Appendix K

R Function

K.1 TestPattern

In this appendix, some liberties have been taken with format. Titles match code names, which are case sensitive. To be readable, the code is structured. To fit, the structured code is in landscape orientation and single spaced. N.B.: The following code reflects the content of the R package. To run the code outside of the R package, paste the code into an R GUI taking care to correct double quotes “ as necessary. Sometimes the double quotes in Word are misinterpreted in the R GUI, e.g.,

```
> plot(krig2$x, krig2$y, ty=”n”, xlab=””, ylab=””, xlim=c(5, 8), ylim=c(5, 8), asp=1)
Error: unexpected input in "plot(krig2$x, krig2$y, ty=””
```

.

```
TestPattern <- function()
{
	x <- seq(-1,1,length = 251)
	y <- x
	x <- rep(x, 251)
	y <- rep(y, ea = 251)
	dir <- atan2(y,x)
	dir[x == 0] <- NA
	dir[y == 0] <- NA
	u <- cos(dir)
	v <- sin(dir)
	return(as.data.frame(list(x = x, y = y, u = u, v = v)))
}
```

K.2 CircDataimage

The function CircDataimage embeds jpegs in the R package into the GUI. To paste this function into an R GUI, remove the green highlighted/bolded code on this page and on the 14th through 16th pages of this code. However, it is recommended that the R package CircSpatial be installed for examination. The Tcl path statements must match the user installation path of Active State Tcl from http://downloads.activestate.com/ActiveTcl/Windows/

```
CircDataimage <- function()
{
    # 2008-11-12.1919
    require(tcltk, quietly=TRUE, warn.conflicts=TRUE)
    require(fields, quietly=TRUE, warn.conflicts=TRUE)

    # Sys.setenv("TCL_LIBRARY"="C:/Tcl/lib/tcl8.5")
    # Sys.setenv("MY_TCLTK"="Yes")
    # addTclPath(path = "C:/Tcl/lib/teapot/package/win32-ix86/lib")
    # tclRequire("img::jpeg")

    # Make color wheel data
    x <- seq(-1,1,length=201) # x must be consistent with next image statement
    y <- x
    x2 <- rep(x, 201)
    y2 <- rep(y, ea=201)
    dir <- atan2(y2,x2)
    dir[dir<0] <- dir[dir<0] + 2*pi # Directions in [0, 2*pi)
    Dist <- sqrt(x2^2 + y2^2) # distance from origin
    filter <- Dist > 1
    dir[filter] <- NA
    wheel <- matrix(data=dir, nrow=201, ncol=201, byrow=FALSE)

    # Make FirstColorVector
    Angles1 <- 0:89
    Angles2 <- 90:179
    Angles3 <- 180:269
    Angles4 <- 270:359
    Dist1 = 255*Angles1/90
```

```
	Dist2 = 255*(Angles2-90)/90
	Dist3 = 255*(Angles3-180)/90
	Dist4 = 255*(Angles4-270)/90
	Q1 <- rgb(0, 255-Dist1, Dist1, maxColorValue=255)
	Q2 <- rgb(Dist2, Dist2, 255-Dist2, maxColorValue=255)
	Q3 <- rgb(255, 255-Dist3, 0, maxColorValue=255)
	Q4 <- rgb(255-Dist4, Dist4, 0, maxColorValue=255)
	FirstColorVector <- c(Q1,Q2,Q3,Q4) # GBYR
	if(is.null(dev.list())) dev.image=2 else dev.image= max(dev.list()) + 1
	dev.wheel = dev.image + 1
	windows() # device dev.image
	windows(width = 1.15, height = 1, pointsize = 7) # device 3, width so menu bar on one row
	# Current device is dev.wheel
	par(plt=c(0.03,0.97,0.03,0.97)) # Applies to current device, min margin between labels and window
	angles=seq(0, 315, by=45)
	plot(x=1.2*cos(angles*pi/180), y=1.2*sin(angles*pi/180), type="n", asp=1, xaxt="n", yaxt="n", xlab="", ylab="",
		bty="n")
	text(x=1.2*cos(angles*pi/180), y=1.2*sin(angles*pi/180), labels=as.character(angles))
	image(x, y, z= wheel, col= FirstColorVector, add=TRUE)
	CircDataimageGlobals <<- list() # This erases content of CircDataimageGlobals if it exists from previous session
####################################################################################################
R1.Prime <- function()
{
	#2007-08-20.1347
	# The following global variables are not dependent on data
	R1.WriteBinColorVectors() # 360 elements for each color vector
	CircDataimageGlobals$ColorGap <<- 0
	R1.WriteContColorVectors() # Must come after CircDataimageGlobals$ColorGap

	CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$GBYR # Must come after R1.WriteContColorVectors
	CircDataimageGlobals$ColorVector   <<- CircDataimageGlobals$GBYR
	CircDataimageGlobals$ColorVectorID <<- 1
	CircDataimageGlobals$ColorRotation <<- 0
	CircDataimageGlobals$PlotArrows <<- FALSE
	CircDataimageGlobals$ArrowAdj <<- 1
	CircDataimageGlobals$cpa <<- 15
```

```
        CircDataimageGlobals$Mask <<- NULL
        CircDataimageGlobals$PlotMask <<- FALSE
}
#########################################################################################################
R1.WriteBinColorVectors <- function()
{
        # 2007-08-04.1103
        # Each vector has 360 elements for each of 360 degrees (º)
        Q1 <-  rep(rgb(  0, 235,  35, maxColorValue=255), 18)
        Q2 <-  rep(rgb(  0, 245,   0, maxColorValue=255), 18)
        Q3 <-  rep(rgb(102, 250,   0, maxColorValue=255), 18)
        Q4 <-  rep(rgb(153, 255,   0, maxColorValue=255), 18)
        Q5 <-  rep(rgb(204, 255,   0, maxColorValue=255), 18)
        Q6 <-  rep(rgb(255, 255,   0, maxColorValue=255), 18)
        Q7 <-  rep(rgb(255, 204,   0, maxColorValue=255), 18)
        Q8 <-  rep(rgb(255, 153,   0, maxColorValue=255), 18)
        Q9 <-  rep(rgb(255, 102,   0, maxColorValue=255), 18)
        Q10 <- rep(rgb(255,   0,   0, maxColorValue=255), 18)
        Q11 <- rep(rgb(230,   0,  20, maxColorValue=255), 18)
        Q12 <- rep(rgb(195,   0,  51, maxColorValue=255), 18)
        Q13 <- rep(rgb(153,   0, 102, maxColorValue=255), 18)
        Q14 <- rep(rgb(102,   0, 153, maxColorValue=255), 18)
        Q15 <- rep(rgb(  0,   0, 150, maxColorValue=255), 18)
        Q16 <- rep(rgb( 51,   0, 175, maxColorValue=255), 18)
        Q17 <- rep(rgb(  0,  51, 204, maxColorValue=255), 18)
        Q18 <- rep(rgb(  0, 102, 153, maxColorValue=255), 18)
        Q19 <- rep(rgb(  0, 153, 102, maxColorValue=255), 18)
        Q20 <- rep(rgb(  0, 204,  51, maxColorValue=255), 18)
        CircDataimageGlobals$Rainbow.20Bin <<- c(Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15, Q16, Q17, Q18, Q19, Q20)

        Q1 <-  rep(rgb(  0,   0, 128, maxColorValue=255), 30)
        Q2 <-  rep(rgb(  0,   0, 192, maxColorValue=255), 30)
        Q3 <-  rep(rgb(  0,   0, 255, maxColorValue=255), 30)
        Q4 <-  rep(rgb(128, 128, 255, maxColorValue=255), 30)
        Q5 <-  rep(rgb(192, 192, 255, maxColorValue=255), 30)
        Q6 <-  rep(rgb(255, 255, 255, maxColorValue=255), 30)
```

```
Q7 <-  rep(rgb(255, 219, 219, maxColorValue=255), 30)
Q8 <-  rep(rgb(255, 128, 128, maxColorValue=255), 30)
Q9 <-  rep(rgb(255,   0,   0, maxColorValue=255), 30)
Q10 <- rep(rgb(192,   0,  32, maxColorValue=255), 30)
Q11 <- rep(rgb(130,   0,   0, maxColorValue=255), 30)
Q12 <- rep(rgb(  0,   0,   0, maxColorValue=255), 30)
CircDataimageGlobals$KBWR.12Bin <<- c(Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12)

Q1 <-  rep(rgb(103,   0,  31, maxColorValue=255), 36)
Q2 <-  rep(rgb(178,  24,  43, maxColorValue=255), 36)
Q3 <-  rep(rgb(214,  96,  77, maxColorValue=255), 36)
Q4 <-  rep(rgb(244, 165, 130, maxColorValue=255), 36)
Q5 <-  rep(rgb(253, 219, 199, maxColorValue=255), 36)
Q6 <-  rep(rgb(224, 224, 224, maxColorValue=255), 36)
Q7 <-  rep(rgb(186, 186, 186, maxColorValue=255), 36)
Q8 <-  rep(rgb(135, 135, 135, maxColorValue=255), 36)
Q9 <-  rep(rgb( 77,  77,  77, maxColorValue=255), 36)
Q10 <- rep(rgb( 64,  13,  28, maxColorValue=255), 36)
CircDataimageGlobals$Brewer10Div6.10Bin <<- c(Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10)

Q1 <- rep(rgb(255,     255,      0,       maxColorValue=255),23) # yellow
Q2 <- rep(rgb(255,     255*0.65, 0,       maxColorValue=255),45) # orange
Q3 <- rep(rgb(255,     0,        0,       maxColorValue=255),45) # red
Q4 <- rep(rgb(255*0.75, 0,       0,       maxColorValue=255),45) # dark red
Q5 <- rep(rgb(0,       255,      0,       maxColorValue=255),45) # green
Q6 <- rep(rgb(0,       255*0.6,  0,       maxColorValue=255),45) # dark green
Q7 <- rep(rgb(0,       255*.75,  255,     maxColorValue=255),45) # blue
Q8 <- rep(rgb(0,       0,        255,     maxColorValue=255),45) # dark blue
Q9 <- rep(rgb(255,     255,      0,       maxColorValue=255),22) # yellow
CircDataimageGlobals$YRGB.8Bin <<- c(Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9)

Q1 <-  rep(rgb(140,  81,  10, maxColorValue=255), 45)
Q2 <-  rep(rgb(191, 129,  45, maxColorValue=255), 45)
Q3 <-  rep(rgb(223, 194, 125, maxColorValue=255), 45)
Q4 <-  rep(rgb(246, 232, 195, maxColorValue=255), 45)
Q5 <-  rep(rgb(199, 234, 229, maxColorValue=255), 45)
```

```
	Q6 <-  rep(rgb(128, 205, 193, maxColorValue=255), 45)
	Q7 <-  rep(rgb( 53, 151, 143, maxColorValue=255), 45)
	Q8 <-  rep(rgb(  1, 102,  94, maxColorValue=255), 45)
	CircDataimageGlobals$Brewer8Div2.8Bin <<- c(Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8)

	Q1 <-  rep(rgb(255, 0,   0, maxColorValue=255), 60) # red
	Q2 <-  rep(rgb(255, 0,   255, maxColorValue=255), 60) # magenta
	Q3 <-  rep(rgb(0,   0,   255, maxColorValue=255), 60) # blue
	Q4 <-  rep(rgb(0,   255, 0, maxColorValue=255), 60) # green
	Q5 <-  rep(rgb(255, 255, 0, maxColorValue=255), 60) # yellow
	Q6 <-  rep(rgb(255, 165, 0, maxColorValue=255), 60) # orange
	CircDataimageGlobals$RMBGYO.6Bin <<- c(Q1, Q2, Q3, Q4, Q5, Q6)
}
########################################################################################################
R1.WriteContColorVectors <- function()
{
	# 2007-09-11.1943
	# Each vector has 360 elements for each of 360º

	gap <- CircDataimageGlobals$ColorGap
	Angles1 <- 0:89
	Angles2 <- 90:179
	Angles3 <- 180:269
	Angles4 <- 270:359
	Dist1 = (1-gap)*255*Angles1/90
	Dist2 = (1-gap)*255*(Angles2-90)/90
	Dist3 = (1-gap)*255*(Angles3-180)/90
	Dist4 = (1-gap)*255*(Angles4-270)/90
	Q1 <- rgb(0, 255-Dist1, Dist1, maxColorValue=255)
	Q2 <- rgb(Dist2, Dist2, 255-Dist2, maxColorValue=255)
	Q3 <- rgb(255, 255-Dist3, 0, maxColorValue=255)
	Q4 <- rgb(255-Dist4, Dist4, 0, maxColorValue=255)
	CircDataimageGlobals$GBYR <<- c(Q1,Q2,Q3,Q4)

	Q1 <- rgb(Dist1, 255, 0, maxColorValue=255)
	Q2 <- rgb(255, 255-Dist2, 0, maxColorValue=255)
```

```
Q3 <- rgb(255-Dist3, 0, Dist3, maxColorValue=255)
Q4 <- rgb(0, Dist4, 255-Dist4, maxColorValue=255)
CircDataimageGlobals$GYRB <<- c(Q1,Q2,Q3,Q4)
Q1 <- rgb(Dist1, 255, 0, maxColorValue=255)
Q2 <- rgb(255-Dist2, 255-Dist2, Dist2, maxColorValue=255)
Q3 <- rgb(Dist3, 0, 255-Dist3, maxColorValue=255)
Q4 <- rgb(255-Dist4, Dist4, 0, maxColorValue=255)
CircDataimageGlobals$GYBR <<- c(Q1,Q2,Q3,Q4)

Q1 <- rgb(0, 0, Dist1, maxColorValue=255)
Q2 <- rgb(Dist2, Dist2, 255, maxColorValue=255)
Q3 <- rgb(255, 255-Dist3, 255-Dist3, maxColorValue=255)
Q4 <- rgb(255-Dist4, 0, 0, maxColorValue=255)
CircDataimageGlobals$KBWR <<- c(Q1,Q2,Q3,Q4)

Dist1 = (1-gap)*Angles1/360
Dist2 = 0.25 + (1-gap)*0.25*(Angles2 -90)/90
Dist3 = 0.50 + (1-gap)*0.25*(Angles3-180)/90
Dist4 = 0.75 + (1-gap)*0.25*(Angles4-270)/90
Q1 <- hsv(h=Dist1, s=0.5, v=1)
Q2 <- hsv(h=Dist2, s=0.5, v=1)
Q3 <- hsv(h=Dist3, s=0.5, v=1)
Q4 <- hsv(h=Dist4, s=0.5, v=1)
CircDataimageGlobals$HSV <<- c(Q1,Q2,Q3,Q4)

Angles1 <- 0:59
Angles2 <- 60:119
Angles3 <- 120:179
Angles4 <- 180:239
Angles5 <- 240:299
Angles6 <- 300:359
```

```
Dist1 = (1-gap)*255*Angles1/60
Dist2 = (1-gap)*255*(Angles2-60)/60
Dist3 = (1-gap)*255*(Angles3-120)/60
Dist4 = (1-gap)*255*(Angles4-180)/60
Dist5 = (1-gap)*255*(Angles5-240)/60
Dist6 = (1-gap)*255*(Angles6-300)/60
Q1 <- rgb(0, 0, Dist1, maxColorValue=255)
Q2 <- rgb(0, Dist2, 255, maxColorValue=255)
Q3 <- rgb(Dist3, 255, 255, maxColorValue=255)
Q4 <- rgb(255, 255, 255-Dist4, maxColorValue=255)
Q5 <- rgb(255, 255-Dist5, 0, maxColorValue=255)
Q6 <- rgb(255-Dist6, 0, 0, maxColorValue=255)
CircDataimageGlobals$KBCWYR <<- c(Q1, Q2, Q3, Q4, Q5, Q6)

Angles1 <- 0:59
Angles2 <- 60:119
Angles3 <- 120:179
Angles4 <- 180:239
Angles5 <- 240:299
Angles6 <- 300:359
Dist1 = (1-gap)*(Angles1-  0)/60
Dist2 = (1-gap)*(Angles2- 60)/60
Dist3 = (1-gap)*(Angles3-120)/60
Dist4 = (1-gap)*(Angles4-180)/60
Dist5 = (1-gap)*(Angles5-240)/60
Dist6 = (1-gap)*(Angles6-300)/60

Q1 <- rgb(          255,            165*Dist1,              0, maxColorValue=255)
Q2 <- rgb(          255, 165+(255-165)*Dist2,              0, maxColorValue=255)
Q3 <- rgb(255*(1-Dist3),                  255,      255*Dist3, maxColorValue=255)
Q4 <- rgb(            0,        255*(1-Dist4),            255, maxColorValue=255)
Q5 <- rgb(    255*Dist5,                    0,            255, maxColorValue=255)
Q6 <- rgb(          255,                    0, 255*(1-Dist6), maxColorValue=255)
CircDataimageGlobals$ROYBgBPb <<- c(Q1, Q2, Q3, Q4, Q5, Q6)
```

```
        Angles1 <- 0:35
        Angles2 <- 36:71
        Angles3 <- 72:107
        Angles4 <- 108:143
        Angles5 <- 144:179
        Angles6 <- 180:215
        Angles7 <- 216:251
        Angles8 <- 252:287
        Angles9 <- 288:323
        Angles10 <- 324:359
        Dist1 = (1-gap)*Angles1/36
        Dist2 = (1-gap)*(Angles2-36)/36
        Dist3 = (1-gap)*(Angles3-72)/36
        Dist4 = (1-gap)*(Angles4-108)/36
        Dist5 = (1-gap)*(Angles5-144)/36
        Dist6 = (1-gap)*(Angles6-180)/36
        Dist7 = (1-gap)*(Angles7-216)/36
        Dist8 = (1-gap)*(Angles8-252)/36
        Dist9 = (1-gap)*(Angles9-288)/36
        Dist10 = (1-gap)*(Angles10-324)/36
        Q1 <- rgb(103+(178-103)*Dist1,   0   +(24-0)*Dist1,    31+(43-31)*Dist1, maxColorValue=255)
        Q2 <- rgb(178+(214-178)*Dist2,  24  +(96-24)*Dist2,    43+(77-43)*Dist2, maxColorValue=255)
        Q3 <- rgb(214+(244-214)*Dist3,  96+(165- 96)*Dist3,   77+(130-77)*Dist3, maxColorValue=255)
        Q4 <- rgb(244+(253-244)*Dist4, 165+(219-165)*Dist4, 130+(199-130)*Dist4, maxColorValue=255)
        Q5 <- rgb(253+(224-253)*Dist5, 219+(224-219)*Dist5, 199+(224-199)*Dist5, maxColorValue=255)
        Q6 <- rgb(224+(186-224)*Dist6, 224+(186-224)*Dist6, 224+(186-224)*Dist6, maxColorValue=255)
        Q7 <- rgb(186+(135-186)*Dist7, 186+(135-186)*Dist7, 186+(135-186)*Dist7, maxColorValue=255)
        Q8 <- rgb(135 +(77-135)*Dist8, 135 +(77-135)*Dist8,  135+(77-135)*Dist8, maxColorValue=255)
        Q9 <- rgb( 77  +(64-77)*Dist9,  77  +(13-77)*Dist9,   77 +(28-77)*Dist9, maxColorValue=255)
        Q10 <- rgb(64+(103-64)*Dist10,  13  +(0-13)*Dist10,   28+(31-28)*Dist10, maxColorValue=255)
        CircDataimageGlobals$Brewer10Div6 <<- c(Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10)
}
