Appendix B: Code Scripts

This appendix contains the necessary code to reproduce the Three Circles results in Section 4.

Please go to <https://github.com/JimmyJHickey/EM-for-Transmission-Tomography> to find code scripts and data files to generate figures and results for other theta patterns.

##################### Construct a Matrix to Represent a Circle #####################

in\_circle = function(radius){

# radius of scan

# approximately 5% bigger than the radius of the patient cross section

out\_radius = as.integer(radius \* 1.05 +1)

center = (2 \* out\_radius + 1 + 1) %/% 2

nrow = 2 \* out\_radius + 3

ncol = 2 \* out\_radius + 3

# start everything at -1

in\_circ = matrix(-1, nrow = nrow, ncol = ncol)

# if outside of the scan -1

# if inside the scan but outside the patient 0

# if inside the patient random(Uniform(0, 0.25))

for(x in 1:nrow){

for(y in 1:ncol){

# if within the patient radius

if( sqrt( (x - center -1 )^2 + (y - center-1)^2 ) <= radius ){

in\_circ[x,y] = runif(1, 0, 0.25)

}

else if( sqrt( (x - center -1 )^2 + (y - center-1)^2 ) <= out\_radius ){

in\_circ[x,y] = 0

}

}

}

return(in\_circ)

}

############################## Plot a Matrix of Pixels ##############################

plot\_matrix = function(in\_mat){

require(ggplot2)

x = seq(1, nrow(in\_mat))

y = seq(1, ncol(in\_mat))

df = expand.grid(X=x, Y=y)

df$Z = c(in\_mat)

# doesn't seem to care about those colors at all, but it works

ggplot(df, aes(X, Y, fill= Z)) +

geom\_tile()+

scale\_fill\_continuous(type = "viridis",

limits = c(0,4),

breaks = c(0, 1, 2, 3, 4),

guide\_colourbar(nbin = 100),

name = "theta")

}

######################## Create Circles within Cross Section ########################

circle\_pattern = function(in\_mat, radius, center\_x, center\_y){

out\_mat = in\_mat

for(x in 1:nrow(out\_mat)){

for(y in 1:ncol(out\_mat)){

# if in the outer circle and the inscribed circle

if((in\_mat[x,y] >0) &&

(sqrt( (x - center\_x)^2 + (y - center\_y)^2 ) <= radius )){

# generate data inversely proportional to

# distance from center of inscribed circle

out\_mat[x,y] = out\_mat[x,y] + abs(rnorm(1,

1/sqrt( 1+(x - center\_x)^2 + (y - center\_y)^2),

1))

}

}

}

return(out\_mat)

}

######################## Simulate CT Scan X-ray Projections ########################

#ST793 Final Project

#EM algorithm for CT Data

#Eric Yanchenko, Alvin Sheng, and Jimmy Hickey [Copyright]

#November 4, 2020

# calculates column-majorized vector index of a matrix index

calc\_index = function(nrow, x, y){

return(nrow \* (y-1) + x)

}

# inverse of calc\_index (for testing purposes)

calc\_matrix\_index <- function(idx, num\_row) {

row\_idx <- rep(NA, length(idx))

col\_idx <- rep(NA, length(idx))

for (i in 1:length(idx)) {

col\_idx[i] <- ceiling(idx[i] / num\_row)

row\_idx[i] <- idx[i] - (col\_idx[i] - 1) \* num\_row

}

# This order is more intuitive as Cartesian coordinates

return(cbind(col\_idx, row\_idx))

}

# returns a logical vector indicating the entries that are -1 and at either end of the vector

# I use this method of taking out -1's, so I can detect weird cases in which

# -1 is in the middle of the vector instead

neg\_ends <- function(hit\_thetas) {

p <- length(hit\_thetas)

nends <- rep(FALSE, length(hit\_thetas))

i <- 1

while (hit\_thetas[i] < 0 && i <= p) {

nends[i] <- TRUE

i <- i + 1

}

if (i == p) { # if it went through entire vector already

return(nends)

}

# go from the other side

other\_i <- p

while(hit\_thetas[other\_i] < 0 && nends[other\_i] == FALSE && other\_i >= 1) {

nends[other\_i] <- TRUE

other\_i <- other\_i - 1

}

return(nends)

}

#Function which generates the observed data for a single theta vector

data\_gen <- function(theta, d, l){

#Theta is the given theta vector used to compute attenutation probs.

