Assignment 2a

Yinqi Zhao

2/23/2020

Artshow

The function I choose is $f(x) = 9x^8 - 3x^5 + 2x^2 - 0.7269 + 0.1889i$.

```
# This is an example of how to implement the Newton-Raphson method on complex numbers
# using R's built-in 'complex' number definition, which is written so that you can use +,-,*,/,^ in the

library(future.apply)
```

```
## Loading required package: future
```

```
plan(multisession)

# Next we define a global variable to control whether we record roots or the number of iterations taken
# This is an inelegant way of doing it. It should really be part of the argument to the relevent functi
bRootOrIterations <- 0   # Set <-1 to record which root is found, or <- 0 to record number of iteration


# Here's the function that performs Newton-Raphson
TwoDNewtonRaphson <- function(func, StartingValue, Tolerance, MaxNumberOfIterations) {
  i <- 0  # something to count the iterations
  NewZ <- StartingValue  # start the algorithm at the complex number 'StartingValue'
  Deviation = abs(func(StartingValue))   # Work out how far away from (0,0) func(NewZ) is.

  #Set up a while loop until we hit the required target accuracderivative not defined errory or the max
  while ((i < MaxNumberOfIterations) && (Deviation > Tolerance)) {
    # Find the next Z-value using Newton-Raphson's formula
    Z <- func(NewZ)
    if (is.nan(Z)) {
      break
    }

    # calculate how far f(z) is from 0
    NewZ <- func(NewZ)
    Deviation <- abs(NewZ[1])
    i <- i + 1
    #cat(paste("\nIteration ",i,":   Z=",NewZ,"  Devn=",Deviation))
  }

  # what the function returns depends upon whether you are counting how many iterations it takes
  # to converge or checking which root it converged to...
  if (bRootOrIterations == 1) {
    return(NewZ)
  } else {
```

```r
    return(c(i, i))
  }
}

# A function to check whether two points are close together
CloseTo <- function(x, y) {
  # returns 1 if x is close to y
  if (abs(x - y) < 0.1) {
    return(1)
  } else {
    return(0)
  }
}

# And now here's the function that will draw a pretty picture
Root_calculator <- function(Funcn, xmin, xmax, xsteps, ymin, ymax, ysteps) {
  # First define a grid of x and y coordinates over which to run Newton-Raphson.
  # When we run ut for the point (x,y) it will start with the complex number x+iy

  x <- seq(xmin, xmax, length.out = xsteps)
  y <- seq(ymin, ymax, length.out = ysteps)

  out_dat <- expand.grid(x = x, y = y)

  ThisZ <- complex(1, out_dat$x, out_dat$y)

  Root <- future_sapply(ThisZ,
                        FUN = TwoDNewtonRaphson,
                        func = Funcn,
                        Tolerance = 1e-2,
                        MaxNumberOfIterations = 100)

  if(bRootOrIterations == 0) {
    out_dat$color <- 261 + 5 * Root[1, ]
    out_dat$root1 <- Root[1, ]
    out_dat$root2 <- Root[2, ]
  } else {
    out_dat$color <- 261 + 5 * Root
    out_dat$root1 <- Root
  }

  return(out_dat)
}

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------- tidyverse 1.3.0

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

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

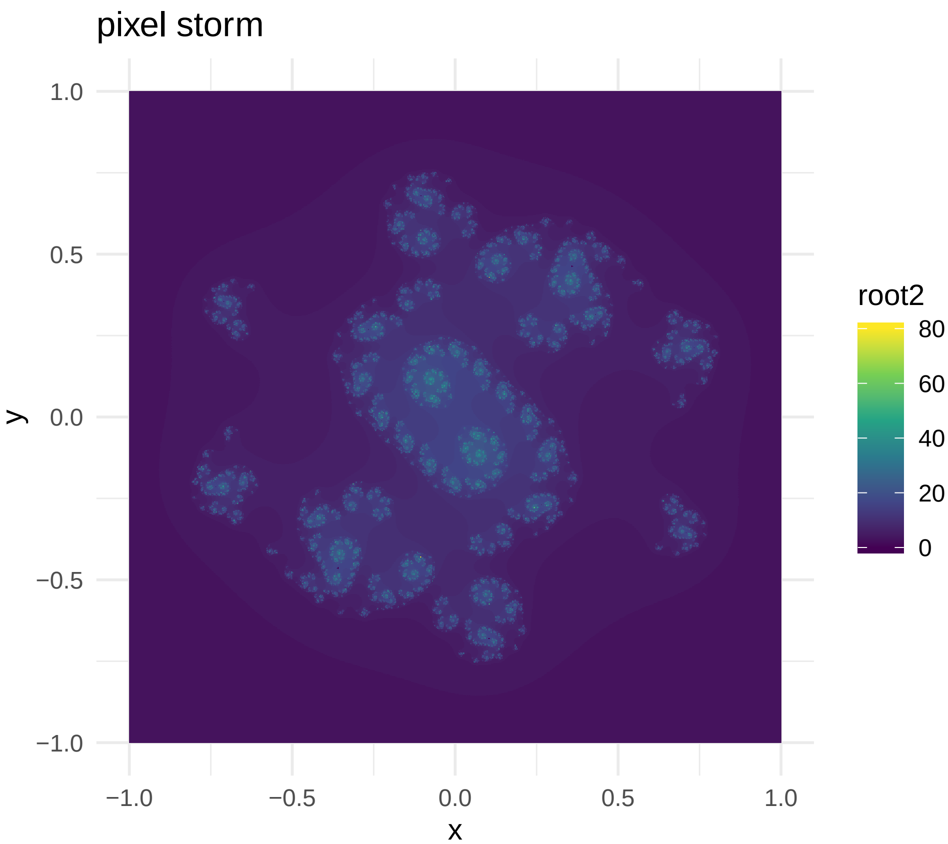
```
## x dplyr::lag()    masks stats::lag()
```

```r
ggplot_plotter <- function(data) {
  ggplot(data, aes(x, y, fill = root2)) +
    geom_tile() +
    scale_fill_viridis_c(direction = 1, na.value = "black") +
    #scale_fill_gradientn(colors = rainbow(100), na.value = "black") +
    theme_minimal() +
    coord_fixed() +
    ggtitle("pixel storm")
}

# The following are a bunch of examples I used to create some of the figures shown in class
# Note that I used a 500x500 grid to get nice pictures, so this will take a while to run
# You shoudl use a smaller grid while you are preactising (say 100,100), but then use a bigger grid for


F7 <- function(z){ # this has three roots 1,-1/2 + sqrt(3)/2i and -1/2 - sqrt(3)/2i
  # a complex function z^3-1
  c <- 0.1027 - 0.527i
  9 * z^8 - 3 * z^6 + z^2 + c # note that arithmetic ops on complex numbers work just like they should
}


A <- Root_calculator(F7, -1, 1, 1000, -1, 1, 1000)
A %>%
  mutate(root2 = if_else(root2 == 100, NA_real_, root2)) %>%
  ggplot_plotter()
```


pixel storm
1.0
0.5
0.0
−0.5
−1.0
y
−1.0
−0.5
0.0
0.5
1.0
x
root2
80
60
40
20
0