##########################################################################################################
```

```
R1.Initialize <- function(data.name2, mask.name2, nObs.cb.value2)
{
	# 2007-09-12.1930
	# Variable name suffix ".g" indicates variable is global, i.e. at limits of data
	# Variable name suffix ".d" indicates variable has been subset for display
	# Coordinates and direction will apply at center of pixel
	InputData <- as.matrix(eval(parse(file="", text=as.character(tclvalue(data.name2))))) # Must be matrix for loop below
	mask <- as.character(tclvalue(mask.name2))
	if(mask == "unknown" | mask == "") CircDataimageGlobals$Mask <<- NULL else CircDataimageGlobals$Mask <<- as.matrix(eval(parse(file="", text=mask)))

	CircDataimageGlobals$Data <<- InputData
	x <- sort(unique(InputData[,1])) # Ascending unique horizontal coordinates of sampling locations.
	y <- sort(unique(InputData[,2])) # Ascending unique vertical coordinates of sampling locations.
	CircDataimageGlobals$MinX.g <<- min(x) # Global minimum X.
	CircDataimageGlobals$MaxX.g <<- max(x) # Global maximum X.
	CircDataimageGlobals$MinY.g <<- min(y) # Global minimum Y.
	CircDataimageGlobals$MaxY.g <<- max(y) # Global maximum Y.
	# Measurement location horizontal spacing assumed to be constant in X.
	# Grid vertical spacing assumed to be constant in Y.  Horiz and vert spacing do not have to be equal.
	CircDataimageGlobals$DX <<- x[2] - x[1] # Horizontal spacing of sampling grid.
	CircDataimageGlobals$DY <<- y[2] - y[1] # Vertical spacing of sampling grid.
	# Simple check
	DX2 <- x[3] - x[2]
	DY2 <- y[3] - y[2]
	if(DX2 != CircDataimageGlobals$DX | DY2 != CircDataimageGlobals$DY) stop("Measurement spacing not constant")
	CircDataimageGlobals$nx.g <<- round((CircDataimageGlobals$MaxX.g-CircDataimageGlobals$MinX.g)/CircDataimageGlobals$DX + 1, digits = 0)
	CircDataimageGlobals$ny.g <<- round((CircDataimageGlobals$MaxY.g-CircDataimageGlobals$MinY.g)/CircDataimageGlobals$DY + 1, digits = 0)
	CircDataimageGlobals$x.g <<- seq(from=CircDataimageGlobals$MinX.g, to=CircDataimageGlobals$MaxX.g,length=CircDataimageGlobals$nx.g)
	CircDataimageGlobals$y.g <<- seq(from=CircDataimageGlobals$MinY.g, to=CircDataimageGlobals$MaxY.g,length=CircDataimageGlobals$ny.g)

	# for display if Pan() not invoked
	CircDataimageGlobals$MinX.d <<- CircDataimageGlobals$MinX.g
	CircDataimageGlobals$MaxX.d <<- CircDataimageGlobals$MaxX.g
	CircDataimageGlobals$MinY.d <<- CircDataimageGlobals$MinY.g
	CircDataimageGlobals$MaxY.d <<- CircDataimageGlobals$MaxY.g
	# The number of rows of the matrix will be = CircDataimageGlobals$nx.g = length(CircDataimageGlobals$x.g)
	CircDataimageGlobals$StartRow <<- 1
```

```
CircDataimageGlobals$EndRow <<- CircDataimageGlobals$nx.g
CircDataimageGlobals$StartCol <<- 1
CircDataimageGlobals$EndCol <<- CircDataimageGlobals$ny.g
if(as.character(tclvalue(nObs.cb.value2)) == "0")
{
	u.g <- matrix(data=NA,nrow=CircDataimageGlobals$nx.g,ncol=CircDataimageGlobals$ny.g) # u accumulator, because atan2(0,0)=0
	v.g <- u.g # v accumulator
	Rows    <- round((InputData[, 1]- CircDataimageGlobals$MinX.g)/CircDataimageGlobals$DX + 1, digits = 0) # Indexing vector
	Columns <- round((InputData[, 2]- CircDataimageGlobals$MinY.g)/CircDataimageGlobals$DY + 1, digits = 0) # Indexing vector
	u.g[cbind(Rows, Columns)] <- InputData[, 3]
	v.g[cbind(Rows, Columns)] <- InputData[, 4]
} else
{
	cat("The initial computations necessarily may take significant time\n")
	u.g <- matrix(data=0, nrow=CircDataimageGlobals$nx.g, ncol=CircDataimageGlobals$ny.g) # u accumulator
	v.g <- u.g # v accumulator
	N.g <- u.g # Number of observations per cell
	Row    <- round((InputData[, 1]- CircDataimageGlobals$MinX.g)/CircDataimageGlobals$DX + 1, digits = 0) # Indexing scalar
	Column <- round((InputData[, 2]- CircDataimageGlobals$MinY.g)/CircDataimageGlobals$DY + 1, digits = 0) # Indexing scalar
	for (i in 1:nrow(InputData))
	{
		u.g[Row[i], Column[i]] <- u.g[Row[i], Column[i]] + InputData[i, 3]
		v.g[Row[i], Column[i]] <- v.g[Row[i], Column[i]] + InputData[i, 4]
		N.g[Row[i], Column[i]] <- N.g[Row[i], Column[i]] + 1
	}
	# Averages
	filter1 <- N.g > 0
	u.g[filter1] <- u.g[filter1]/N.g[filter1]
	v.g[filter1] <- v.g[filter1]/N.g[filter1]
	# Replace 0's with NAs where there are no observations
	u.g[!filter1] <- NA
	v.g[!filter1] <- NA
}
```

```
	CircDataimageGlobals$u.g <<- u.g # Cell contains average u or NA
	CircDataimageGlobals$v.g <<- v.g
	CircDataimageGlobals$Direction.g <<- R1.Standardize(atan2(v.g, u.g)) # atan2(NA,NA)=NA
	CircDataimageGlobals$Direction <<- CircDataimageGlobals$Direction.g
	R1.SubsetColorScale(CircDataimageGlobals$Direction[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
		CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol])
	CircDataimageGlobals$PlotMask <<- FALSE
	CircDataimageGlobals$PlotArrows <<- FALSE
	R1.PlotImage()
}
####################################################################################################
R1.Standardize <- function(Input)
{
	# 2007-08-05.1218
	# Input and Output in radians
	filter <- !is.na(Input)
	temp <- Input[filter]
	temp[temp < 0] <- temp[temp < 0] + 2*pi
	temp[temp > 2*pi] <- temp[temp > 2*pi] - 2*pi
	Input[filter] <- temp
	return(Input)
}
#######################################################################################################
R1.SubsetColorScale <- function(Input)
{
	# 2007-08-20.1331
	filter <- !is.na(Input)
	Range <- floor(range(Input[filter]*180/pi))
	a <- which((0:359) == Range[1])
	b <- which((0:359) == Range[2])
	CircDataimageGlobals$ColorFilter <<- a:b
}
#######################################################################################################
```

```
R1.PlotImage <- function()
{
	# 2007-09-14.1720
	# Composite image = data overplotted with arrows overplotted with mask
	dev.set(which=dev.image)
	# The image color vector is subset based on range of data.
	image(x= CircDataimageGlobals$x.g[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow],
		y= CircDataimageGlobals$y.g[CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol],
		z= CircDataimageGlobals$Direction[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
			CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol],
		col= CircDataimageGlobals$ColorVector[CircDataimageGlobals$ColorFilter], xlab="X", ylab="Y", asp=1)
	if(CircDataimageGlobals$PlotMask) R1.PlotMask()
	if(CircDataimageGlobals$PlotArrows) R1.PlotArrows()
}
##########################################################################################
R1.PlotArrows <- function()
{
	# 2007-09-20.2244
	x <- CircDataimageGlobals$x.g[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow]
	y <- CircDataimageGlobals$y.g[CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol]
	Directions <- CircDataimageGlobals$Direction[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
		CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol]
	nx=length(x); ny=length(y)
	x <- rep(x, ny)
	y <- rep(y, each=nx)
	Directions <- as.vector(Directions)
	filter1 <- rep(rep(CircDataimageGlobals$cpa:1, length=nx), ny) == CircDataimageGlobals$cpa
	filter2 <- as.vector(t(matrix(data=rep(rep(CircDataimageGlobals$cpa:1, length=ny), nx), nrow=ny))) == CircDataimageGlobals$cpa
	if(!CircDataimageGlobals$PlotMask | is.null(CircDataimageGlobals$Mask)) filter <- !is.na(Directions) & filter1 & filter2
	if(CircDataimageGlobals$PlotMask & !is.null(CircDataimageGlobals$Mask))
	{
		mask.boolean <- matrix(data=TRUE, nrow=CircDataimageGlobals$nx.g, ncol=CircDataimageGlobals$ny.g)
		mask.boolean[!is.na(CircDataimageGlobals$Mask)] <- FALSE
		mask.boolean <- mask.boolean[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
			CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol]
		filter <- !is.na(Directions) & filter1 & filter2 & as.vector(mask.boolean)
```

```
	}
	if(sum(filter) > 0)
	{
		x <- x[filter]; y <- y[filter]
		Directions <- Directions[filter]
		arrow.plot(x, y, u = cos(Directions), v = sin(Directions), arrow.ex = 0.05*CircDataimageGlobals$ArrowAdj,  xpd = FALSE,
			true.angle = TRUE, arrowfun=arrows, length=.05, angle=15, col=1)
	} else cat("No arrows can be displayed at current spacing\n")
}
##########################################################################################################
R1.PlotMask <- function()
{
	#2007-08-09.2013
	image(x=CircDataimageGlobals$x.g[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow],
	    y=CircDataimageGlobals$y.g[CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol],
	    z=CircDataimageGlobals$Mask[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
	        CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol], col= "tan", add=TRUE)
}
#######################################################################################################
R1.PlotWheel <- function()
{
	dev.set(which=dev.wheel)
	# The image color vector is not subset based on range of data.
	image(x=seq(-1,1,length=201), y=seq(-1,1,length=201), z= wheel, col= CircDataimageGlobals$ColorVector, add=TRUE)
}
#######################################################################################################
R1.Pan <- function()
{
	# 2007-09-11.2139
	CircDataimageGlobals$StartRow <<- round((CircDataimageGlobals$MinX.d - CircDataimageGlobals$MinX.g)/CircDataimageGlobals$DX + 1, digits=0)
	CircDataimageGlobals$EndRow   <<- round((CircDataimageGlobals$MaxX.d - CircDataimageGlobals$MinX.g)/CircDataimageGlobals$DX + 1, digits=0)
	CircDataimageGlobals$StartCol <<- round((CircDataimageGlobals$MinY.d - CircDataimageGlobals$MinY.g)/CircDataimageGlobals$DY + 1, digits=0)
	CircDataimageGlobals$EndCol   <<- round((CircDataimageGlobals$MaxY.d - CircDataimageGlobals$MinY.g)/CircDataimageGlobals$DY + 1, digits=0)
	R1.SubsetColorScale(CircDataimageGlobals$Direction[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
	    CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol])
	R1.PlotImage()
```

```
}
##########################################################################################################
R1.ChangeColorWheel <- function()
{
	# 2007-08-20.1903
	ID <- CircDataimageGlobals$ColorVectorID
	if(ID == "1")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$GBYR
	if(ID == "2")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$GYRB
	if(ID == "3")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$ROYBgBPb
	if(ID == "4")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$HSV
	if(ID == "5")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$KBWR
	if(ID == "6")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$KBCWYR
	if(ID == "7")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$Brewer10Div6
	if(ID == "8")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$Rainbow.20Bin
	if(ID == "9")  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$KBWR.12Bin
	if(ID == "10") CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$Brewer10Div6.10Bin
	if(ID == "11") CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$YRGB.8Bin
	if(ID == "12") CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$Brewer8Div2.8Bin
	if(ID == "13") CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$RMBGYO.6Bin
	R1.AutoRotateColorWheel()
	R1.PlotImage()
	R1.PlotWheel()
}
##########################################################################################################
R1.AutoRotateColorWheel <- function()
{
	# 2007-08-06.1603
	Rotation <- CircDataimageGlobals$ColorRotation
	if(Rotation > 0)
	{
		a <- (360-Rotation+1):360
		CircDataimageGlobals$ColorVector <<- c(CircDataimageGlobals$ColorVector.g[a], CircDataimageGlobals$ColorVector.g[-a])
	} else
	if(Rotation == 0) {CircDataimageGlobals$ColorVector <<- CircDataimageGlobals$ColorVector.g} else
	{CircDataimageGlobals$ColorVector <<- c(CircDataimageGlobals$ColorVector.g[-1:Rotation], CircDataimageGlobals$ColorVector.g[1:-Rotation])}
}
```

```
################################################################################################
R1.RotateColorWheel <- function()
{
	# 2007-08-20.1933
	# To return to unrotated color wheel, enter zero for rotation.
	Rotation <- CircDataimageGlobals$ColorRotation
	if(Rotation > 0)
	{
		a <- (360-Rotation+1):360
		CircDataimageGlobals$ColorVector <<- c(CircDataimageGlobals$ColorVector.g[a], CircDataimageGlobals$ColorVector.g[-a])
	} else
	if(Rotation == 0) {CircDataimageGlobals$ColorVector <<- CircDataimageGlobals$ColorVector.g} else
	{CircDataimageGlobals$ColorVector <<- c(CircDataimageGlobals$ColorVector.g[-1:Rotation], CircDataimageGlobals$ColorVector.g[1:-Rotation])}
	R1.PlotImage()
	R1.PlotWheel()
}
################################################################################################
R1.ChangeColorGap <- function()
{
	# 2007-08-11.1223
	R1.WriteContColorVectors() # Recompute with gap
	if(CircDataimageGlobals$ColorVectorID == 1)  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$GBYR
	if(CircDataimageGlobals$ColorVectorID == 2)  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$GYRB
	if(CircDataimageGlobals$ColorVectorID == 3)  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$ROYBgBPb
	if(CircDataimageGlobals$ColorVectorID == 4)  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$HSV
	if(CircDataimageGlobals$ColorVectorID == 5)  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$KBWR
	if(CircDataimageGlobals$ColorVectorID == 6)  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$KBCWYR
	if(CircDataimageGlobals$ColorVectorID == 7)  CircDataimageGlobals$ColorVector.g <<- CircDataimageGlobals$Brewer10Div6
	R1.AutoRotateColorWheel()
	R1.PlotImage()
	R1.PlotWheel()
}
################################################################################################
```

```
R1.Prime()

Top <- tktoplevel()
tkwm.geometry(Top,"565x600")
tkwm.title(Top,"Circular Dataimage")
FontHeading <- tkfont.create(family="arial", size=8, weight="bold")
FrameTop <- tkframe(Top, relief="flat", borderwidth=2)
data.name <- tclVar("unknown")
data.name.entry <-tkentry(FrameTop, width="15", textvariable=data.name)
mask.name <- tclVar("unknown")
mask.name.entry <- tkentry(FrameTop, width="15",textvariable=mask.name)
nObs.cb.value <- tclVar("0")
nObs.cb <- tkcheckbutton(FrameTop); tkconfigure(nObs.cb,variable=nObs.cb.value)
