PS6

2024-10-20

Q1

Vibrothermography is a modern nondestructive evaluation imaging technique for detecting cracks or flaws in industrial, dental and aerospace applications. The imaging modality works on the principle that a sonic or ultrasonic energy pulse when applied to a unit being tested causes it to vibrate. As a result, it is expected that the faces of a crack will rub against each other, resulting in an increase in temperature in that region. An infrared camera captures this increased temperature and produces a sequence of images of the temperature intensities over a short period of time starting just before the pulse of energy is applied and ending around the time that generated heat has dissipated. A sequence of images records the temperature changes over time. The primary objective of this technology is to detect flaws in the material with a high degree of precision. If the crack is larger than a certain threshold, the part needs to be taken out of operation and repaired or replaced.

The file etcpod 05-400 102307 1 trig01.dat, on measurements made on an aircraft engine part, and available at the class datasets site on Canvas is three-dimensional with 72 × 72 × 150 elements, and each element representing the temperature intensity at that pixel and time-point. Thus there are 150 frames, each of 72 × 72 pixels.

(a)

Read in the dataset and store in an appropriate array. Note that this is simply a file of numeric values in ASCII format.

```
dat <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS6/etcpod_05-400_102307_

dat <- as.matrix(dat)

arrayDat <- array(dat, dim = c(72, 72, 150))

dim(arrayDat)
```

```
## [1]  72  72 150
```

```
# dat <- scan("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS6/etcpod_05-400_102307_1_tr
#
# dim(dat) <- c(72, 72, 150)
```

(b)

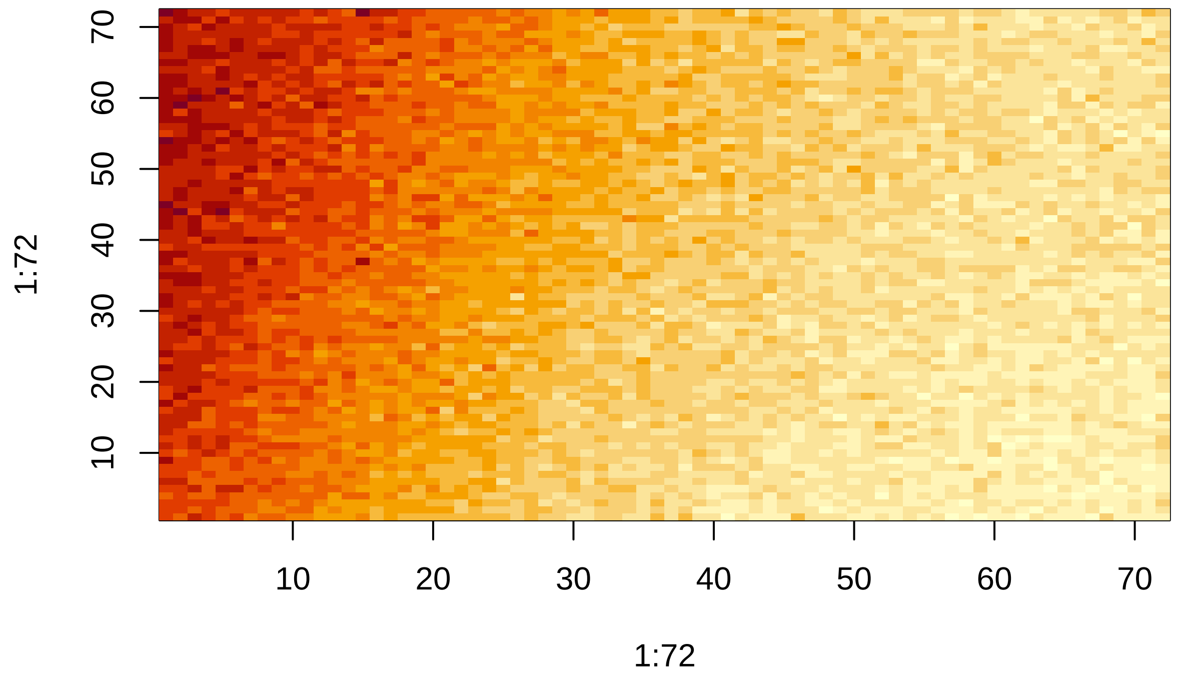
One way to decide on where there is an indication or a flaw in the material is to determine whether there is a hotspot in the image and whether the high temperature elevation is significant enough to warrant further action. However, there are complications. We will not address all of them now, but only some of them in a simplified and idealized setting.

i.