#d is mean of initial Poisson distribution

#l is the length of each projection through each pixel

p = length(theta)

if(p<=0){#If empty theta vector is given, return NA

return(NA)

}

X1 = rpois(1, lambda = d) #Initial number of counts from Poisson dist.

X = c(X1, rep(0, p)) #Vector to hold the number of counts after each interaction

for(j in 1:p){

X[j+1] <- rbinom(1, size = X[j], prob = exp(-theta[j] \* l))

#Generate the number of photons from Binom(X\_i, exp(-theta\_i \* l))

}

Y = X[p+1] #Actually observed data (number of counts)

return(Y)

}

#' Function which generates the observed data for entire theta matrix and arbitrary projections

#'

#' @param THETA input theta matrix used to compute probs.

#' @param d Poisson mean for initial Poisson generation

#' @param ROW rows which we project on (vector) Enter negative row number to get opposite direction

#' @param COL columns which we project on (vector) Enter negative col number to get opposite direction

#' @param reps number of times to run the function

#' @param rise\_vec vector of numerator(s) of the slopes of the parallel projections (projections perpendicular to them will

#' also be generated)

#' @param run\_vec vector of denominator(s) of the slopes of the parallel projections (projections perpendicular to them will

#' also be generated)

#' Note: One of rise or run is assumed to be one, and the other is assumed to be an integer >= one.

#' @return estimated theta values

#' @export

data\_gen\_df <- function(THETA, d, ROW, COL, reps=1, rise\_vec = 1, run\_vec = 1){

r = dim(THETA)[1] #Number of rows in THETA

c = dim(THETA)[2] #Number of cols in THETA

#Check that ROW and COL projection indices are valid

if(max(abs(ROW))> r || max(abs(COL)) > c){

return(simpleError("Row and/or column indices out of range."))

}

#Return projection list

#with d, length of projection within each pixel, indices (in order) beam went through, counts

proj.list <- vector("list", 1)

pl\_idx <- 1

for(a in 1:reps){

#Run over ROWS first

for(rr in ROW){

if(rr > 0){

y <- data\_gen(THETA[rr,THETA[rr,]>=0], d, l = 1)

#Matrix indices that this beam goes through

idx = seq(rr, rr+(c-1)\*r, r)

idx = idx[which(THETA[rr,]>=0, arr.ind=TRUE)] #Drop indices with negative thetas

}else{ #switch order of theta if row number is negative

y <- data\_gen(rev(THETA[-(rr),THETA[-(rr),]>=0]), d, l = 1)

#Matrix indices that this beam goes through

idx = seq(-rr, -rr+(c-1)\*r, r)

idx = rev(idx[which(THETA[-rr,]>=0, arr.ind=TRUE)])

}

if (!is.na(y)) {

add.list <- list(d = d, l = 1, idx = idx, y = y)

proj.list[[pl\_idx]] <- add.list

pl\_idx <- pl\_idx + 1

}

}

#Run over COLS second

for(cc in COL){

if(cc >0){

y<- data\_gen(THETA[THETA[,cc]>=0,cc], d, l = 1)

idx = seq((cc-1)\*r+1, cc\*r, 1)

idx = idx[which(THETA[,cc]>=0, arr.ind = TRUE)] # Drop indices with negative thetas

}else{

y <- data\_gen(rev(THETA[THETA[,-(cc)]>=0,-(cc)]), d, l = 1)

idx = seq((-cc-1)\*r+1, -cc\*r, 1)

idx = rev(idx[which(THETA[,-cc]>=0, arr.ind=TRUE)])

}

if (!is.na(y)) {

add.list <- list(d = d, l = 1, idx = idx, y = y)

proj.list[[pl\_idx]] <- add.list

pl\_idx <- pl\_idx + 1

}

}

#Check that rise\_list and run\_list have the same lengths

if(length(rise\_vec) != length(run\_vec)){

return(simpleError("rise\_vec and run\_vec have different numbers of elements"))

}

if (length(rise\_vec) != 0) {

for (i in 1:length(rise\_vec)) {

# call a separate function for the angular projections

angular\_proj\_list <- angular\_proj\_list\_gen(THETA, d, ROW, COL, rise = rise\_vec[i], run = run\_vec[i])

proj.list <- c(proj.list, angular\_proj\_list)

pl\_idx + length(angular\_proj\_list)

}

}

}

#Return list with d, l, idx, y for each

return(proj.list)

}

#' Function to generate one set of angular projections, to be used within data\_gen\_df

#'

#' @param THETA input theta matrix used to compute probs.