Input.but <- tkbutton(FrameTop, text="OK", command=function(){
        R1.Initialize(data.name, mask.name, nObs.cb.value)
        tclvalue(MinX.g) <- as.character(CircDataimageGlobals$MinX.g)
        tclvalue(MaxX.g) <- as.character(CircDataimageGlobals$MaxX.g)
        tclvalue(MinY.g) <- as.character(CircDataimageGlobals$MinY.g)
        tclvalue(MaxY.g) <- as.character(CircDataimageGlobals$MaxY.g)
        tclvalue(MinX) <- as.character(CircDataimageGlobals$MinX.g)
        tclvalue(MaxX) <- as.character(CircDataimageGlobals$MaxX.g)
        tclvalue(MinY) <- as.character(CircDataimageGlobals$MinY.g)
        tclvalue(MaxY) <- as.character(CircDataimageGlobals$MaxY.g)
        tclvalue(Smooth) <- "0"
        tclvalue(arrow.cb.value) <- "0"
        tclvalue(mask.cb.value) <- "0"
        }
)
tkgrid(tklabel(FrameTop, text="Input Dataframe"), data.name.entry, tklabel(FrameTop,text="     "),
tklabel(FrameTop,text="Mask Matrix"), mask.name.entry,      tklabel(FrameTop,text="     "),
tklabel(FrameTop,text="Obs Per Cell > 1"), nObs.cb,         tklabel(FrameTop,text="     "),
Input.but, sticky="w")
```

```
FrameLeft <- tkframe(Top, relief="groove", borderwidth=2)
tkgrid(tklabel(FrameLeft,text="Continuous Color Scales", font=FontHeading), sticky="e")
image1 <- tclVar(); image2 <- tclVar(); image3 <- tclVar(); image4 <- tclVar(); image5 <- tclVar(); image6 <- tclVar()
image7 <- tclVar(); image8 <- tclVar(); image9 <- tclVar(); image10 <- tclVar(); image11 <- tclVar(); image12 <- tclVar()
image13 <- tclVar()
tcl("image", "create", "photo", image1,  file=system.file("graphics", "GBYR.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image2,  file=system.file("graphics", "GYRB.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image3,  file=system.file("graphics", "ROYBgBPb.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image4,  file=system.file("graphics", "HSV.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image5,  file=system.file("graphics", "KBWR.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image6,  file=system.file("graphics", "KBCWYR.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image7,  file=system.file("graphics", "BREWER10D6.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image8,  file=system.file("graphics", "Rainbow.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image9,  file=system.file("graphics", "KBWR.12.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image10, file=system.file("graphics", "Brewer10D6.10.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image11, file=system.file("graphics", "YRGB.8.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image12, file=system.file("graphics", "Brewer8D2.8.jpeg", package="CircSpatial"))
tcl("image", "create", "photo", image13, file=system.file("graphics", "RMBGYO.6.jpeg", package="CircSpatial"))
wheel1 <-  tklabel(FrameLeft, image=image1) # Image as label
wheel2 <-  tklabel(FrameLeft, image=image2)
wheel3 <-  tklabel(FrameLeft, image=image3)
wheel4 <-  tklabel(FrameLeft, image=image4)
wheel5 <-  tklabel(FrameLeft, image=image5)
wheel6 <-  tklabel(FrameLeft, image=image6)
wheel7 <-  tklabel(FrameLeft, image=image7)
wheel8 <-  tklabel(FrameLeft, image=image8)
wheel9 <-  tklabel(FrameLeft, image=image9)
wheel10 <- tklabel(FrameLeft, image=image10)
wheel11 <- tklabel(FrameLeft, image=image11)
wheel12 <- tklabel(FrameLeft, image=image12)
wheel13 <- tklabel(FrameLeft, image=image13)
rb1 <- tkradiobutton(FrameLeft)
rb2 <- tkradiobutton(FrameLeft)
rb3 <- tkradiobutton(FrameLeft)
rb4 <- tkradiobutton(FrameLeft)
rb5 <- tkradiobutton(FrameLeft)
```

```
rb6 <- tkradiobutton(FrameLeft)
rb7 <- tkradiobutton(FrameLeft)
rb8 <- tkradiobutton(FrameLeft)
rb9 <- tkradiobutton(FrameLeft)
rb10 <- tkradiobutton(FrameLeft)
rb11 <- tkradiobutton(FrameLeft)
rb12 <- tkradiobutton(FrameLeft)
rb13 <- tkradiobutton(FrameLeft)
rbValue <- tclVar("1")
ChangeColor <- function() {CircDataimageGlobals$ColorVectorID <<- as.character(tclvalue(rbValue)); R1.ChangeColorWheel()}
tkconfigure(rb1, variable=rbValue,value="1", command=ChangeColor)
tkconfigure(rb2, variable=rbValue,value="2", command=ChangeColor)
tkconfigure(rb3, variable=rbValue,value="3", command=ChangeColor)
tkconfigure(rb4, variable=rbValue,value="4", command=ChangeColor)
tkconfigure(rb5, variable=rbValue,value="5", command=ChangeColor)
tkconfigure(rb6, variable=rbValue,value="6", command=ChangeColor)
tkconfigure(rb7, variable=rbValue,value="7", command=ChangeColor)
tkconfigure(rb8, variable=rbValue,value="8", command=ChangeColor)
tkconfigure(rb9, variable=rbValue,value="9", command=ChangeColor)
tkconfigure(rb10,variable=rbValue,value="10", command=ChangeColor)
tkconfigure(rb11,variable=rbValue,value="11", command=ChangeColor)
tkconfigure(rb12,variable=rbValue,value="12", command=ChangeColor)
tkconfigure(rb13,variable=rbValue,value="13", command=ChangeColor)
tkgrid(tklabel(FrameLeft,text="GBYR "), wheel1, rb1, sticky="e")
tkgrid(tklabel(FrameLeft,text="GYRB "), wheel2, rb2, sticky="e")
tkgrid(tklabel(FrameLeft,text="ROYBgBPb "), wheel3, rb3, sticky="e")
tkgrid(tklabel(FrameLeft,text="HSV "), wheel4, rb4, sticky="e")
tkgrid(tklabel(FrameLeft,text="KBWR "), wheel5, rb5, sticky="e")
tkgrid(tklabel(FrameLeft,text="KBCWYR "), wheel6, rb6, sticky="e")
tkgrid(tklabel(FrameLeft,text="Brewer divergent #6 "), wheel7, rb7, sticky="e")
tkgrid(tklabel(FrameLeft,text="           "), column=1)
tkgrid(tklabel(FrameLeft,text="Binned Color Scales", font=FontHeading), sticky="e")
tkgrid(tklabel(FrameLeft,text="Rainbow 20 bins "), wheel8, rb8, sticky="e")
tkgrid(tklabel(FrameLeft,text="KBWR 12 bins "), wheel9, rb9, sticky="e")
tkgrid(tklabel(FrameLeft,text="Brewer divergent #6 10 bins "), wheel10, rb10, sticky="e")
tkgrid(tklabel(FrameLeft,text="YRGB 8 bins "), wheel11, rb11, sticky="e")
```

```
tkgrid(tklabel(FrameLeft,text="Brewer divergent #2 8 bins "), wheel12, rb12, sticky="e")
tkgrid(tklabel(FrameLeft,text="RMBGYO 6 bins "), wheel13, rb13, sticky="e")
tkgrid(tklabel(FrameLeft,text="          "))
SliderValue1 <- tclVar("0")
SliderValueLabel1 <- tklabel(FrameLeft,text=as.character(tclvalue(SliderValue1)))
tkconfigure(SliderValueLabel1,textvariable=SliderValue1)
slider1 <- tkscale(FrameLeft, from=-180, to=180, showvalue=TRUE, variable=SliderValue1, resolution=5, orient="horizontal", length="1.15i")
tkbind(slider1,"<ButtonRelease-1>", function() {CircDataimageGlobals$ColorRotation <<- as.numeric(tclvalue(SliderValue1));
R1.RotateColorWheel()})
tkgrid(tklabel(FrameLeft,text="Color Scale Rotation", font=FontHeading), column=0, sticky="e")
tkgrid(slider1, column=0, sticky="e")
tkgrid(tklabel(FrameLeft,text="-180                    +180"), sticky="e")
FrameRight <- tkframe(Top, relief="groove", borderwidth=2)
tkgrid(tklabel(FrameRight, text="Display Coordinates", font=FontHeading))
MinX <- tclVar("")
MinX.entry <-tkentry(FrameRight, width="12",textvariable= MinX)
MinX.g <- tclVar("unknown")
MinX.g.label <- tklabel(FrameRight,text=tclvalue(MinX.g))
tkconfigure(MinX.g.label, textvariable=MinX.g)
tkgrid(tklabel(FrameRight,text="Min X"), MinX.entry, tklabel(FrameRight, text="Global Min  X ="), MinX.g.label, sticky="e")
MaxX <- tclVar("")
MaxX.entry <-tkentry(FrameRight, width="12",textvariable= MaxX)
MaxX.g <- tclVar("unknown")
MaxX.g.label <- tklabel(FrameRight,text=tclvalue(MaxX.g))
tkconfigure(MaxX.g.label, textvariable=MaxX.g)
tkgrid(tklabel(FrameRight,text="Max X"), MaxX.entry, tklabel(FrameRight, text="Global Max X ="), MaxX.g.label, sticky="e")
MinY <- tclVar("")
MinY.entry <-tkentry(FrameRight, width="12",textvariable= MinY)
MinY.g <- tclVar("unknown")
MinY.g.label <- tklabel(FrameRight,text=tclvalue(MinY.g))
tkconfigure(MinY.g.label, textvariable=MinY.g)
tkgrid(tklabel(FrameRight,text="Min Y"), MinY.entry, tklabel(FrameRight, text="Global Min  Y ="), MinY.g.label, sticky="e")
MaxY <- tclVar("")
MaxY.entry <-tkentry(FrameRight, width="12",textvariable= MaxY)
MaxY.g <- tclVar("unknown")
MaxY.g.label <- tklabel(FrameRight,text=tclvalue(MaxY.g))
```

```
tkconfigure(MaxY.g.label, textvariable=MaxY.g)
tkgrid(tklabel(FrameRight,text="Max Y"), MaxY.entry, tklabel(FrameRight, text="Global Max Y ="), MaxY.g.label, sticky="e")
Coord.but <- tkbutton(FrameRight,text="OK", command=function(){
        CircDataimageGlobals$MinX.d <<- as.numeric(tclvalue(MinX))
        CircDataimageGlobals$MaxX.d <<- as.numeric(tclvalue(MaxX))
        CircDataimageGlobals$MinY.d <<- as.numeric(tclvalue(MinY))
        CircDataimageGlobals$MaxY.d <<- as.numeric(tclvalue(MaxY))
        indexClosest <- which.min(abs(CircDataimageGlobals$x.g - CircDataimageGlobals$MinX.d)); CircDataimageGlobals$MinX.d <<- CircDataimageGlobals$x.g[indexClosest]
        indexClosest <- which.min(abs(CircDataimageGlobals$x.g - CircDataimageGlobals$MaxX.d)); CircDataimageGlobals$MaxX.d <<- CircDataimageGlobals$x.g[indexClosest]
        indexClosest <- which.min(abs(CircDataimageGlobals$y.g - CircDataimageGlobals$MinY.d)); CircDataimageGlobals$MinY.d <<- CircDataimageGlobals$y.g[indexClosest]
        indexClosest <- which.min(abs(CircDataimageGlobals$y.g - CircDataimageGlobals$MaxY.d)); CircDataimageGlobals$MaxY.d <<- CircDataimageGlobals$y.g[indexClosest]

        tclvalue(MinX) <- as.character(CircDataimageGlobals$MinX.d);   tclvalue(MaxX) <- as.character(CircDataimageGlobals$MaxX.d)
        tclvalue(MinY) <- as.character(CircDataimageGlobals$MinY.d);   tclvalue(MaxY) <- as.character(CircDataimageGlobals$MaxY.d)
        R1.Pan()
  }
)
tkgrid(Coord.but, column=1, sticky="e")
tkgrid(tklabel(FrameRight,text="          "))
tkgrid(tklabel(FrameRight,text="          "))
Smooth <- tclVar("0")
Smooth.function <- function()
{
        Bandwidth <- as.numeric(tclvalue(Smooth))
        if(Bandwidth > 0)
        {
                XVEC <- rep(CircDataimageGlobals$x.g, CircDataimageGlobals$ny.g)
                YVEC <- rep(CircDataimageGlobals$y.g, ea=CircDataimageGlobals$nx.g)
                ImageList.x <- as.image(as.vector(CircDataimageGlobals$u.g), x=data.frame(lon=XVEC, lat=YVEC),
                        nrow= CircDataimageGlobals$nx.g, ncol= CircDataimageGlobals$ny.g, boundary.grid=FALSE, na.rm=TRUE)
                u.g.Smooth <- image.smooth(ImageList.x, theta = Bandwidth)
                ImageList.y <- as.image(as.vector(CircDataimageGlobals$v.g), x=data.frame(lon=XVEC, lat=YVEC),
                        nrow= CircDataimageGlobals$nx.g, ncol= CircDataimageGlobals$ny.g, boundary.grid=FALSE, na.rm=TRUE)
                v.g.Smooth <- image.smooth(ImageList.y, theta = Bandwidth)
                CircDataimageGlobals$Direction <<- R1.Standardize(atan2(v.g.Smooth$z, u.g.Smooth$z))
        } else CircDataimageGlobals$Direction <<- CircDataimageGlobals$Direction.g
        R1.SubsetColorScale(CircDataimageGlobals$Direction[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
```

```
		CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol])
	R1.PlotImage()
}
Smooth.entry <-tkentry(FrameRight, width="12", textvariable=Smooth)
Smooth.but <- tkbutton(FrameRight,text="OK", command=Smooth.function)
tkgrid(tklabel(FrameRight,text="Smooth Bandwidth", font=FontHeading), Smooth.entry, sticky="e")
tkgrid(Smooth.but, column=1, sticky="e")
tkgrid(tklabel(FrameRight,text="          ")); tkgrid(tklabel(FrameRight,text="          "))
SliderValue2 <- tclVar("0")
SliderValueLabel2 <- tklabel(FrameRight,text=as.character(tclvalue(SliderValue2)))
tkconfigure(SliderValueLabel2,textvariable=SliderValue2)
slider2 <- tkscale(FrameRight, from=0, to=1, showvalue=TRUE, variable=SliderValue2, resolution=.05, orient="horizontal", length=".8i")
tkbind(slider2,"<ButtonRelease-1>", function() {CircDataimageGlobals$ColorGap <<- as.numeric(tclvalue(SliderValue2)); R1.ChangeColorGap()})
tkgrid(tklabel(FrameRight,text="Color Scale Gap", font=FontHeading), column=0, sticky="e")
tkgrid(slider2, column=1, sticky="e")
tkgrid(tklabel(FrameRight,text="0                    1"), column=1)
tkgrid(tklabel(FrameRight,text=" "), sticky="e")
tkgrid(tklabel(FrameRight,text=" "), sticky="e")
arrow.cb.value <- tclVar("0")
arrow.cb.function <- function()
{
	cbVal <- as.character(tclvalue(arrow.cb.value))
	if (cbVal=="1") {CircDataimageGlobals$PlotArrows <<- TRUE; R1.PlotImage()}
	if (cbVal=="0") {CircDataimageGlobals$PlotArrows <<- FALSE; R1.PlotImage()}
}
arrow.cb <- tkcheckbutton(FrameRight, command=arrow.cb.function)
tkconfigure(arrow.cb, variable=arrow.cb.value)
tkgrid(tklabel(FrameRight,text="Arrows", font=FontHeading), sticky="e")
tkgrid(arrow.cb, row=17, column=1, sticky="w")
arrow.length <- tclVar("1")
arrow.density <- tclVar("15")
arrow.function <- function()
{
	CircDataimageGlobals$ArrowAdj <<- as.numeric(tclvalue(arrow.length))
	CircDataimageGlobals$cpa <<- as.numeric(tclvalue(arrow.density))
	R1.PlotImage()
```

```
}
arrow.length.entry <-tkentry(FrameRight, width="6",textvariable=arrow.length)
tkgrid(tklabel(FrameRight,text="Arrow Length Multiplier"), arrow.length.entry,
        tklabel(FrameRight, text=">   0              "), sticky="e")
arrow.density.entry <-tkentry(FrameRight, width="6",textvariable=arrow.density)
tkgrid(tklabel(FrameRight,text="Arrow Spacing in Pixels"), arrow.density.entry,
        tklabel(FrameRight, text="1, 2, 3, ...       "), sticky="e")
Arrow.but <- tkbutton(FrameRight,text="OK", command=arrow.function)
tkgrid(Arrow.but, column=1, sticky="e")
tkgrid(tklabel(FrameRight,text="          "))
tkgrid(tklabel(FrameRight,text="          "))
mask.cb.value <- tclVar("0")
mask.cb.function <- function()
{
        cbVal <- as.character(tclvalue(mask.cb.value))
        if (cbVal=="1") {if(!is.null(CircDataimageGlobals$Mask)) {CircDataimageGlobals$PlotMask <<- TRUE; R1.PlotImage()}}
        if (cbVal=="0") {CircDataimageGlobals$PlotMask <<- FALSE; R1.PlotImage()}
}
mask.cb <- tkcheckbutton(FrameRight, command=mask.cb.function)
tkconfigure(mask.cb, variable=mask.cb.value)

tkgrid(tklabel(FrameRight,text="Mask", font=FontHeading), sticky="e")
tkgrid(mask.cb, row=23, column=1, sticky="w")
tkgrid(tklabel(FrameRight,text="          "))
tkgrid(tklabel(FrameRight,text="          "))
tkpack(FrameTop, side="top", fill="x")
tkpack(FrameLeft, side="left", fill="both", expand=TRUE)
tkpack(FrameRight, side="right", fill="both", expand=TRUE)
}
```