Estimation of background noise. Materials with large grains are noisy (and generate temperatures with widely varying intensities) so we may try to reduce this effect by (crudely) estimating and eliminating the background from each frame. We do so by creating a frame of average intensities from the first five frames (when the sonic or high energy pulse has not been ramped up substantially. Use matrix and array operations and functions to obtain this averaged image.

```r
firstFive <- arrayDat[ , , 1:5]

averageFrames <- apply(
  X = firstFive,
  MARGIN = c(1, 2),
  FUN = mean)
```

```r
image(x= 1:72,
      y = 1:72,
      averageFrames[, 72:1]
      )
```



```r
# firstFive <- dat[ , , 1:5]
#
# averageFrames <- apply(
#   X = firstFive,
#   MARGIN = c(1, 2),
#   FUN = mean)
#
# image(x= 1:72,
#       y = 1:72,
#       averageFrames[, 72:1 ]
#       )
```

ii.

Elimination of background noise. In the next step, we will substract the background from each of the 150 frames. Perform this operation using the background estimate obtained from the previous part.

```
backgroundRep <- array(rep(averageFrames, times = 150),
                       dim = c(72, 72, 150)
                       )

arrayDatSub <- arrayDat - backgroundRep
```

iii.

Identifying frame with hottest signal. Our next objective is to identify the frame in the sequence with the hottest intensity pixel. Using matrix and apply operations, find this frame.

```
maxPerFrame <- apply(X = arrayDatSub,
                     MARGIN = 3,
                     FUN = max)

hottestInd <- which.max(maxPerFrame)
hottestVal <- max(maxPerFrame)

hottestInd
```

```
## [1] 93
```

```
hottestVal
```

```
## [1] 0.15064
```

iv.

Actually, the operation in the previous part is a bit too simplified. To guard against spurious spikes, a smoothing averaging is applied. In other words, for each pixel indexed by (i, j) the values of the pixels indexed by (i-1, j),(i, j-1),(i+1, j),(i, j+1) and (i, j) are averaged and the hottest frame is picked from the one with the highest neighborhood-averaged intensity. Use array techniques to pick out this hottest frame. (Hint: Consider adding another dimension to your 3-d array, and also expanding the frames by one pixel at each edge)

```r
# BEHOLD MY FOR LOOP!
# padFrame <- function(frame) {
#   padded <- matrix(0, nrow = 74, ncol = 74)
#   padded[2:73, 2:73] <- frame
#   return(padded)
# }
#
# paddedArray <- array(0, dim = c(74, 74, 150))
# for (k in 1:150) {
#   paddedArray[,,k] <- padFrame(arrayDatSub[,,k])
# }
#
# smoothFrame <- function(padFrame) {
#   smooth <- matrix(0, nrow = 72, ncol = 72)
#
#   for (i in 2:73) {
#     for (j in 2:73) {
#       smooth[i-1, j-1] <- mean(c(padFrame[i, j],     # current pixel (i,j)
#                                  padFrame[i-1, j],   # pixel above (i-1,j)
#                                  padFrame[i+1, j],   # pixel below (i+1,j)
#                                  padFrame[i, j-1],   # pixel left (i,j-1)
#                                  padFrame[i, j+1]))  # pixel right (i,j+1)
#     }
#   }
#   return(smooth)
# }
#
# smoothArray <- array(0, dim = c(72, 72, 150))
# for (k in 1:150) {
#   smoothArray[,,k] <- smoothFrame(paddedArray[,,k])
# }
#
# maxSmoothed <- apply(X = smoothArray,
#                      MARGIN = 3,
#                      FUN = max)
#
# hottestFrame <- which.max(maxSmoothed)
# hottestSmoothedVal <- max(maxSmoothed)
```

```r
padFrame <- function(frame) {
  padded <- matrix(0, nrow = 74, ncol = 74)
  padded[2:73, 2:73] <- frame
  padded
}

paddedArray <- array(apply(X = arrayDatSub,
                           MARGIN = 3,
                           FUN = padFrame),
                     dim = c(74, 74, 150)
                     )

smoothFrame <- function(padFrame) {
  smooth <- matrix(0, nrow = 72, ncol = 72)
  smooth <- (padFrame[2:73, 2:73] +   # Current pixel
             padFrame[1:72, 2:73] +   # Pixel above
             padFrame[3:74, 2:73] +   # Pixel below
             padFrame[2:73, 1:72] +   # Pixel left
             padFrame[2:73, 3:74]     # Pixel right
             ) / 5

  smooth
}

smoothArray <- array(apply(X = paddedArray,
                           MARGIN = 3,
                           FUN = smoothFrame),
                     dim = c(72, 72, 150)
                     )

maxSmoothed <- apply(X = smoothArray,
                     MARGIN = 3,
                     FUN = max)

hottestFrame <- which.max(maxSmoothed)
hottestSmoothedVal <- max(maxSmoothed)
```

```r
hottestFrame
```

```
## [1] 93
```

```r
hottestSmoothedVal
```

