Name: _____

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing.

1. (30) Fill in the blanks in the following R code, which computes and returns the matrix A in Equation (10.20), C = AX/n. Note: R's exponentiation operator is `^`, so that for instance `2^3` is 8.

R's **diag()** function is quite versatile. When applied to a matrix, you get a vector (formed from the diagonal of the matrix), and vice versa.

```
makeamat <- fqnction(n,q)  {
   m <- matrix(nrow=n,ncol=n)
   for (i in 1:n) {
      for (j in i:n) {
         if (i == j) {
                                  # first blank (one line)
         }
         else {
                                  # second blank (one line)
            m[j,i] <- m[i,j]
         }
      }
   }
   return(m)
}
```

2. (30) Fill in the blanks in the following R code to implement the Jacobi algorithm on a GPU. It solves **ax = b**, returning **x**.

```
library(gputools)

jcb <- function(a,b,eps) {
   n <- length(b)
   d <- diag(a)  # a vector, not a matrix
   tmp <- diag(d)  # a matrix, not a vector
   o <-                     # first blank (partial line)
   di <- 1/d
   x <- b  # initial guess, could be better
   repeat {
      oldx <- x
      tmp <-                # second blank (partial line)
      tmp <- b - tmp
      x <- di * tmp  # elementwise multiplication
                            # third blank (one line)
   }
}
```

3. (40) Fill in the blanks in the following R code, which implements smoothing (removal of "blips") on sound data (loudness) **snd**, cutting off frequencies for k greater than **maxidx** in (10.13). R's **fft()** function, when applied to a vector **x**, finds the Discrete Fourier Transform of **x**. If the optional second argument **inverse** is set to TRUE, it will find the inverse transform instead. You'll find R's **rep()** function useful; e.g. **rep(2,5)** is the vector (2,2,2,2,2).

```
lp <- function(snd,maxidx) {
         four <-          # first blank (partial line)
         n <- length(four)
         newfour <-       # second blank (partial line)
                          # third blank (one line)
}
```

Solutions:

1.

```
makeamat <- function(n,u)  {
   m <- matrix(nrow=n,ncol=n)
   for (i in 1:n) {
      for (j in i:n) {
         if (i == j) {
            m[i,i] <- u^((i-1)^2)
         }
         else {
            m[i,j] <- u^((i-1)*(j-1))
            m[j,i] <- m[i,j]
         }
      }
   }
   return(m)
}
```

2.

```
library(gputools)

jcb <- function(a,b,eps) {
   n <- length(b)
   d <- diag(a)  # a vector, not a matrix
   tmp <- diag(d)  # a matrix, not a vector
   o <- a - diag(d)
   di <- 1/d
   x <- b  # initial guess, could be better
   repeat {
      oldx <- x
      tmp <- gpuMatMult(o,x)
      tmp <- b - tmp
      x <- di * tmp  # elementwise multiplication
      if (sum(abs(x-oldx)) < n * eps) return(x)
   }
}
```

3.

```
lp <- function(snd,maxidx) {
         four <- fft(snd)
         n <- length(four)
         newfour <- c(four[1:maxidx],rep(0,n-maxidx))
         return(Re(fft(newfour,inverse=T)/n))
}
```