Assignment2

Assignment2a The Art Show

The function I choose is

$$y = 4z^3 - 9z^2 + z + c$$

c = -0.1011 + 0.9563i

My defined function is

```
Myfunction <- function(z){
  c <- -0.4749 - 0.21314i
  return(4^z^3 - 9*z^2 + z + c)
  }

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 i
terations taken to find them when drawing our picture
# This is an inelegant way of doing it. It should really be part of the argument to the
 relevent functions.
bRootOrIterations <- 0   # Set <-1 to record which root is found, or <- 0 to record numb
er of iterations needed


# 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) i
s.

  #Set up a while loop until we hit the required target accuracderivative not defined er
rory or the max. number of steps
  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 {
    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 ──
```

```
## ✔ ggplot2 3.2.1     ✔ purrr   0.3.3
## ✔ tibble  2.1.3     ✔ dplyr   0.8.3
## ✔ tidyr   1.0.0     ✔ stringr 1.4.0
## ✔ readr   1.3.1     ✔ forcats 0.4.0
```

```
## ── Conflicts ────────────────────────────────────────────── tidyv
erse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
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 = colorspace::diverge_hcl(5)) +
    #scale_fill_gradientn(colors = colorspace::heat_hcl(10), na.value = "black") +
    #scale_fill_distiller(palette="RdPu")+
    #scale_fill_gradientn(colors = terrain.colors(10), na.value = "black") +
    theme_minimal() +
    coord_fixed() +
    ggtitle("Peony")
}

#terrain.colors(10)

A <- Root_calculator(Myfunction, -1, 1.2, 500, -1.2, 1.2, 500)
ggplot_plotter(A)
```