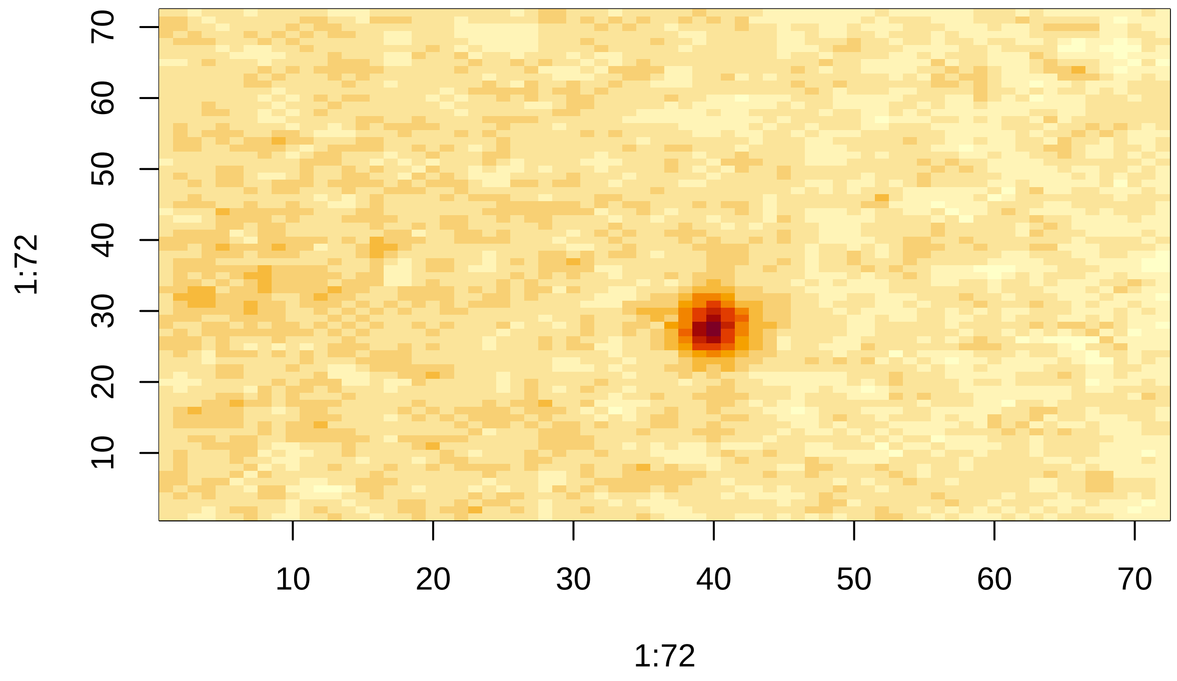
```
## [1] 0.1329
```

v.

Display. Plot the resulting frame in the part (with or without extra credit above).

```r
hottestFrameSmooth <- smoothArray[ , , 93]

# image(hottestFrame)

image(x= 1:72,
      y = 1:72,
      hottestFrameSmooth[, 72:1]
      # xlab = "",
      # ylab = "",
      # axes = FALSE
      )
```