K.3 SimulateSill

```
SimulateSill <- function()
{
	require(CircStats)

	CircDist <- function(alpha,beta)
	{
		alpha[alpha < 0] <- 2*pi + alpha[alpha < 0]
		beta[beta < 0] <- 2*pi + beta[beta < 0]
		theta <- abs(alpha - beta)
		theta[theta > pi] <- 2*pi - theta[theta > pi]
		return(theta)
	}

	VM <- c(); U <- c(); C <- c(); WC <- c(); T <- c()
	Cavg  <- vector(mode="numeric", length=1000)
	Tavg  <- vector(mode="numeric", length=1000)
	Uavg  <- vector(mode="numeric", length=1000)
	VMavg <- vector(mode="numeric", length=1000)
	WCavg <- vector(mode="numeric", length=1000)

	filter <- upper.tri(matrix(data=NA, nrow=100, ncol=100), diag = F)
```

```
	for (i in 1:1000)
	{
		Sample <- rcard(n=100,mu=0,r=.25); cosines <- cos(outer(Sample, Sample, FUN="CircDist"))
		C <- c(C, cosines[filter])
		Cavg[i] <- mean(C)

		Sample <- rtri(n=100, r=.5*4/pi^2); cosines <- cos(outer(Sample, Sample, FUN="CircDist"))
		T <- c(T, cosines[filter])
		Tavg[i] <- mean(T)

		Sample <- 2*pi*runif(100); cosines <- cos(outer(Sample, Sample, FUN="CircDist"))
		U <- c(U, cosines[filter])
		Uavg[i] <- mean(U)

		Sample <- rvm(n=100, mean=0, k=5); cosines <- cos(outer(Sample, Sample, FUN="CircDist"))
		VM <- c(VM, cosines[filter])
		VMavg[i] <- mean(VM)

		Sample <- rwrpcauchy(n=100,location=0,rho=exp(-1)); cosines <- cos(outer(Sample, Sample, FUN="CircDist"))
		WC <- c(WC, cosines[filter])
		WCavg[i] <- mean(WC)
	}
	return(list(Cavg=Cavg, Tavg=Tavg, Uavg=Uavg, VMavg=VMavg, WCavg=WCavg))
}
```

