PM520HW2

Lulu song

2/22/2020

hw2a

art show

```
# 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 usual way

# First define a function to work with - it will return two values.
# The first will be the value of the function at z (i.e., f(z)), the second will be the derivate of f
# at z (i.e., f'(z))
library(viridis)

## Loading required package: viridisLite

palette(viridis(256))
library(future.apply)

## Loading required package: future

plan(multicore)
# I use f(z)=1-z^5+5*z-3/z
F7<-function(z){
  return(c(1-(z^5)+5*z-3/z, 6-5*(z^4)+3*(z^-2))) # note that arithmetic ops on complex numbers work just like they should
}

# Next we define a global variable to control whether we record roots or the number of iterations 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 number 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)[1])   # Work out how far away from (0,0) 
func(NewZ) is.

    #Set up a while loop until we hit the required target accuracy or the 
max. number of steps
  while ((i<MaxNumberOfIterations)&&(Deviation>Tolerance))
  {
    # Find the next Z-value using Newton-Raphson's formula
    Z<-func(NewZ)   # Remember, this is is a vector of two elements. Z[1] is 
the is the value of the function; Z[2] is its derivative
    if ((Z[1]=="NaN")||(Z[2]=="NaN")){
      cat("Function or derivative not defined error.")
      cat("\n",NewZ,Z)
      break
    }

    # So we need to calculate the next value of Z using this formula Z(n+1) 
<- Z(n)-f(Z(n))/f'(z(n))
    NewZ <- NewZ-Z[1]/Z[2]

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

  # output the result
  if (Deviation>Tolerance){
    cat(paste("\nConvergence failure. Deviation:",Deviation, "after ", i, 
"iterations"))
  }

  # 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
RootPlotter<-function(Funcn,xmin,xmax,xsteps,ymin,ymax,ysteps,PtSize)
{
  # 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 <- sapply(ThisZ,
               FUN = TwoDNewtonRaphson,
               func = Funcn,
               Tolerance = 1e-1,
               MaxNumberOfIterations = 100)

if(bRootOrIterations == 0) {
    out_dat$color <- 320 + 8 * Root[1, ]
    out_dat$root1 <- Root[1, ]
    out_dat$root2 <- Root[2, ]
  } else {
    out_dat$color <- 4 + 2 * Root
    out_dat$root1 <- Root
  }

  return(out_dat)
}
library("RColorBrewer")
A <- RootPlotter(F7, -0.6, 0.6, 1000, 0, 1.3, 1000)
library(tidyverse)

## ── Attaching packages ──────────────────────────────────────────────────────
─────────── tidyverse 1.2.1 ──

## ✔ ggplot2 3.2.1     ✔ purrr   0.3.2
## ✔ tibble  2.1.1     ✔ dplyr   0.8.3
## ✔ tidyr   0.8.3     ✔ stringr 1.3.1
## ✔ readr   1.3.1     ✔ forcats 0.4.0
```

```
## ── Conflicts ──────────────────────────────────────────
──────────── tidyverse_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_colour_viridis_c(direction = 1, na.value = "blue", guide =
"colourbar", aesthetics = "colour") +
    scale_fill_gradientn(colours=brewer.pal(n = 12, name = "Set3")) +
    theme_minimal() +
    coord_fixed()+
    labs(title = "butterfly")
}

A %>%
  mutate(root2 = if_else(root2 == 100, NA_real_, root2)) %>%
  ggplot_plotter()
```