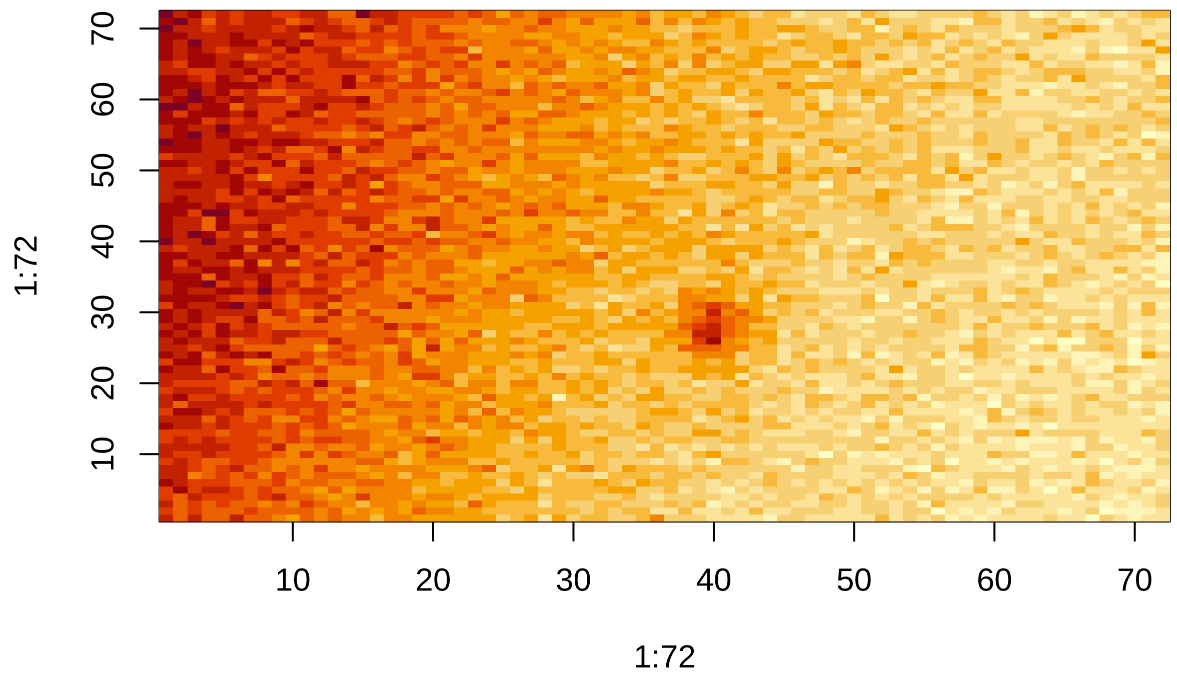
Frame 93 With Smoothing



```r
hottestFrame <- arrayDat[ , , 93]
```

```r
# image(hottestFrame)

image(x= 1:72,
      y = 1:72,
      hottestFrame[, 72:1]
      # xlab = "",
      # ylab = "",
      # axes = FALSE
      )
```

Frame 93 as-is, Without smoothing



Q2

Complex materials development involves understanding the molecular structure of the material being tested and the movement of molecules under application of force. The dataset available in the Excel spreadsheet file PreliminaryData.xlsx on the class datasets site on Canvas was on measurements recorded by a former graduate student in Iowa State University's Materials Science Engineering department. The observations are on several things but for this exercise, we will ignore most of them, but for the last two columns (which are measurements obtained by two methods) and the x- and y- coordinates.

(a)

Read in and store the dataset. Note that because this file was written in Excel which does not require formatting, there are three columns which are meaningless (and should be removed to conserve space).

```
library(readxl)
PreliminaryData <- read_excel("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5790/PS/PS6/Prelimina
```

```
## New names:
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...8`
## * `` -> `...9`
## * `` -> `...11`
```

```
cleanPrelim <- PreliminaryData |>
  # subset( , select = -c(8, 9, 11))
  subset( , select = c(5, 6, 10, 12))

names(cleanPrelim) <- c("x", "y", "method1", "method2")
```

(b)

Data quality. The measurements in the last two columns are measurements on a grid. We will now investigate if every value in the x- and y coordinate is measured. (Note that both the x and y coordinates increment in terms of 10 units and start at 0.). An important aspect of this data quality check is to ascertain that every grid value is present. It would be helpful to know if the x- and the y-values are increasing in a nested sequence. Run checks to determine if this is indeed true.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
xVal <- sort(unique(cleanPrelim$x))
yVal <- sort(unique(cleanPrelim$y))

xInc <- all(diff(xVal) == 10)
yInc <- all(diff(yVal) == 10)

xRep <- rep(x = xVal,
                  each = length(yVal))
yRep <- rep(x = yVal,
                  times = length(xVal))

expMat <- data.frame(x = xRep,
                     y = yRep)

missingXy <- anti_join(x = expMat,
                       y = cleanPrelim,
                       by = c("x", "y")
                       )

missingXy
```

```
## [1] x y
## <0 rows> (or 0-length row.names)
```

(c)

For each of the (old and new) method measurements, we will create a matrix of the observed values (after scaling these down by 10), where the rows will correspond to x- and the columns will correspond to the y-coordinates (and the value of the measurement will be stored in the (x, y)th entry.) Do the above.