K.4 CorrelationTransfer

```
CorrelationTransfer <- function(nPoints=50, CircDistr2="vM", Rho2=.75, Range2=10, Ext2=2, CovModel2="spherical", GRID=NULL,
OVERFIT=TRUE)
{
        # 2008-8-10.1356
        # Circular parameters: Triangular, 0 < Rho <= 4/pi^2; cardioid, 0 < Rho <= 0.5; vM and wrapped Cauchy, 0 < Rho < 1; uniform, Rho = 0
        # nPoints= number of points per simulation
        # OverFit=TRUE, or standardization (centering and rescaling realization of the GRV to mean 0 sd 1) results in closer fit
        #   for qualitative evaluation of the CRV.  Undesirable effects are loss of independence of the marginal GRVs, biased GRF
        #   covariance, and biased testing.  Standardization is suitable for demonstration with closer fit, visualization, and
        #   illustrations.  Do not standardize for purposes of simulation and testing.  OverFit=FALSE, or non-standardization (default)
        #   includes expected variation from transformation of variation in mean and sd of sample of GRV.

        if(is.null(GRID)) {output <- SimulateCRF(N=nPoints, CircDistr=CircDistr2, Rho=Rho2, Range=Range2, Ext=Ext2,
                CovModel=CovModel2, OverFit=OVERFIT)} else {output <- SimulateCRF(CircDistr=CircDistr2, Rho=Rho2, Range=Range2,
                CovModel= CovModel2, OverFit=OVERFIT, Grid=GRID)}
        par(mfrow=c(3,1), mgp=c(1.5,.5,0), mai=c(.4,.4,.3,.1))

        # GRF variogram
        vario.z <- variog(coords = cbind(output$x, output$y), data = output$Z, option = "bin", uvec=seq(2,54,by=2))
        plot(vario.z$u, vario.z$v, main="Variogram of GRF", cex.main=1.2, xlab="Distance", ylab="Semi Variance", ylim=c(0, 2))
        abline(v=Range2, col="grey")

        # Cumulative Probability variogram
        vario.p <- variog(coords = cbind(output$x, output$y), data =pnorm(output$Z, mean=0, sd=1, lower.tail = TRUE),
                option = "bin", uvec=seq(2,54,by=2))
        plot(vario.p$u, vario.p$v, main="Variogram of Cumulative Probabilities of GRV", cex.main=1.2, xlab="Distance", ylab="Semi Variance",
                ylim=c(0,0.2))
        abline(v=Range2, col="grey")

        # Cosineogram
        CosinePlots(x=output$x, y=output$y, directions=output$direction, Lag.n.Adj= 1, Lag=vario.p$u,
                main="Cosineogram of CRF", cex.main=1.2, ylim=c(0,1))
        abline(v=Range2, col="grey")
}
```

