Assignment 6

1. Revise your R package `pareto` to compute the pareto density, CDF, and quantile function using C code. A version of the `AddOne` package that uses C code for its computations can serve as an example. This uses a mechanism for C routine registration that is currently mandatory for CRAN submissions. You can use the

   `package_native_routine_registration_skeleton`

   function in the `tools` package to generate this registration code. This package also uses the `.registration` and `.fixes` arguments with `useDynLib` in the `NAMESPACE` file.

   Your package should pass `R CMD check` without errors or warnings. You should submit your package as a source package file created by `R CMD build`.

   Also commit and push your revised package code to your class GitLab repository.

2. This problem is based on the BrainWeb artificial brain imaging data base, which is used for assessing the performance of brain image processing methods. The data are available on the workstations in

   `/group/statsoft/data/Brainweb/images`

   The motivating problem is classifying individual volume elements (or *voxels*) in an image into the three major brain tissue types (gray matter, white matter, and cerebro-spinal fluid) based on MR images. Several image types are available; this problem will look at a T1 image, the type most commonly used for tissue classification. The Wikibedia page

   `http://en.wikipedia.org/wiki/Magnetic_resonance_imaging`

   provides some useful background.

   Brain images are large 3D arrays and some care is used in storing them to reduce size and allow for efficient access to full images or slices of images. A number of structured file formats are in use, including Minc, Analyze, and Niftii. The data we will use is in a simple binary form that can easily be read using `readBin`. A simple function to do this is

```
readVol <- function(fname, dim = c(181, 217, 181)) {
    f<-gzfile(fname, open="rb")
    on.exit(close(f))
    b<-readBin(f,"integer",prod(dim),size = 1, signed = FALSE)
    array(b, dim)
}
```

The data have been scaled and rounded to an integer value between 0 and 255 and encoded as an unsigned byte.

The particular image we will use is stored in the file

```
/group/statsoft/data/Brainweb/images/T1/t1_icbm_normal_1mm_pn3_rf20.rawb.gz
```

In this problem we will ignore the spatial structure of the images and fit a normal mixture model to the image intensities. This model can then be used to classify the voxels into the three tissue types.

(a) Read the data and use the `image` function to plot a middle slice along each axis of the three-dimensional array.

(b) A first step in brain tissue classification is to identify the part of the image corresponding to the brain. This is often done by specialized pre-processing methods; a mask image, in the same format as the T1 image, with volume elements in the brain coded as one and elements outside the brain as zero is available in

```
/group/statsoft/data/Brainweb/images/mask.rawb.gz
```

Read in the mask and plot an estimate of the density of the image intensities for voxels in the brain. You should see three peaks; the lower one corresponds to cerebro-spinal fluid (CSF), the middle one to gray matter, and the higher one to white matter.

(c) Write a function to fit a normal mixture model by the EM algorithm and use the function to fit a model to the intensities within the brain. You can use the single step `EMmix1` functon from the class notes.

(d) Write a function that takes the intensities and the normal moxture parameters and returns for each voxel the index of the most likely tissue class for that voxel. The E step computations and the function `max.col` may be useful. Show the result using the `image` function for the same slices used in the first part of this problem.

(e) The EM calculation may be rather slow. Use profiling tools `Rprof` and `summaryRprof` to see where your code is spending most of its time.

(f) Can you improve peformance by taking advantage of the fact that the intensities can take on only a relatively small number of distinct values?

Your submission should include your source package archive for Problem 1 and a pdf file with your writeup, and text files with a `.R` extension with your code for Problem 2.

