Name: ____________________________

Directions: MAKE SURE TO COPY YOUR ANSWERS TO A SEPARATE SHEET FOR SENDING ME AN ELECTRONIC COPY LATER.

1. (100) This problem implements the Mandelbrot computation on p.28 of our book. The idea is to generate the rectangular grid of points, then for each grid point, determine whether the sequence generated in Equation (2.1) in our text remains bounded after a specified number of iterations. Our definition of "bounded" will be that all the points in the sequence are within the disk of radius 2.0 with center (0,0).

The function **mandelbrot()** will do the main computation, returning an R list that is been designed to feed in to the R **image()** function. We will break the computation into chunks, and use R Snow to parallelize.

Say our X-axis interval is (3,4), while for Y it is (2,2.4), with an increment 0.2 between tick marks on the axes. The marks on X will be 3,3.2,...,4 and on Y they will be at 2,2.2,2.4. That's 6 marks on X and 3 on Y, for a total of 18 grid points.

Note:

```
> seq(1.2,2.2,0.3)
[1] 1.2 1.5 1.8 2.1
```

Fill in the blanks:

```
# arguments:

#     xl: left limit
#     xr: right limit
#     yt: top limit
#     yb: bottom limit
#     inc: distance between ticks on X, Y axes
#     maxiters: maximum number of iterations

# return value:

#     we aim to return an R list which will be
#     used as input to R's image() function:
#     a matrix of 1s and 0s, 1 meaning sequence
#     remains bounded, absolute value <= 2,
#     plus the tick marks vectors for the X
#     and Y axes

mandelbrot <-
     function(xl,xr,yb,yt,inc,maxiters)
{
   # determine where the tick marks go on
   # the X and Y axes
   xticks <- seq(xl,xr,inc)
   yticks <- seq(yb,yt,inc)
   nxticks <- length(xticks)
   nyticks <- length(yticks)
   m <- matrix(0,nrow=nxticks,ncol=nyticks)

   for (i in 1:nxticks) {
      xti <- xticks[i]
      for (j in 1:nyticks) {
         ytj <- yticks[j]
         # this is c in (2.1), but don't
         # confuse with R's c() ftn
         cpt <-
            complex(real=xti,imaginary=ytj)
         z <- cpt
         for (k in 1:maxiters) {
            blank (a)
            if ( blank (b) ) break
            blank (c)
         }
      }
   }
   list(x=xticks,y=yticks,z=m)
}

# R Snow wrapper, divide grid into strips,
# then call mandelbrot() on each one; we won't
# mind if there are a few duplicate pixels in the
# end

mandelsnow <-
   function(cls,xl,xr,yb,yt,inc,maxiters)
{
   clusterExport(cls,c("xl","xr",
      "yb","yt","inc","maxiters"),
      envir=environment())
   ncls <- length(cls)
   clusterExport(cls,
      c("mandelbrot","ncls"),
      envir=environment())
   getmyxinterval <- function(i) {
      width <- (xr - xl) / ncls
      myxl <<- blank (d)
      myxr <<- blank (e)
   }
   blank (f)
   tmp <- clusterEvalQ(cls,
      blank (g) )
   fullmat <- NULL
   for (i in 1:ncls)
      fullmat <- blank (h)
   g <- list()
   g$x <- seq(xl,xr,inc)
   g$y <- seq(yb,yt,inc)
   g$z <- fullmat
   # make up for possible overlap
   nrx <- length(g$x)
   nrz <- nrow(g$z)
   if (nrx < nrz) {
      g$z <- g$z[-(1:(nrz-nrx)),]
   } else if (nrz < nrx)
      g$x <- g$x[-(1:(nrx-nrz))]
   image(g)
}
```

Solutions:

```
# computes Mandelbrot fractal matrix

# adapted from code by ucgamdo@ucl.ack.uk

# arguments:

#     xl: left limit
#     xr: right limit
#     yt: top limit
#     yb: bottom limit
#     inc: distance between ticks on X, Y axes
#     maxiters: maximum number of iterations

# return value:

#     we aim to return an R list which will be used as input to R's
#     image() function: a matrix of 1s and 0s, 1 meaning sequence remains
#     bounded, absolute value <= 1, plus the tick marks vectors for the X
#     and Y axes

mandelbrot <- function(xl,xr,yb,yt,inc,maxiters)
{
   # determine where the tick marks go on the X and Y axes
   xticks <- seq(xl,xr,inc)
   yticks <- seq(yb,yt,inc)
   nxticks <- length(xticks)
   nyticks <- length(yticks)
   # eventual return value
   m <- matrix(0,nrow=nxticks,ncol=nyticks)

   # iteratate through the entire grid, setting c to each grid point and
   # then seeing where z goes as a result
   for (i in 1:nxticks) {
      xti <- xticks[i]
      for (j in 1:nyticks) {
         ytj <- yticks[j]
         # this is c, but don't confuse with R's c() ftn
         cpt <- complex(real=xti,imaginary=ytj)
         z <- cpt
         for (k in 1:maxiters) {
            z <- z^2 + cpt
            if (abs(z) > 2.0) break
            if (k == maxiters) m[i,j] <- 1
         }
      }
    }
    # return
    list(x=xticks,y=yticks,z=m)
}

# R Snow wrapper, divide grid into vertical strips, then call
# mandelbrot(); we won't mind if there are a few duplicate pixels in the
# end

mandelsnow <-
   function(cls,xl,xr,yb,yt,inc,maxiters)
{
   clusterExport(cls,c("xl","xr",
      "yb","yt","inc","maxiters"),
      envir=environment())
   ncls <- length(cls)
   clusterExport(cls,
      c("mandelbrot","ncls"),
      envir=environment())
   getmyxinterval <- function(i) {
      width <- (xr - xl) / ncls
      myxl <<- xl + (i-1) * width
      myxr <<- myxl + width
   }
   clusterApply(cls,1:ncls,getmyxinterval)
   tmp <-
      clusterEvalQ(cls,mandelbrot(myxl,myxr,yb,yt,inc,maxiters))
   fullmat <- NULL
   for (i in 1:ncls)
      fullmat <- rbind(fullmat,tmp[[i]]$z)
   g <- list()
   g$x <- seq(xl,xr,inc)
   g$y <- seq(yb,yt,inc)
   g$z <- fullmat
   # make up for possible overlap
   nrx <- length(g$x)
   nrz <- nrow(g$z)
   browser()
   if (nrx < nrz) {
      g$z <- g$z[-(1:(nrz-nrx)),]
   } else if (nrz < nrx)
      g$x <- g$x[-(1:(nrx-nrz))]
   image(g)
}
```