K.5 SimulateCRF

```
SimulateCRF <-function(N=100, CircDistr, Rho, Mu=0, Range, Ext=1, CovModel, Grid=NULL, Anisotropy=NULL, OverFit=FALSE,
Resolution=0.01)
{
	# 2008-11-10.2001
	# Simulate CRF ~ (Range, CircDistr, Rho, mu=0)

	# Input Arguments
	# N: Number of spatial locations to simulate
	# CircDistr: Circular distribution in {U, vM, WrC, Tri, Card},
	# Rho: Mean resultant length parameter
	#	For triangular, 0 < Rho <= 4/pi^2
	#	For cardioid, 0 < Rho <= 0.5
	#	For vM and wrapped Cauchy, 0 < Rho < 1, 1== degenerate
	#	For uniform, Rho = 0
	# Range: Distance at which CRV independent
	# Ext: Range*Ext is horizontal and vertical length of sample space
	# CovModel: Name of spatial correlation function, see package geoR Help cov.spatial
	# Grid: Regular or irregular N x 2 matrix of simulation locations, overides N and Ext
	# Anisotropy: Vector of geometric anisotropy angle in radians, ratio > 1.
	# OverFit=TRUE, or standardization (centering and rescaling realization of the GRV to mean 0 sd 1) results in closer fit
	#   for qualitative evaluation of the CRV.  Undesirable effects are loss of independence of the marginal GRVs, biased GRF
	#   covariance, and biased testing.  Standardization is suitable for demonstration with closer fit, visualization, and
	#   illustrations.  Do not standardize for the purposes of simulation and testing.  OverFit=FALSE, or non-standardization (default)
	#   includes expected variation from transformation of variation in mean and sd of sample of GRV.

	# Values
	# x,y: Vectors of location coordinates
	# direction: Vector of directions
	# Z: Vector of simulated observations of the associated GRV

	# Note:
	# At n > 500, geoR transfers processing the the package Random Fields because the option RF is set.

	if(CircDistr !="U" & CircDistr !="vM" & CircDistr !="WrC" & CircDistr !="Tri" & CircDistr !="Card")
```

```
		stop("CRF not implemented for input CircDistr")

	if(CircDistr =="U") Rho = 0
	if(abs(Mu) > pi) stop("abs(Mu) <= pi")

	if(!is.null(Grid)) {
		if(class(Grid) != "matrix") stop("Grid not a matrix")
		if(dim(Grid)[2] !=2) stop("Grid not a N x 2 matrix")
		N <- dim(Grid)[1]}
	if(!is.null(Anisotropy)) {
		if(length(Anisotropy) != 2) stop("Anisotropy is not a 2 element vector.  See geoR Help")
	}

	if(N <=0 | Rho < 0 | Range < 0| Ext <=0 | Resolution <= 0) stop("Improper numeric input")

	direction <- vector(mode="numeric", length=N)

	require(CircStats)
	require(geoR)
	# Standard normal GRF, see Help geoR grf
	if(is.null(Grid)) {
		GRF <- grf(n=N, xlims=c(0, Range*Ext), ylims=c(0, Range*Ext), cov.model=CovModel,
			nugget=0, cov.pars=c(1, Range), aniso.pars=Anisotropy, RF=TRUE, messages=FALSE) } else {
		GRF <- grf(grid=Grid, cov.model=CovModel,
			nugget=0, cov.pars=c(1, Range), aniso.pars=Anisotropy, RF=TRUE, messages=FALSE)}

	XY <- GRF$coords # N x 2 matrix
	x <- XY[,1]; y <- XY[,2]
	Z <- GRF$data # Vector of GRV
	if(OverFit) {Z <- (Z - mean(Z))/sd(Z); GRF$data <- Z}
	CumProbZ <- pnorm(Z, mean=0, sd=1, lower.tail = TRUE)

	if(CircDistr=="U") {direction <- -pi + 2*pi*CumProbZ} else
	if(CircDistr == "Tri")
	{
		if(Rho==0 | Rho > 4/pi^2) stop("Tri: 0 < Rho <= 4/pi^2")
```

```
		filter <- CumProbZ < 0.5
		u1 <- CumProbZ[filter]
		a <- Rho/8
		b <- (4+pi^2*Rho)/(8*pi)
		c <- 0.5 - u1
		q <- -.5*(b+sqrt(b^2-4*a*c))
		direction[filter] <- c/q

		u2 <- CumProbZ[!filter]
		a <- -Rho/8
		b <- (4+pi^2*Rho)/(8*pi)
		c <- 0.5 - u2
		q <- -.5*(b+sqrt(b^2-4*a*c))
		direction[!filter]<- c/q
	} else
	{
		# For OTHER circular distributions compute table of circular CDF and interpolate
		CircScale <- seq(-pi, pi, length=2*pi/Resolution)
		# With resolution=.01, circular support from -pi to +pi has 629 elements, delta ~0.01000507, CircScale[315] = 0
		n <- length(CircScale)
		if(CircDistr == "vM")
		{
			if(Rho==0 | Rho >= 1) stop("vM: 0 < Rho < 1")
			CircProb <- rep(-1, n)
			Kappa=A1inv(Rho) # N. I Fisher, Statistical Analysis of Circular Data, 2000 p. 49
			# As direction increases from -pi, pvm increases from .5
			for(i in 1:length(CircScale)) CircProb[i] <- pvm(CircScale[i], mu=0, kappa=Kappa)
			filter <- CircScale < 0
			CircProb[filter] <- CircProb[filter] - 0.5
			CircProb[!filter] <- CircProb[!filter] + 0.5
		} else
		if(CircDistr == "Card")
		{
			if(Rho==0 | Rho > 0.5) stop("Cardioid: 0 < Rho <= 0.5")
			CircProb <- (CircScale + pi + 2*Rho*sin(CircScale))/(2*pi)
		} else
```

```
        if(CircDistr == "WrC")
        {
                if(Rho==0 | Rho >= 1) stop("Wrapped Cauchy: 0 < Rho < 1 ")
                Angles1 <- CircScale[CircScale < 0]
                Angles2 <- CircScale[CircScale >= 0]
                prob1 <- 0.5 - acos(((1+Rho^2)*cos(Angles1) - 2*Rho)/(1 + Rho^2 - 2*Rho * cos(Angles1)))/(2*pi)
                prob2 <- 0.5 + acos(((1+Rho^2)*cos(Angles2) - 2*Rho)/(1 + Rho^2 - 2*Rho * cos(Angles2)))/(2*pi)
                CircProb <-c(prob1, prob2)
        }
        CircProb[1] <- 0; CircProb[n] <- 1

        # Interpolation
        DeltaTh <- CircScale[2] + pi
        for(i in 1:N)
        {
                p <- CumProbZ[i]  # Cumulative prob of GRV
                a <- max((1:n)[CircProb <= p]) # Index
                if(a==n) {r <- 0} else
                {
                        if(CircProb[a]==p) {r <- 0} else {r <- (p -CircProb[a])/( CircProb[a+1] -CircProb[a])}
                }
                direction[i] <- CircScale[a] + r*DeltaTh
        }
    }

    direction <- direction + Mu
    return(list(x=x, y=y, direction=direction, Z=Z))
}
```

