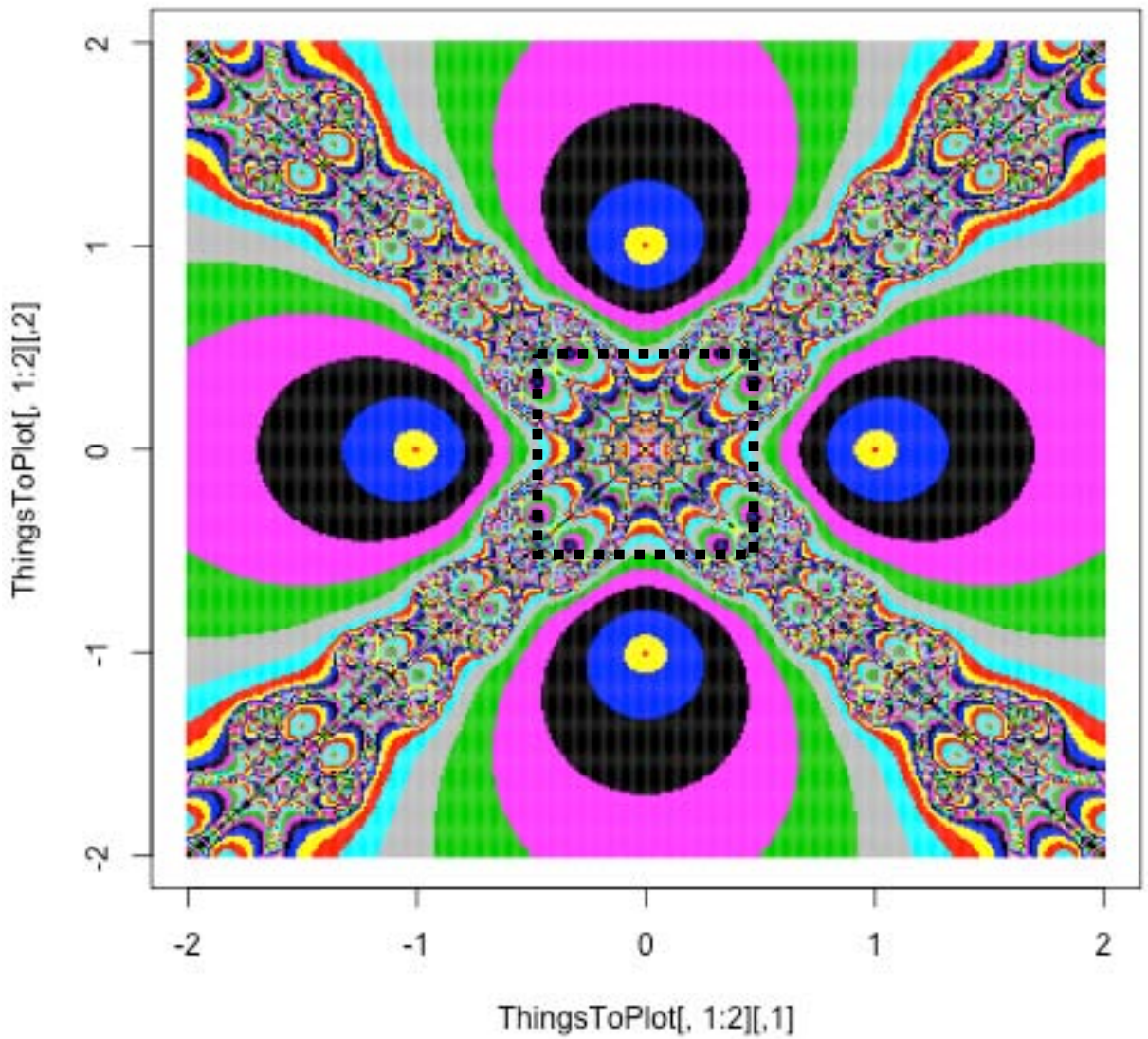

ThingsToPlot[, 1:2][,2]
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
-0.80
-0.75
-0.70
-0.65
-0.60
-0.55
-0.50
ThingsToPlot[, 1:2][,1]