#' @param d Poisson mean for initial Poisson generation

#' @param ROW rows which we project on (vector) Enter negative row number to get opposite direction

#' @param COL columns which we project on (vector) Enter negative col number to get opposite direction

#' @param rise numerator of the slope of the parallel projections (projections perpendicular to them will

#' also be generated)

#' @param run denominator of the slope of the parallel projections (projections perpendicular to them will

#' also be generated)

#' Note: One of rise or run is assumed to be one, and the other is assumed to be an integer >= one.

#' @return estimated theta values

#' @export

angular\_proj\_list\_gen <- function(THETA, d, ROW, COL, rise, run) {

r = dim(THETA)[1] #Number of rows in THETA

c = dim(THETA)[2] #Number of cols in THETA

rise\_init <- rise

run\_init <- run

# standardize the slope

rise <- rise / max(rise\_init, run\_init)

run <- run / max(rise\_init, run\_init)

# length that each projection traverses across each pixel

l <- sqrt(rise^2 + run^2)

#Return projection list

#with d, length of projection within each pixel, indices (in order) beam went through, counts

proj.list <- vector("list", 1)

pl\_idx <- 1

# 1. Projections with slope (rise/run), starting from left wall

for(rr in ROW){

if(rr > 0){

curr\_row = rr

curr\_col = 1

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= floor(curr\_col) && floor(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), floor(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), floor(curr\_col)])

curr\_row = curr\_row + rise

curr\_col = curr\_col + run

}

}

else{ # switch order of idx and theta if row number is negative

curr\_row = -rr

curr\_col = 1

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= floor(curr\_col) && floor(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), floor(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), floor(curr\_col)])

curr\_row = curr\_row + rise

curr\_col = curr\_col + run

}

idx <- rev(idx)

hit\_thetas <- rev(hit\_thetas)

}

idx = idx[!neg\_ends(hit\_thetas)] #Drop indices with negative thetas

hit\_thetas <- hit\_thetas[!neg\_ends(hit\_thetas)]

if(length(idx) != 0 && sum(hit\_thetas < 0) == 0){

# if the projection didn't only go through -1's

# and if there isn't a -1 in the middle of the vector

y <- data\_gen(hit\_thetas, d, l = l)

add.list <- list(d = d, l = l, idx = idx, y = y)

proj.list[[pl\_idx]] <- add.list

pl\_idx <- pl\_idx + 1

}

}

# 2. Projections with slope (-run/rise), starting from left wall

for(rr in ROW){

if(rr > 0){

curr\_row = rr

curr\_col = 1

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= ceiling(curr\_col) && ceiling(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), ceiling(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), ceiling(curr\_col)])

curr\_row = curr\_row - run

curr\_col = curr\_col + rise

}

}

else{ # switch order of idx and theta if row number is negative

curr\_row = -rr

curr\_col = 1

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= ceiling(curr\_col) && ceiling(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), ceiling(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), ceiling(curr\_col)])

curr\_row = curr\_row - run

curr\_col = curr\_col + rise

}

idx <- rev(idx)

hit\_thetas <- rev(hit\_thetas)

}

idx = idx[!neg\_ends(hit\_thetas)] #Drop indices with negative thetas

hit\_thetas <- hit\_thetas[!neg\_ends(hit\_thetas)]

if(length(idx) != 0 && sum(hit\_thetas < 0) == 0){

# if the projection didn't only go through -1's

# and if there isn't a -1 in the middle of the vector

y <- data\_gen(hit\_thetas, d, l = l)

add.list <- list(d = d, l = l, idx = idx, y = y)

proj.list[[pl\_idx]] <- add.list

pl\_idx <- pl\_idx + 1

}

}

# 3. Projections with slope (rise/run), starting from bottom wall

for(cc in COL[abs(COL) != 1]){ # skipping one to avoid redundancy

if(cc > 0){

curr\_row = 1

curr\_col = cc

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= floor(curr\_col) && floor(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), floor(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), floor(curr\_col)])

curr\_row = curr\_row + rise

curr\_col = curr\_col + run

}

}

else{ # switch order of idx and theta if col number is negative

curr\_row = 1

curr\_col = -cc

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= floor(curr\_col) && floor(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), floor(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), floor(curr\_col)])

curr\_row = curr\_row + rise

curr\_col = curr\_col + run

}

idx <- rev(idx)

hit\_thetas <- rev(hit\_thetas)