K.6 AssessCRF

```
AssessCRF <- function(nPoints=100, CircDistr2="vM", Rho2=.75, Range2=10, Ext2=3, CovModel2="spherical", GRID=NULL, OVERFIT=TRUE)
{
        # 2008-2-15.0600
        # Generate a nPoints x nPoints and compare QQ Circ plots to QQ norm plots
        # range of parameters
        #   For triangular, 0 < Rho <= 4/pi^2
        #   For cardioid, 0 < Rho <= 0.5
        #   For vM and wrapped Cauchy, 0 < Rho < 1, 1== degenerate
        #   For uniform, Rho = 0
        # nPoints= number of points
        # OverFit=TRUE, or standardization (centering and rescaling realization of the GRV to mean 0 sd 1) results in closer fit
        #   for qualitative evaluation of the CRV.  Undesirable effects are loss of independence of the marginal GRVs, biased GRF
        #   covariance, and biased testing.  Standardization is suitable for demonstration with closer fit, visualization, and
        #   illustrations.  Do not standardize for purposes of simulation and testing.  OverFit=FALSE, or non-standardization (default)
        #   includes expected variation from transformation of variation in mean and sd of sample of GRV.

        require(CircStats)
        if(is.null(GRID)) {output <- SimulateCRF(N=nPoints, CircDistr=CircDistr2, Rho=Rho2, Range=Range2, Ext=Ext2,
                CovModel=CovModel2, OverFit=OVERFIT)} else {output <- SimulateCRF(CircDistr=CircDistr2, Rho=Rho2, Range=Range2,
                CovModel= CovModel2, OverFit=OVERFIT, Grid=GRID); nPoints <- nrow(GRID) }

        Z <- output$Z
        Zsort <- sort(Z)
        a <- ifelse(nPoints <= 10, 3/8, 1/2)
        CumProb <- ((1:nPoints)- a)/(nPoints + 1 - 2*a) # Vector of symmetric cumulative probabilities for QQ plots
        ZQuantiles <- qnorm(CumProb, mean=0, sd=1, lower.tail = TRUE)

        Theta <- output$direction
        Thetasort <- sort(Theta)
        # Compute theta quantiles
        if(CircDistr2=="U") { ThetaQuantiles <- -pi + 2*pi*CumProb } else
        {
                # For non-uniform circular distributions use circular CDF to get ThetaQuantiles
                CircScale <- seq(-pi, pi, length=2*pi/.01) # Circular support from -pi to +pi, 629 elements, d~.01, CircScale[315] is zero
```

```
n <- length(CircScale)
if(CircDistr2=="vM")
{
	if(Rho2==0 | Rho2 >= 1) stop("vM: 0 < Rho < 1")
	CircProb <- rep(-1, n)
	Kappa=A1inv(Rho2) # N. I Fisher, Statistical Analysis of Circular Data, 2000 p. 49
	# As theta increases from -pi, pvm increases from .5
	for(i in 1:n) {CircProb[i] <- pvm(CircScale[i], mu=0, kappa=Kappa)}
	filter <- CircScale < 0
	CircProb[filter] <- CircProb[filter] - 0.5
	CircProb[!filter] <- CircProb[!filter] + 0.5
} else
if(CircDistr2=="Tri")
{
	if(Rho2==0 | Rho2 > 4/pi^2) stop("Tri: 0 < Rho <= 4/pi^2")
	Angles1 <- CircScale[CircScale < 0] + 2*pi
	Angles2 <- CircScale[CircScale >= 0]
	CircProb <- c( (4 - 3*pi^2*Rho2 + pi*Rho2*(Angles1 + pi))*(Angles1-pi)/(8*pi),
	.5 + (4 +   pi^2* Rho2 - pi*Rho2 * Angles2) * Angles2/(8*pi) )
} else
if(CircDistr2=="Card")
{
	if(Rho2==0 | Rho2 > 0.5) stop("Cardioid: 0 < Rho <= 0.5")
	Angles1 <- CircScale[CircScale < 0] + 2*pi
	Angles2 <- CircScale[CircScale >= 0]
	CircProb <- c( (Angles1 - pi + 2*Rho2*sin(Angles1))/(2*pi), 0.5 + (Angles2 + 2*Rho2*sin(Angles2))/(2*pi) )
} else
if(CircDistr2=="WrC")
{
	if(Rho2==0 | Rho2 >= 1) stop("Wrapped Cauchy: 0 < Rho < 1")
	Angles1 <- CircScale[CircScale < 0] + 2*pi
	Angles2 <- CircScale[CircScale >= 0]
	CircProb <-c( 0.5 - acos(((1+Rho2^2)*cos(Angles1) - 2*Rho2)/(1 + Rho2^2 - 2*Rho2 *
	cos(Angles1)))/(2*pi), 0.5 + acos(((1+Rho2^2)*cos(Angles2) - 2*Rho2)/(1 + Rho2^2 - 2*Rho2 * cos(Angles2)))/(2*pi) )
}
CircProb[1] <- 0; CircProb[n] <- 1 # For any numerical imprecision
```

```
	# Quantiles From Inverse Circular CDF
	ThetaQuantiles <- vector(mode="numeric", length=nPoints)
	DeltaTh <- CircScale[2] + pi
	for(i in 1:nPoints)
	{
		p <- CumProb[i]
		a <- max((1:n)[CircProb <= p]) # Left index
		if(a==n) { r <- 0} else { if(CircProb[a]==p) {r <- 0} else {r <- (p - CircProb[a])/( CircProb[a+1]- CircProb[a])} }
		ThetaQuantiles[i] <- CircScale[a] + r*DeltaTh
	}
}
par(mfrow=c(3,2), mgp=c(2,1,0), mar=c(4.1,3.1,3.1,1.1))

# QQ std norm
plot(ZQuantiles, Zsort, main=paste("QQ Standard Normal of", "\nGRV With Spatial Correlation", sep=""), cex.main=1,
	xlab="Theoretical Quantiles", ylab="Ordered GRV", col=1, xlim=c(-pi,pi),ylim=c(-pi,pi), ty='l')
abline(0,1,col=4); abline(v=0, col="grey"); abline(h=0, col="grey")

# GRF variogram
vario.b <- variog(coords = cbind(output$x, output$y), data = Z, option = "bin")
plot(vario.b$u, vario.b$v, main=paste("Variogram of GRF", "\n Model=",  CovModel2, ", Range=", Range2, ", Sill=1", ", mean=0", sep=""),
	cex.main=1, xlab="Distance", ylab="Semi Variance")
abline(v=Range2, col="grey")
abline(h=1, col="grey")

# QQ Circ probability law
if(CircDistr2=='U') {Distrib = "Uniform"} else
if(CircDistr2=='vM') {Distrib = "von Mises"} else
if(CircDistr2=='WrC') {Distrib = "Wrapped Cauchy"} else
if(CircDistr2=='Tri') {Distrib = "Triangular"} else
if(CircDistr2=='Card') {Distrib = "Cardioid"}
```

```
        if(CircDistr2=='U') {title.rho=""} else {title.rho=paste(", Rho =", round(Rho2, digits=3), sep="")}
        plot(ThetaQuantiles, Thetasort, main=paste( "QQ ", Distrib, title.rho, "\nCRV With Spatial Correlation",sep="" ), cex.main=1,
                xlab="Theoretical Quantiles (Rad)", ylab="Ordered CRV", col=1, xlim=c(-pi,pi),ylim=c(-pi,pi), ty='l')
        lines(c(-pi,pi), c(-pi,pi), col=4); abline(v=0, col="grey"); abline(h=0, col="grey")

        # Cosineogram
        CosinePlots(x=output$x, y=output$y, directions=output$direction, Lag.n.Adj= 1, Lag=vario.b$u, main="Cosineogram of CRF")
        if(CircDistr2!="U") abline(h=Rho2^2, col="grey") else abline(h=0, col="grey")
        abline(h= est.rho(Theta)^2, col=4, lty=3) # Sample mean resultant length
        abline(v=Range2, col="grey")

        # Uniformity plot
        probabilities <- pnorm(Z, mean=0, sd=1, lower.tail = TRUE)
        uniformity <- mean(abs(CumProb - sort(probabilities))) # Mean absolute deviation
        plot(CumProb, sort(probabilities) , main=paste( "QQ Uniform of Cumulative Probabilities",
                "\nMean - 1/2=", round(mean(probabilities)-.5, digits=3), ", Var - 1/12=", round((sd(probabilities))^2-1/12, digits=3),
                ", Closeness=", round(uniformity, digits=3), sep="" ), cex.main=1,
                xlab="Theoretical Quantiles", ylab="Ordered Probabilities", col=1, xlim=c(0,1),ylim=c(0,1), ty='l')
        lines(c(0,1), c(0,1),col=4); abline(v=.5, col="grey"); abline(h=.5, col="grey")
}
```

K.7 PlotVectors

```
PlotVectors <- function(x, y, h, v, UnitVector=TRUE, TriIcon=FALSE, AdjArrowLength=1, AdjHeadLength=1, TriIconAdj=1,
	TriRatio=4, JitterPlot=FALSE, Jitter=1, ...)
{
	# 2008-11-11.1535
	# Arrows do not plot where data is missing.
	require(fields)

	if( (length(x) != length(y)) | (length(h) != length(v)) | (length(x) != length(h)) ) stop("lengths of vector inputs unequal")

	filter <- is.na(h) | is.na(v) | (h==0 & v==0)
	x <- x[!filter]; y <- y[!filter]; h <- h[!filter]; v <- v[!filter]
	# fields function arrows omits arrowheads with a warning on any arrow of length less than 1/1000 inch.

	Dir <- atan2(v, h)
	Dir[Dir<0] <- Dir[Dir<0]+2*pi
	if(JitterPlot==TRUE)
	{
		x <- x + Jitter*runif(length(x))
		y <- y + Jitter*runif(length(y))
	}
	plot(x, y, ty="n", asp=1, ...)

	if(UnitVector)
	{ arrow.plot(x, y, cos(Dir), sin(Dir), true.angle=TRUE, arrow.ex=AdjArrowLength*0.05, length=AdjHeadLength*0.125,
		angle=20, xpd=FALSE)
	} else
	{
		if(TriIcon)
		{
			m <- sqrt(h^2 + v^2) # magnitude
			w = sqrt(m/TriRatio)
			n <- length(x)
			xa <- x + TriIconAdj*	w*cos(Dir+pi/2)
			ya <- y + TriIconAdj*	w*sin(Dir+pi/2)
```

```
				xb <- x + TriIconAdj*TriRatio*w*cos(Dir)
				yb <- y + TriIconAdj*TriRatio*w*sin(Dir)
				xc <- x + TriIconAdj*      w*cos(Dir-pi/2)
				yc <- y + TriIconAdj*      w*sin(Dir-pi/2)

				for(i in 1:n) polygon(x=c(xa[i], xb[i], xc[i]), y=c(ya[i], yb[i], yc[i]), density=-1, col=1)
			} else arrow.plot(x, y, h, v, true.angle=TRUE, arrow.ex=AdjArrowLength*0.05, length=AdjHeadLength*0.125,
				angle=20, xpd=FALSE)
		}
	}
```

K.8 CircResidual

```
CircResidual <- function(X, Y, Raw, Trend, Plot = FALSE, AdjArrowLength = 1, ...)
{
	# 2008-11-10.2053
	# Assumptions: Raw may have NAs, trend has no NAs.  Trend locations and Raw locations are identical to compute residuals.
	require(fields)
	if((length(X) != length(Y)) | (length(X) != length(Raw)) | (length(X) != length(Trend)) | (length(Y) != length(Raw)) |
		(length(Y) != length(Trend)) | (length(Raw) != length(Trend))) stop("lengths of vector inputs unequal")
	if(AdjArrowLength <= 0) stop("AdjArrowLength invalid")
	if(sum(is.na(Trend)) > 0) stop("NAs not allowed in Trend")

	FilterNA <- is.na(Raw)
	x <- X[!FilterNA]; y <- Y[!FilterNA]; raw <- Raw[!FilterNA]; trend <- Trend[!FilterNA]
	raw[raw < 0] <- raw[raw < 0] + 2*pi # Like R1.Standardize in CircDataimage
	trend[trend < 0] <- trend[trend < 0] + 2*pi
	circdist <- abs(raw - trend) # Linear distance in radians with NAs where raw has NAs
	circdist[circdist > pi] <- 2*pi - circdist[circdist > pi] # Circular distance in radians
	resids <- circdist
	filter <- (trend>raw) & (trend-raw)<pi | (raw >trend) & (raw-trend)>pi
	resids[filter] <- -1* circdist[filter]
	if(Plot==TRUE)
	{
		plot(X, Y, type="n", xlab="", ylab="", asp=1, ...)
		arrow.plot(x, y, u=cos(raw), v=sin(raw), xpd=FALSE, true.angle=TRUE, arrow.ex=.15*AdjArrowLength, length=.1,col=1)
		arrow.plot(X, Y, u=cos(Trend), v=sin(Trend), xpd=FALSE, true.angle=TRUE, arrow.ex=.15*AdjArrowLength, length=.1,
			col="tan", lwd=3)
		arrow.plot(x, y, u=cos(resids), v=sin(resids), xpd=FALSE, true.angle=TRUE, arrow.ex=.15*AdjArrowLength, length=.1,
			col=2, lty=2)
	} else return(list(x=x, y=y, direction=resids))
}
```