```r
xScaledUnique <- sort(unique(cleanPrelim$x / 10))
yScaledUnique <- sort(unique(cleanPrelim$y / 10))

method1Mat <- matrix(NA,
                     nrow = length(xScaledUnique),
                     ncol = length(yScaledUnique)
                     )

method2Mat <- matrix(NA,
                     nrow = length(xScaledUnique),
                     ncol = length(yScaledUnique)
                     )

xIndex <- match(cleanPrelim$x / 10, xScaledUnique)
yIndex <- match(cleanPrelim$y / 10, yScaledUnique)

method1Mat[cbind(xIndex, yIndex)] <- cleanPrelim$method1
method2Mat[cbind(xIndex, yIndex)] <- cleanPrelim$method2
```

Hey, so I realize the above uses the function "match", which is a dplyr-specific function. I did also initially get this working with a "for" loop as well as using it with "apply" usage. If you are going to replicate any of the below, the "apply" one takes a looooooong time to run.

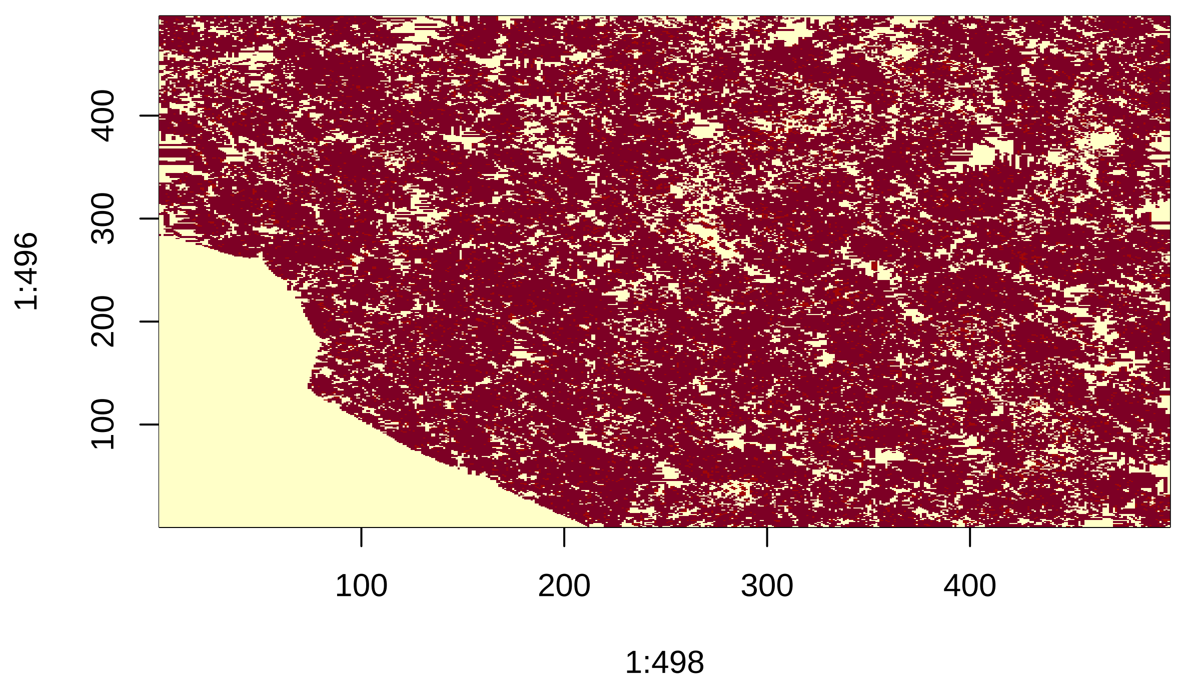
Please Gautham, have mercy!