-0.19
-0.20
-0.21
-0.22
-0.23
-0.24
ThingsToPlot[, 1:2][,2]
-0.72
-0.71
-0.70
-0.69
-0.68
-0.67
-0.66
-0.65
ThingsToPlot[, 1:2][,1]

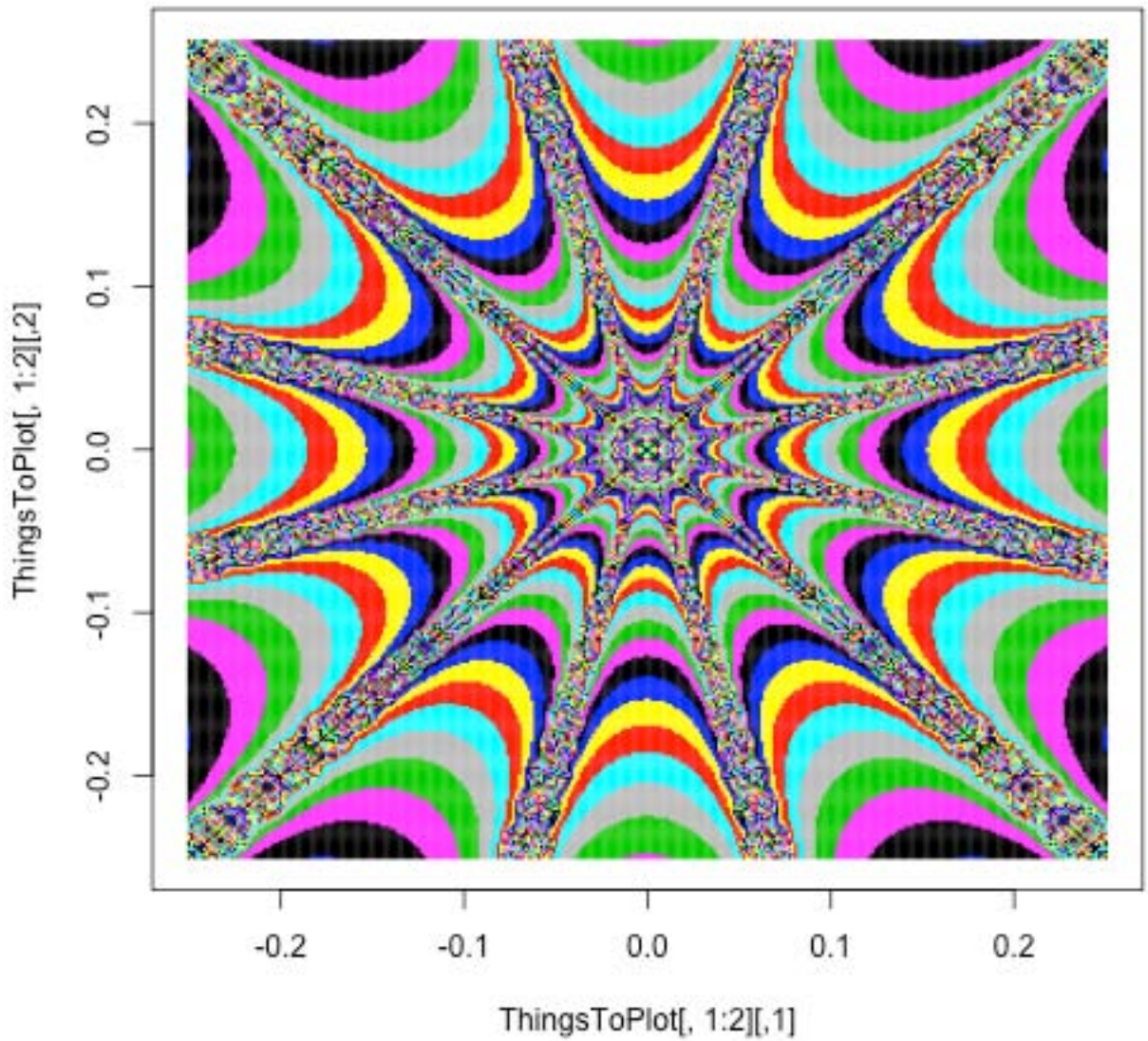

-0.200
-0.202
-0.204
-0.206
-0.208
-0.210
ThingsToPlot[, 1:2][,2]
-0.710
-0.708
-0.706
-0.704
-0.702
-0.700
ThingsToPlot[, 1:2][,1]