}

idx = idx[!neg\_ends(hit\_thetas)] #Drop indices with negative thetas

hit\_thetas <- hit\_thetas[!neg\_ends(hit\_thetas)]

if(length(idx) != 0 && sum(hit\_thetas < 0) == 0){

# if the projection didn't only go through -1's

# and if there isn't a -1 in the middle of the vector

y <- data\_gen(hit\_thetas, d, l = l)

add.list <- list(d = d, l = l, idx = idx, y = y)

proj.list[[pl\_idx]] <- add.list

pl\_idx <- pl\_idx + 1

}

}

# 4. Projections with slope (-run/rise), starting from top wall

for(cc in COL[abs(COL) != 1]){ # skipping one to avoid redundancy

if(cc > 0){

curr\_row = r

curr\_col = cc

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= ceiling(curr\_col) && ceiling(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), ceiling(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), ceiling(curr\_col)])

curr\_row = curr\_row - run

curr\_col = curr\_col + rise

}

}

else{ # switch order of idx and theta if col number is negative

curr\_row = r

curr\_col = -cc

#Matrix indices that this beam goes through

idx = c()

# thetas intersected

hit\_thetas = c()

# while still in bounds of the scan

while((1 <= floor(curr\_row) && floor(curr\_row) <= r) &&

(1 <= ceiling(curr\_col) && ceiling(curr\_col) <= c)){

idx = c(idx, calc\_index(r, floor(curr\_row), ceiling(curr\_col)))

hit\_thetas = c(hit\_thetas, THETA[floor(curr\_row), ceiling(curr\_col)])

curr\_row = curr\_row - run

curr\_col = curr\_col + rise

}

idx <- rev(idx)

hit\_thetas <- rev(hit\_thetas)

}

idx = idx[!neg\_ends(hit\_thetas)] #Drop indices with negative thetas

hit\_thetas <- hit\_thetas[!neg\_ends(hit\_thetas)]

if(length(idx) != 0 && sum(hit\_thetas < 0) == 0){

# if the projection didn't only go through -1's

# and if there isn't a -1 in the middle of the vector

y <- data\_gen(hit\_thetas, d, l = l)

add.list <- list(d = d, l = l, idx = idx, y = y)

proj.list[[pl\_idx]] <- add.list

pl\_idx <- pl\_idx + 1

}

}

return(proj.list)

}

############### Expectation Maximization for Transmission Tomography ###############

#' Calculates mij for the E step

#'

#' @param proj list of information for this projection

#' @param theta matrix of initial theta values

#' @param j pixel of interest

#' @return mij

#' @export

mij <- function(proj, theta, j) {

d <- proj$d

idx <- proj$idx

y <- proj$y

l <- proj$l

if (j %in% proj$idx) {

k <- which(proj$idx == j)

} else {

return(0)

}

if (k > 1) {

idx\_sub <- idx[1:(k - 1)]

theta\_sub <- theta[idx\_sub]

} else {

theta\_sub <- 0

}

return(d \* (exp(-sum(l \* theta\_sub)) - exp(-sum(l \* theta[idx]))) + y)

}

#' Calculates nij for the E step

#'

#' @param proj list of information for this projection

#' @param theta matrix of initial theta values

#' @param j pixel of interest

#' @return nij

#' @export

nij <- function(proj, theta, j) {

d <- proj$d

idx <- proj$idx

y <- proj$y

l <- proj$l

if (j %in% proj$idx) {

k <- which(proj$idx == j)

} else {

return(0)

}

idx\_sub <- idx[1:k]

theta\_sub <- theta[idx\_sub]

return(d \* (exp(-sum(l \* theta\_sub)) - exp(-sum(l \* theta[idx]))) + y)

}

#' Maximizes Q function for pixel j to estimate theta j

#'