```r
# BEHOLD THE FALLEN!
# cleanPrelim$xScaled<- cleanPrelim$x / 10
# cleanPrelim$yScaled <- cleanPrelim$y / 10
#
# xScaledUnique <- sort(unique(cleanPrelim$xScaled))
# yScaledUnique <- sort(unique(cleanPrelim$yScaled))
#
# method1Mat <- matrix(NA,
#                       nrow = length(xScaledUnique),
#                       ncol = length(yScaledUnique)
#                       )
#
# method2Mat <- matrix(NA,
#                       nrow = length(xScaledUnique),
#                       ncol = length(yScaledUnique)
#                       )

# apply(X = cleanPrelim,
#       MARGIN = 1,
#       FUN = function(row) {
#   xIndex <- match(row['xScaled'], xScaledUnique)
#   yIndex <- match(row['yScaled'], yScaledUnique)
#
#   method1Mat[xIndex, yIndex] <- row['method1']
# })
#
# apply(X = cleanPrelim,
#       MARGIN = 1,
#       FUN = function(row) {
#   xIndex <- match(row['xScaled'], xScaledUnique)
#   yIndex <- match(row['yScaled'], yScaledUnique)
#
#   method2Mat[xIndex, yIndex] <- row['method2']
# })

# for (i in 1:nrow(cleanPrelim)) {
#   xIndex <- match(x = cleanPrelim$xScaled[i],
#                   table = xScaledUnique)
#   yIndex <- match(x = cleanPrelim$yScaled[i],
#                   table = yScaledUnique)
#
#   method1Mat[xIndex, yIndex] <- cleanPrelim$method1[i]
#   method2Mat[xIndex, yIndex] <- cleanPrelim$method2[i]
# }

# rownames(method1Mat) <- xScaledUnique
# colnames(method1Mat) <- yScaledUnique
# rownames(method2Mat) <- xScaledUnique
# colnames(method2Mat) <- yScaledUnique
```

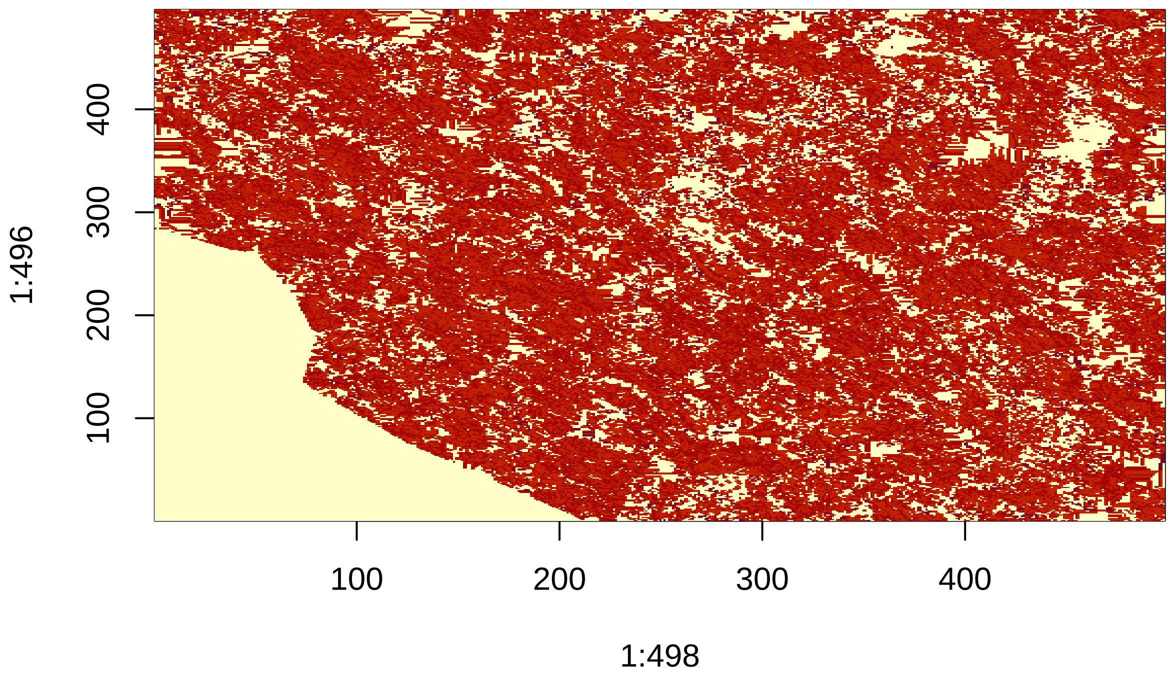
(d)

For both sets of measurements, display the matrix in terms of an image.

```r
image(x = 1:498,
      y = 1:496,
      method1Mat[, 496:1]
      )
```



```r
image(x = 1:498,
      y = 1:496,
      method2Mat[, 496:1]
      )
```


1:496
100
200
300
400
1:498