-0.2058
-0.2059
-0.2060
-0.2061
-0.2062
ThingsToPlot[, 1:2][,2]
-0.7061
-0.7060
-0.7059
-0.7058
-0.7057
ThingsToPlot[, 1:2][,1]

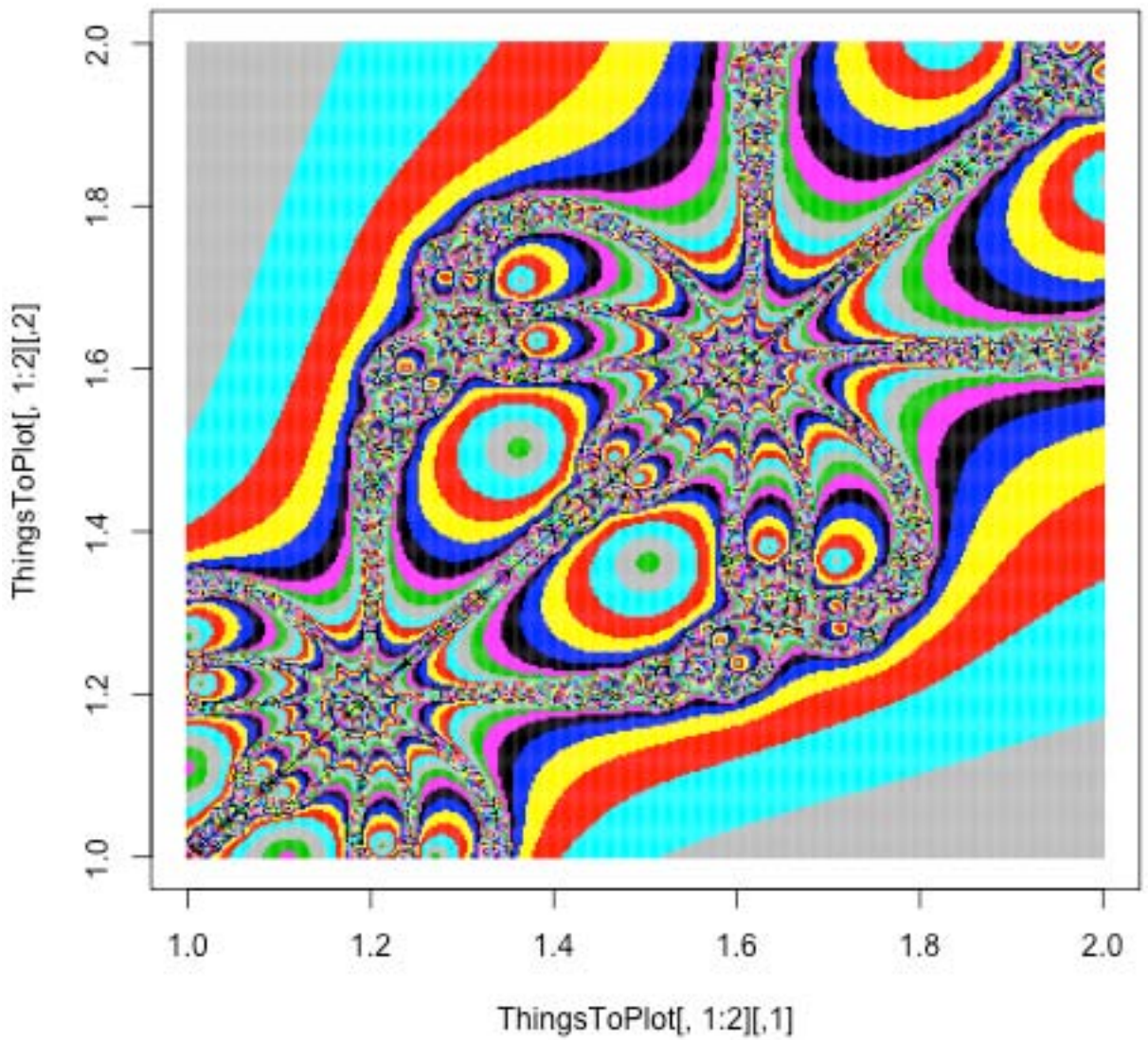