#' @param theta\_j theta for pixel j to optimize for

#' @param proj\_list list of projections

#' @param theta matrix of initial theta values

#' @param j pixel of interest

#' @return q function value for theta j

#' @export

q\_fun\_j <- function(thetaj, proj\_list, theta, j) {

num\_proj <- length(proj\_list)

val <- 0

for (i in 1:num\_proj) {

l <- proj\_list[[i]]$l

m\_exp <- mij(proj\_list[[i]], theta, j)

n\_exp <- nij(proj\_list[[i]], theta, j)

val <- val - n\_exp \* l + (m\_exp - n\_exp) \* l / (exp(l \* thetaj) - 1)

}

return(val)

}

#' Detects whether there are any projections with y = 0 in proj\_list

#'

#' @param proj\_list list of projections

#' @return TRUE/FALSE depending if a zero was detected or not

#' @export

y\_zero <- function(proj\_list) {

has\_zero <- FALSE

for (proj in proj\_list) {

if (proj$y == 0) {

has\_zero <- TRUE

}

}

return(has\_zero)

}

# To bypass the theta = zero error

max\_q\_fun\_j <- function(interval, proj\_list, theta, j) {

if (theta[j] == 0) {

return(0)

}

e <- try(

# if I need more efficiency, calculate nij and mij outside of function

theta\_est <- uniroot(q\_fun\_j, interval, proj\_list, theta, j, extendInt = "downX")$root,

silent = TRUE

)

if (class(e) == "try-error") {

return(0)

} else {

return(theta\_est)

}

}

#' EM algorithm for transmission tomography

#'

#' @param proj\_list list of projections

#' @param theta matrix of initial theta values. Assumes the -1's in the matrix refer to dead space to

#' be ignored

#' @param tol tolerance used for the stopping rule

#' @return estimated theta values

#' @export

em\_alg <- function(proj\_list, theta, tol) {

if(y\_zero(proj\_list)){

return(simpleError("There's a projection with zero photons observed."))

}

theta\_est <- matrix(-1, nrow = nrow(theta), ncol = ncol(theta))

# indices of theta\_est that are nonnegative

nonneg\_idx <- which(theta >= 0)

ctr <- 0

diff <- Inf

while (diff > tol & ctr <= 100000) {

for (j in nonneg\_idx) {

theta\_est[j] <- max\_q\_fun\_j(interval = c(0.0000001, 10), proj\_list, theta, j)

}

diff <- sum((theta\_est - theta)^2)

theta <- theta\_est

ctr <- ctr + 1

}

return(list(theta\_est = theta\_est, ctr = ctr))

}

#################### Construct True Three Circles Theta Matrices ####################

# radius 3

radius = 3

set.seed(316)

in\_circ = in\_circle(radius)

# add two circles

circle\_theta1 = circle\_pattern(in\_circ, 1, 4, 7)

circle\_theta2 = circle\_pattern(circle\_theta1, 1, 8, 8)

true\_theta = circle\_pattern(circle\_theta2, 1, 7, 4)

plot\_matrix(true\_theta)

# check the maximum thetas

summary(as.vector(true\_theta))

save(true\_theta, file = "true\_theta/three\_circles\_rad3.RData")

# EM algorithm, 10 seconds

# radius 5

radius = 5

set.seed(339)

in\_circ = in\_circle(radius)

plot\_matrix(in\_circ)

# add two circles

circle\_theta1 = circle\_pattern(in\_circ, 2, 10, 11)

circle\_theta2 = circle\_pattern(circle\_theta1, 2, 5, 8)

true\_theta = circle\_pattern(circle\_theta2, 1, 11, 6)

plot\_matrix(true\_theta)

# check the maximum thetas

summary(as.vector(true\_theta))

save(true\_theta, file = "true\_theta/three\_circles\_rad5.RData")

# EM algorithm: 2 minutes

# radius 10

radius = 10

set.seed(348)

in\_circ = in\_circle(radius)

plot\_matrix(in\_circ)

# add two circles

circle\_theta1 = circle\_pattern(in\_circ, 4, 16, 17)

circle\_theta2 = circle\_pattern(circle\_theta1, 3, 8, 11)

true\_theta = circle\_pattern(circle\_theta2, 2, 18, 9)

plot\_matrix(true\_theta)