You should submit your assignment electronically using Icon. Submit your work as a single compressed tar file. If your work is in a directory `mywork` then you can create a compressed tar file with the command

```
tar czf mywork.tar.gz mywork
```

Solutions and Comments

1.
   - Please follow the coding standards on use of spaces, indentation and long lines.
   - It is a good idea to use the registration mechanism. This is required for CRAN submissions.
     - When using the registration mechanism it is a good idea to make as many definitions static as possible; using a single file for your C code helps.
     - If you want to use multiple files use the `attribute_hidden` prefix. Or you can make all symbols invisible by default and selectively enable some with `attribute_visible`. The *Controlling Visibility* section of the extensions manual gives details.
     - It is also a good idea to use optional arguments to `useDynLib` to define variables for your entry points. You can use these instead of using strings in your `.C` calls — this can improve performance.
   - Your package should pass `R CMD check` without errors, warnings, or notes.
   - Make sure to include a reasonable set of tests in your package.
   - When using `.C` you should:
     - Check that scalar arguments like `log.p` have at least one element.
     - Use `as.double` or `as.integer` to make sure that the values passed to `.C` have the right types.
   - You should use the constants `NA_REAL`, `R_NaN`, `R_PosInf` or `R_NegInf` in C code when you need these values.
   - If you issue warnings make sure to do so only once per call — do not call `warning` or `error` inside your C loop.
   - Be sure to follow the coding standards in you R and your C code.
   - Make sure you push your changes to your class GitLab repository.
   - You should only export those variable that you want to be public; don't just export all variables.

2. Some notes:
   - It helps to use a good qualitative color scheme for showing the classification results in part (d). Some resources:
     - HCL Wizard
     - ColorBrewer.
   - The axes and labels provided by default in image plots are not useful in this case.

- It is a good idea to use an appropriate aspect ratio for these images.
  - A poor aspect ratio can be distracting:

```
library(ggplot2)
ggplot(map_data("state")) +
    geom_polygon(aes(long, lat, group = group),
                 fill = "white", color = "black")
```

  - Aspect ratio can affect what we see:

```
river <- scan("https://www.stat.uiowa.edu/~luke/data/river.dat")
plot(river)
```

- It is a good idea to describe the convergence criterion used for the EM algorithm. A relative error on the order of 10^{-8} might be reasonable. Using too large a tolerance can cause problems.
- The classification can be computed with a vectorized expression, for example as

```
tclass <- function(x, theta) {
    mu <- theta$mu
    sigma <- theta$sigma
    p <- theta$p
    M <- length(mu)

    Ez <- outer(x, 1:M, function(x, i) p[i] * dnorm(x, mu[i], sigma[i]))
    max.col(Ez, ties.method = "first")
}
```

  The image can then be filled in with

```
cl <- array(NA, dim(T1))
cl[mask] <- tclass(v, res$pars)
```

- For part (f), there are only a few hundred distinct intensity values, and the function `tabulate` can be used to determine how often each occurs. The EM computations can be rewritten to carry out each distinct calculation once and multiply by the appropriate count. This is easiest to do by modifying the EM step to take a weights argument:

```
EMmix1W <- function(x, w, theta) {
    mu <- theta$mu
    sigma <- theta$sigma
    p <- theta$p
    M <- length(mu)

    ## E step
    Ez <- outer(x, 1:M, function(x, i) p[i] * dnorm(x, mu[i], sigma[i]))
    Ez <- sweep(Ez, 1, rowSums(Ez), "/")
```

```
    EzW <- sweep(Ez, 1, w, `*`)
    colSums.EzW <- colSums(EzW)

    ## M step
    xpW <- sweep(EzW, 1, x, "*")
    mu.new <- colSums(xpW) / colSums.EzW

    sqRes <- outer(x, mu.new, function(x, m) (x - m)^2)
    sigma.new <- sqrt(colSums(EzW * sqRes) / colSums.EzW)

    p.new <- colSums.EzW / sum(colSums.EzW)

    ## pack up result
    list(mu = mu.new, sigma = sigma.new, p = p.new)
}
```

This can be tested and compared to the original EM step with

```
counts <- table(as.vector(v))
vv <- sort(unique(as.vector(v)))

EMmix1W(vv, counts, theta)
```

- Binning or discretization can be useful for some large data set computations; for example

  `http://vita.had.co.nz/papers/bigvis.html`
  `https://github.com/hadley/bigvis`

- A recent blog post describes approaches to binning in a data base.