$F(z)=z^4-1$

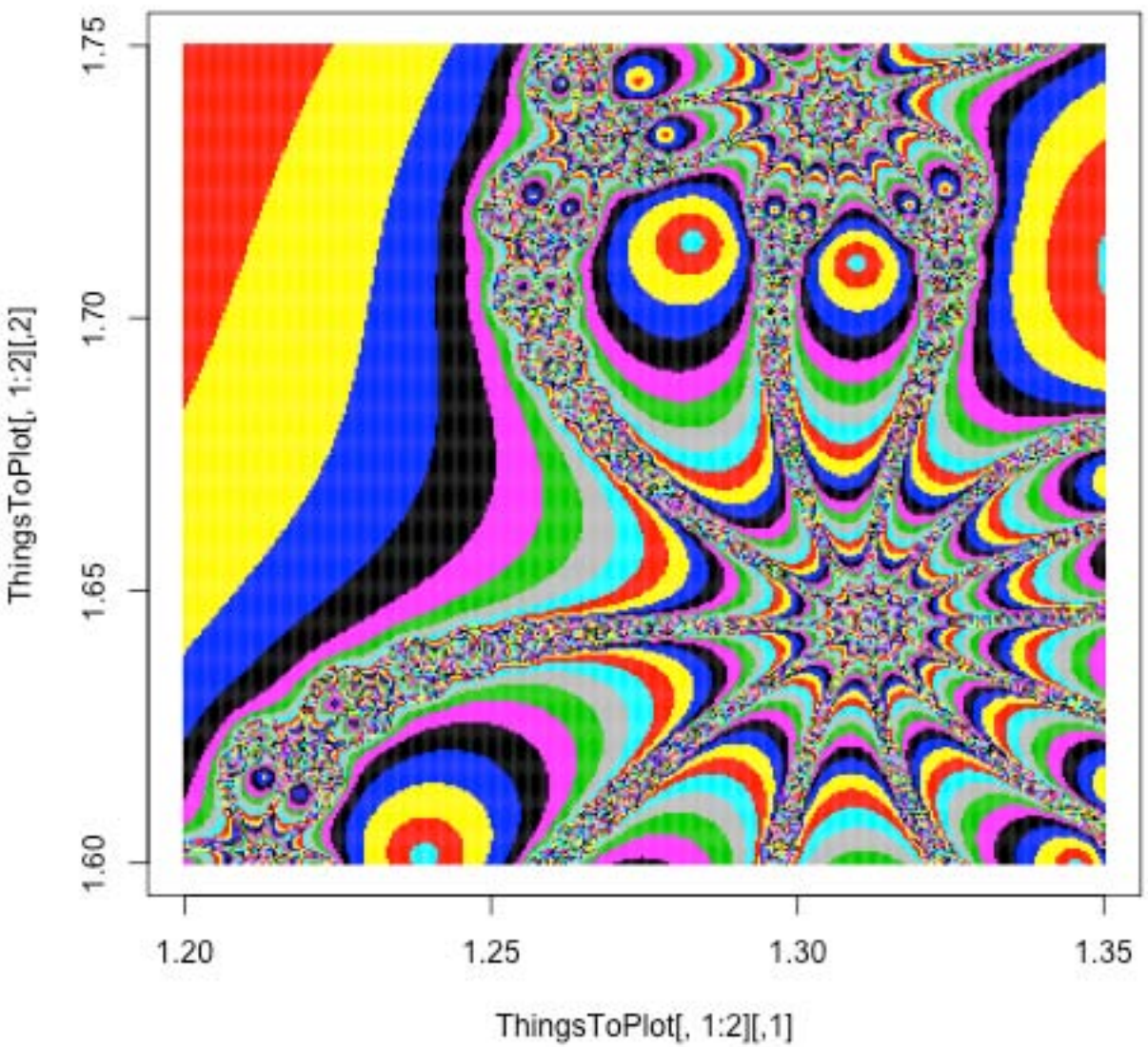


$F(z)=z^4-1$


ThingsToPlot[, 1:2][,2]
0.2
0.1
0.0
-0.1
-0.2
-0.2
-0.1
0.0
0.1
0.2
ThingsToPlot[, 1:2][,1]