# check the maximum thetas

summary(as.vector(true\_theta))

save(true\_theta, file = "true\_theta/three\_circles\_rad10.RData")

# EM algorithm: 37.78225 minutes

# 36.80142 minutes 2nd time through. So having extendInt = "downX" doesn't change things

###################### Run the Three Circles Simulation Study ######################

#Set the names for the array

names.radius = c("rad3", "rad5", "rad10")

names.angles = c("none", "deg45", "deg.all")

names.met = c("RMSE", "Spectral", "Iterations")

names.N = c("N1", "N2", "N3", "N4", "N5", "N6", "N7", "N8", "N9", "N10")

names.list = list(names.radius, names.angles, names.met, names.N)

# store the seeds

seed\_vec <- rep(NA, length(names.radius) \* length(names.angles) \* length(names.N))

loop\_idx <- 1

#Array to hold all of our results

#Rename to match your circle\_theta name

three\_circles\_results <- array(NaN, dim = c(3,3,3,10), dimnames = names.list)

#Output is a multi-dimensional array

#Dim 1: Radius. Length=3 for radius=c(3,5,10)

#Dim 2: Angles Length=3 for angles: none, 45 deg and all three

#Dim 3: Metrics. Length=3 for metric = c('RMSE', 'Spectral', 'Iterations')

#Dim 4: Monte Carlo N. length=N=10

#Thus, we will have an 3x5x3x10 array (=450 data points)

#Set the radius and rep numbers to loop over

radius.seq = c(3,5,10)

rise.list = list(c(), c(1), c(1,2,1))

run.list = list(c(), c(1), c(1,1,2))

a = 1

d = 1e9

##--------------------------------------------------------------

##--------------------------------------------------------------

for(radius in radius.seq){

#Keeps track of progress

print(paste("radius=",radius))

#Generating the three\_circles\_theta matrix. Different for each of us

load(paste("true\_theta/three\_circles\_rad",radius,".RData",sep=""))

three\_circles\_theta = true\_theta

# iterate over angles

for(b in 1:3){

print(paste("angles\_ind =",b))

for(N in 1:10){

print(paste("N=",N))

# generating observations

seed <- as.numeric(ceiling(proc.time()[3]))

seed\_vec[loop\_idx] <- seed

set.seed(seed)

# vector of boundary rows/columns that aren't solely negative space

# this code will not be right if there's more than one layer of -1's

bounds <- 1:nrow(three\_circles\_theta)

#Generate data

proj\_list <- data\_gen\_df(three\_circles\_theta, d = d, ROW = bounds,

COL = bounds,

rise\_vec = rise.list[[b]], run\_vec = run.list[[b]])

#Check for no 0 values. If so, regenerate data

while(y\_zero(proj\_list)){

proj\_list <- data\_gen\_df(three\_circles\_theta, d = d, ROW = bounds,

COL = bounds,

rise\_vec = rise.list[[b]], run\_vec = run.list[[b]])

}

# how many nonnegative numbers are in three\_circles\_theta?

num\_pixel <- sum(three\_circles\_theta >= 0)

# copy the true three\_circles\_theta's negative space, but change the nonnegative

# values randomly

three\_circles\_theta\_init <- three\_circles\_theta

three\_circles\_theta\_init[which(three\_circles\_theta >= 0)] <- runif(num\_pixel, 0, 0.1)

#Run Algorithm

em\_res <- em\_alg(proj\_list, three\_circles\_theta\_init, .0001)

save(em\_res, file=paste("em\_results/three\_circles/three\_circles\_radius",radius,"\_numangles", b,"\_N", N ,".RData",sep=""))

abs\_diff\_mat <- abs(em\_res$theta\_est - three\_circles\_theta)

sq\_diff\_mat <- abs\_diff\_mat^2

#plot\_matrix(abs\_diff\_mat)

#plot\_matrix(sq\_diff\_mat)

# RMSE:

three\_circles\_results[a,b,1,N] <- sqrt(sum(sq\_diff\_mat) / num\_pixel)

#Spectral Norm

three\_circles\_results[a,b,2,N] <-svd(em\_res$theta\_est - three\_circles\_theta)$d[1]

#Number of iterations

three\_circles\_results[a,b,3,N] <-em\_res$ctr

loop\_idx <- loop\_idx + 1

}

}

a = a+1

}

save(three\_circles\_results, file = "three\_circles\_results.RData")

save(seed\_vec, file = "seed\_vec\_three\_circles\_results.RData")