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


ThingsToPlot[, 1:2][,2]
2.0
1.8
1.6
1.4
1.2
1.0
1.0
1.2
1.4
1.6
1.8
2.0
ThingsToPlot[, 1:2][,1]

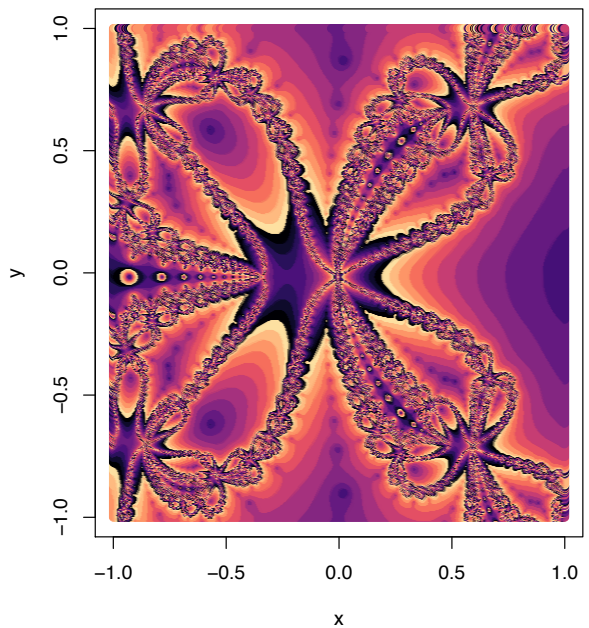

ThingsToPlot[, 1:2][,2]
1.75
1.70
1.65
1.60
1.20
1.25
1.30
1.35
ThingsToPlot[, 1:2][,1]

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

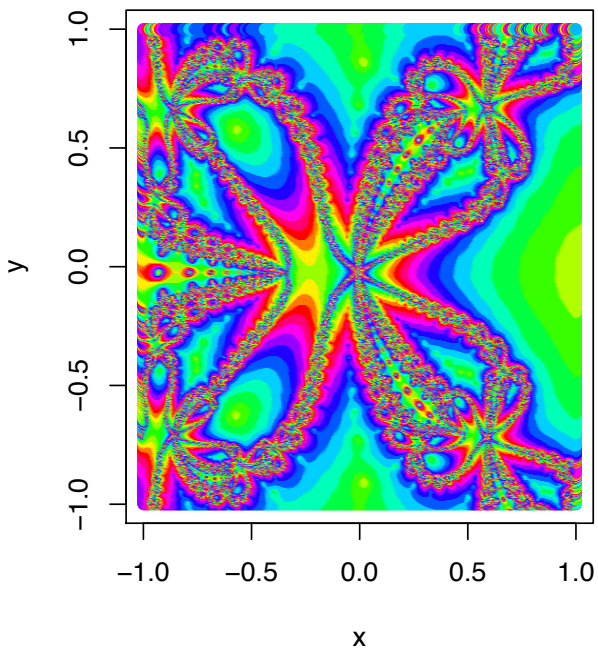
Color palettes

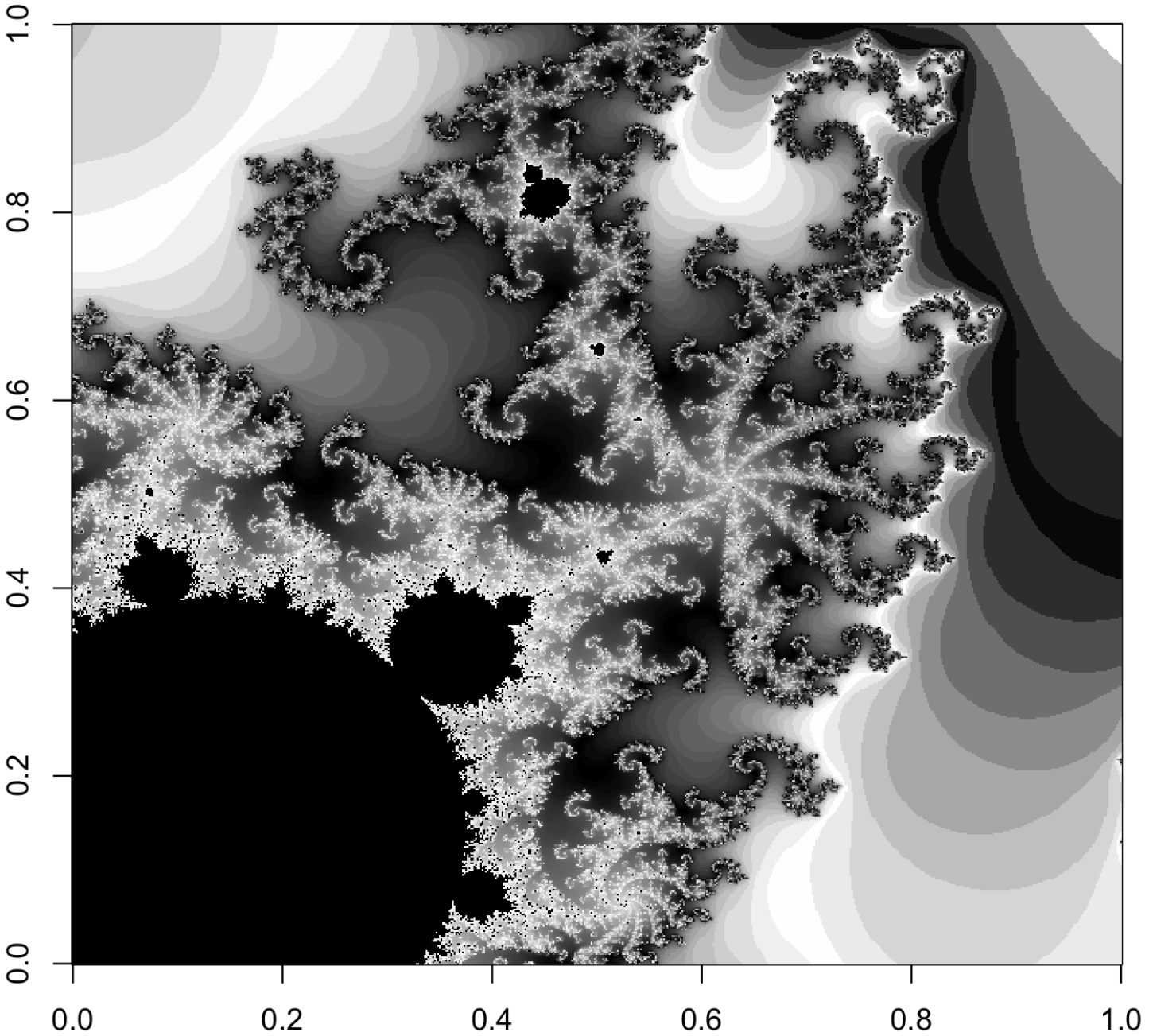
- For better pictures, use something other than R's default color palette. See examples in the FasterFractal repo (versions 6 and 7 of the code)
- E.g. using library("viridis") on $z^4 - 2$, plotting "number of iterations".

palette(magma(64))



palette(rainbow(64))