#################### Plot Figures and Results for Three Circles ####################

# plotting figures for three circles, radius 5, 0 angles

load(paste("true\_theta/three\_circles\_rad",5,".RData",sep=""))

load(paste("em\_results/three\_circles/three\_circles\_radius",5,"\_numangles", 1,"\_N", 8 ,".RData",sep=""))

plot\_matrix(true\_theta)

plot\_matrix(em\_res$theta\_est)

abs\_diff\_mat <- abs(em\_res$theta\_est - true\_theta)

plot\_matrix(abs\_diff\_mat)

# plotting figures for three circles, radius 5, 2 angles

load(paste("true\_theta/three\_circles\_rad",5,".RData",sep=""))

load(paste("em\_results/three\_circles/three\_circles\_radius",5,"\_numangles", 2,"\_N", 10 ,".RData",sep=""))

plot\_matrix(true\_theta)

plot\_matrix(em\_res$theta\_est)

abs\_diff\_mat <- abs(em\_res$theta\_est - true\_theta)

plot\_matrix(abs\_diff\_mat)

# plotting figures for three circles, radius 5, 6 angles

load(paste("true\_theta/three\_circles\_rad",5,".RData",sep=""))

load(paste("em\_results/three\_circles/three\_circles\_radius",5,"\_numangles", 3,"\_N", 7 ,".RData",sep=""))

plot\_matrix(true\_theta)

plot\_matrix(em\_res$theta\_est)

abs\_diff\_mat <- abs(em\_res$theta\_est - true\_theta)

plot\_matrix(abs\_diff\_mat)

##################

# plotting figures for three circles, radius 10, 6 angles

load(paste("true\_theta/three\_circles\_rad",10,".RData",sep=""))

load(paste("em\_results/three\_circles/three\_circles\_radius",10,"\_numangles", 3,"\_N", 7 ,".RData",sep=""))

plot\_matrix(true\_theta)

plot\_matrix(em\_res$theta\_est)

abs\_diff\_mat <- abs(em\_res$theta\_est - true\_theta)

plot\_matrix(abs\_diff\_mat)

load("three\_circles\_results.RData")

#Output is a mulit-dimensional array

#Dim 1: Radius. Length=3 for radius=c(3,5,10)

#Dim 2: Angles Length=3 for angles: none, 45 deg and all three

#Dim 3: Metrics. Length=3 for metric = c('RMSE', 'Spectral', 'Iterations')

#Dim 4: Monte Carlo N. length=N=10

#Thus, we willl have an 3x5x3x10 array (=450 data points)

names.angles = c("none", "deg45", "deg.all")

names.radius = c("rad3", "rad5", "rad10")

names.list = list(names.angles, names.radius)

rmse\_mean\_mat <- matrix(NA, nrow = 3, ncol = 3, dimnames = names.list)

spec\_mean\_mat <- matrix(NA, nrow = 3, ncol = 3, dimnames = names.list)

iter\_mean\_mat <- matrix(NA, nrow = 3, ncol = 3, dimnames = names.list)

rmse\_sd\_mat <- matrix(NA, nrow = 3, ncol = 3, dimnames = names.list)

spec\_sd\_mat <- matrix(NA, nrow = 3, ncol = 3, dimnames = names.list)

iter\_sd\_mat <- matrix(NA, nrow = 3, ncol = 3, dimnames = names.list)

# iterate over angles

for(b in 1:3){

# iterate over radii

for(a in 1:3){

rmse\_mean\_mat[b, a] <- mean(three\_circles\_results[a, b, 1, ])

spec\_mean\_mat[b, a] <- mean(three\_circles\_results[a, b, 2, ])

iter\_mean\_mat[b, a] <- mean(three\_circles\_results[a, b, 3, ])

rmse\_sd\_mat[b, a] <- sd(three\_circles\_results[a, b, 1, ])

spec\_sd\_mat[b, a] <- sd(three\_circles\_results[a, b, 2, ])

iter\_sd\_mat[b, a] <- sd(three\_circles\_results[a, b, 3, ])

}

}

round(rmse\_mean\_mat, digits = 2)

round(rmse\_sd\_mat, digits = 2)