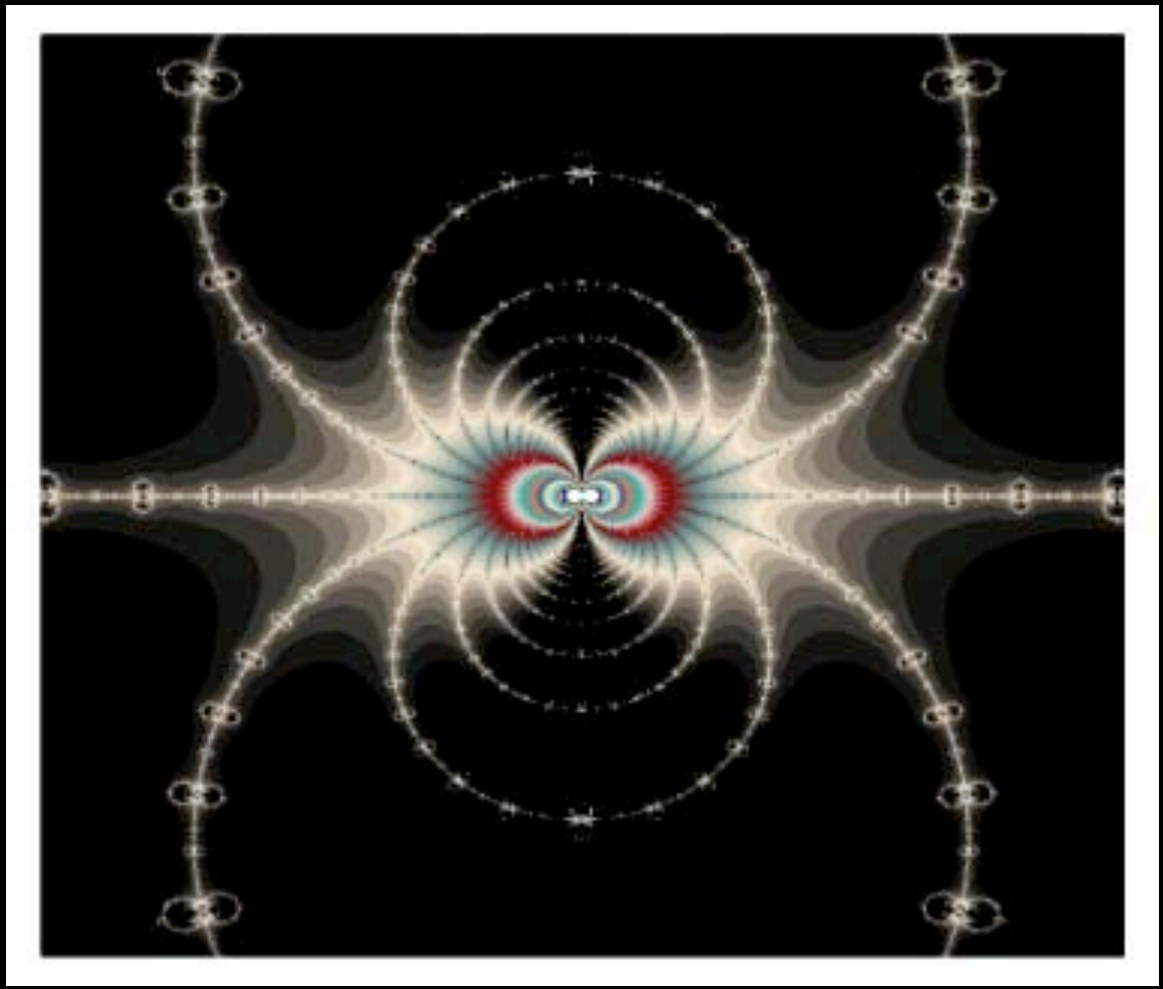


http://people.cryst.bbk.ac.uk/~fdosr01/Rfractals/index.html

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 is a minute - **use this version for your pictures**]
- Use it to draw some fractals for some functions **other than those we saw in today's class**. [Alternatively, 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 three week's time. **They must be drawn by your R code rather than found on the web (and it is ok to just run the code I gave you here).**
- Each person should also commit one picture. You are strongly encouraged to give it a pretentious/artsy name.
- There will be a prize for the best work of art.
- **Deadline: midnight on Wednesday Feb 23rd**

For examples from previous years, see:
https://github.com/PM520-Spring-2022/Week4-ArtShowPreviousYears



The One Who Bit Spiderman

Making code more efficient

- For a problem like this, it is worth investing some time to make your code run more efficiently.
- See example in “FasterFractals” repo.

Rest of class is lab time!

END