K.9 CosinePlots

```
CosinePlots <- function(x, y, directions, Lag=NULL, Lag.n.Adj = 1, BinWAdj=1, Plot = TRUE,
	Cloud = FALSE, Model=FALSE, nugget=0, Range=NULL, sill=NULL, x.legend=0.6, y.legend=1.0, TrimMean=0.1, ...)
{
	# 2008-11-11.1050
	# Assumption: Isotropic circular random field
	# x, y are vectors of location coordinates, directions is a vector of directions in radians.
	# Lag is a vector of lag points.  Lag.n.Adj > 0 multiplies the number of lag points.
	# BinWAdj >= 1 multiplies bin width (to make bins narrower increase Lag.n.adj).  Sturges rule determines nBins.
	# nBins and Lag.n.Adj determine Lag.n.  Lag.n adjusts nBins.  nBins and BinWAdj determine bin width.
	# Plot = TRUE plot cosineocloud or cosineogram, else ouput list of points.  Cloud = TRUE plots cosineocloud, else cosineogram.
	# Model = TRUE overplots exponential, gaussian, and spherical models with nugget, Range, and sill parameters.
	# x.legend and y.legend adjust legend location.
	# TrimMean = 0.1 applies trimmed mean in computing mean cosine.

	if( (length(x) != length(y)) | (length(x) != length(directions)) | (length(y) != length(directions)) )
		stop("lengths of vector inputs unequal")
	if(Lag.n.Adj <= 0) stop("Lag.n.Adj invalid")
	if( (nugget < 0) | (nugget > 1) ) stop("nugget invalid")
	if(!is.null(Range)) {if(Range <= 0) stop("Range negative")}
	if(!is.null(sill)) {
		if( (sill < 0) | (sill >=1) ) stop("sill invalid")
		if(1-nugget < sill) stop("1-nugget < sill")}

	# Repair Input and Remove missings
	if(BinWAdj < 1) BinWAdj <- 1 # points will fall out of bins if adjust < 1
	filter <- !is.na(directions)
	x <- x[filter]; y <- y[filter]; directions <- directions[filter]

	# Pairwise cosines
	# Subroutine to compute circular distances in radians
	CircDist <- function(alpha,beta)
	{
		alpha[alpha < 0] <- 2*pi + alpha[alpha < 0]
		beta[beta < 0] <- 2*pi + beta[beta < 0]
```

```
		theta <- abs(alpha - beta)
		theta[theta > pi] <- 2*pi - theta[theta > pi]
		return(theta)
}
Cosines <- cos(outer(directions, directions, FUN="CircDist"))
filter.tri <- upper.tri(Cosines)
Cosines <- Cosines[filter.tri]

# Pairwise distances
Distances <- as.matrix(dist(cbind(x, y))) # Diagonal of zero distances
X <- Distances[filter.tri] # vector of distances corresponding to vector of cosines Cosines

if(Cloud) {Y <- Cosines} else
{
		if(!is.null(Lag))
		{
				# Assumes equally spaced lags, except for first lag point
				HalfBinWidth <- BinWAdj * 0.5 * (Lag[2] - Lag[1])
				Lag.n <- length(Lag)
		} else
		{
				nBins <- trunc(log2(sum(filter.tri)) + 1) # Sturges rule
				Lag.n <- trunc(Lag.n.Adj*(nBins + 1))
				nBins <- Lag.n - 1
				distance.max <- max(X)
				HalfBinWidth <- BinWAdj * 0.5 * distance.max/nBins
				Lag <- seq(0, distance.max, length.out= Lag.n)
		}

		Y <- vector(mode = "numeric", length = Lag.n)
		if(Lag[1] == 0) {Y[1] <- 1; i1 <- 2} else i1 <- 1
		for (i in i1:Lag.n)
		{
				filter <- abs(X - Lag[i]) <= HalfBinWidth
				Y[i] <- mean(Cosines[filter], trim=TrimMean)
		}
```

```
		X <- Lag # Cloud=FALSE filtered distances replaced by lag vector
	}
	if(Plot)
	{
		plot(X, Y, xlab = "Distance", ylab = "Cosine", cex.main=1, ...)
		if(Cloud == FALSE & Model == TRUE)
		{
			xx <- seq(min(X), max(X), length.out=101)
			c.e <- 1-nugget -(1-nugget-sill)*(1-exp(-3*xx/Range))
			c.g <- 1-nugget -(1-nugget-sill)*(1-exp(-3*(xx/Range)^2))
			X1 <- xx[xx <= Range]
			c.s <- 1-nugget -(1-nugget-sill)*(1.5*X1/Range-0.5*(X1/Range)^3)
			X2 <- xx[xx > Range]
			c.s <- c(c.s, rep(sill, length(X2)))
			lines(xx, c.e, col=2,     lty=1, lwd=1)
			lines(xx, c.g, col="tan", lty=1, lwd=3)
			lines(xx, c.s, col=4,     lty=2, lwd=1)

			legend(x=x.legend*max(X), y=y.legend, c("Exponential","Gaussian","Spherical"),
			lty = c(1, 1, 2), col=c(2, "tan", 4), lwd=c(1, 3, 1), cex=1.1)
		}
	} else {return(list(distance = X, cosine = Y))}
}
```

K.10 KrigCRF

```
KrigCRF <- function(krig.x, krig.y, resid.x, resid.y, resid.direction, Model, Nugget=0, Range, sill,
        Smooth=FALSE, bandwidth, Plot=FALSE, Xlim=NULL, Ylim=NULL, PlotVar=FALSE, ...)
{
        # 2008-11-11.1213
        # select model from covariance models in R package Random Fields functin CovarianceFct
        # resid.x and resid.y have no NAs

        if( (length(krig.x) != length(krig.y)) | (length(resid.x) != length(resid.y)) | (length(resid.x) != length(resid.direction)) |
                (length(resid.y) != length(resid.direction)) ) stop("lengths of vector inputs unequal")
        if( (Nugget < 0) | (Nugget > 1) ) stop("Nugget invalid")

        # fix the order of the kriging coordinates
        xx <- sort(unique(krig.x)); yy <- sort(unique(krig.y))
        nx <- length(xx); ny <- length(yy) # rectangular or square grid
        krig.y <- rep(yy, nx); krig.x <- rep(xx, each=ny)

        require(fields)
        require(RandomFields)

        Distances <- as.matrix(dist(cbind(resid.x, resid.y)))
        Ncol <- ncol(Distances)
        K <- c()
        for (i in 1:Ncol)
        {
                K <- cbind(K, sill + (1 - Nugget - sill)*CovarianceFct(x=Distances[, i]/Range, model=Model,
                        param=c(mean=0,variance=1,nugget=0,scale=1, ...), dim=1, fctcall="Covariance"))
        }
        diag(K) <- 1 # TRUE even if nugget > 0 for any model
        Kinv <- solve(K)

        U <- t(cbind(cos(resid.direction), sin(resid.direction)))
        # V <- t(U) %*% U

        n <- length(krig.x) # krig.x=krig.y for square or rect grid
```

```
krig.direction <- vector(mode="numeric", length=n)
krig.variance <- krig.direction

for(i in 1:n)
{
        distances <- sqrt((krig.x[i]-resid.x)^2 + (krig.y[i]-resid.y)^2)
        c <- sill + (1 - Nugget - sill)*CovarianceFct(x=distances/Range, model=Model,
                param=c(mean=0,variance=1,nugget=0,scale=1, ...), dim=1, fctcall="Covariance")
        c[distances == 0] <- 1 # TRUE even if nugget > 0 for any model
        w <- Kinv %*% c

        # w <- (Kinv %*% c)/sqrt(as.numeric(t(c) %*% Kinv %*% V %*% Kinv %*% c)) # gives same directions
        u <- U %*% w
        krig.direction[i] <- atan2(u[2],u[1])
        krig.variance[i] <- 2 - 2*sqrt(as.numeric(t(c) %*% Kinv %*% c))
}

if(Smooth)
{
        xx.dx <- xx[2] - xx[1]; yy.dy <- yy[2] - yy[1]
        # as.image loads the matrix by row
        ImageList.x <- as.image(cos(krig.direction), x=data.frame(krig.x, krig.y), nrow=nx, ncol=ny, boundary.grid=FALSE)
        smooth.x <- image.smooth( ImageList.x, theta = bandwidth)
        ImageList.y <- as.image(sin(krig.direction), x=data.frame(krig.x, krig.y), nrow=nx, ncol=ny, boundary.grid=FALSE)
        smooth.y <- image.smooth( ImageList.y, theta = bandwidth)
        krig.direction <- as.vector(t(atan2(smooth.y$z, smooth.x$z)))
}

if(Plot)
{
        if(!PlotVar)
        {
                plot(krig.x, krig.y, ty="n", xlab="", ylab="", asp=1, xlim=Xlim, ylim=Ylim)
                arrow.plot(a1=krig.x, a2=krig.y, u=cos(krig.direction), v=sin(krig.direction), arrow.ex=0.06,
                        xpd=FALSE, true.angle=TRUE, length=.05, col=1)
        } else
```

```
		{
			krig.variance.matrix <- matrix(data=krig.variance, nrow=nx, ncol=ny, byrow=TRUE)
			if(!is.null(Xlim))
			{
				filled.contour(x = xx, y = yy, z = krig.variance.matrix, xlim=Xlim, ylim=Ylim,
				color = terrain.colors, key.title = title(main="Circ Krig \nVariance", cex.main=0.8),
				asp = 1, plot.axes={axis(1); axis(2); points(resid.x, resid.y, pch=20, cex=.65)})
			} else {
				filled.contour(x = xx, y = yy, z = krig.variance.matrix,
				color = terrain.colors, key.title = title(main="Circ Krig \nVariance", cex.main=0.8),
				asp = 1, plot.axes={axis(1); axis(2); points(resid.x, resid.y, pch=20, cex=.65)})
			}
		}
	} else return(list(x=krig.x, y=krig.y, direction=krig.direction, variance=krig.variance))
}
```

K.11 InterpDirection

```
InterpDirection <- function(in.x, in.y, in.direction, out.x, out.y)
{
	# 2008-11-11.1444
	# Interpolate models of direction cosines and sines, separately to avoid cross over.  Fit plane to triangular half of cell
	# (rectangular element of regular grid of measurement locations) in which interpolation location occurs.
	# Assumptions - Locations to interpolate are within range of (in.x, in.y), inputs have no missing.

	# Arguments
	# in.x vector of input horizontal coordinates
	# in.y vector of input vertical   coordinates
	# in.direction vector of input direction in radians

	# Value
	# out.x vector of interpolation output horizontal coordinates
	# out.y vector of interpolation output vertical   coordinates
	# out.direction vector of interpolation output direction

	# Verify input
	minx.in <- min(in.x); maxx.in <- max(in.x); miny.in <- min(in.y); maxy.in <- max(in.y)
	minx.out <- min(out.x); maxx.out <- max(out.x); miny.out <- min(out.y); maxy.out <- max(out.y)
	if(minx.out < minx.in | maxx.out > maxx.in | miny.out < miny.in | maxy.out > maxy.in)
		stop("Interpolation range exceeds range of (in.x, in.y)")

	if( (length(in.x) != length(in.y)) | (length(in.x) != length(in.direction)) | (length(in.y) != length(in.direction)))
		stop("lengths of vector inputs unequal")

	if( length(out.x) != length(out.y) ) stop("lengths of vector outputs unequal")

	# Organize model data
	X <- sort(unique(in.x), decreasing = FALSE) # Increases left to right
	m <- length(X)
	Y <- sort(unique(in.y), decreasing = TRUE) # Decreases top to bottom
	n <- length(Y)
	# Col of matrix of directions reflects the horiz or west to east component location
```

```
xmin <- min(X); deltax <- X[2] - X[1]; ymax <- max(Y); deltay <- Y[1] - Y[2]
Col <- 1 + (in.x -xmin)/deltax
Row <- 1 + (ymax - in.y)/deltay
directions <- matrix(data = NA, nrow=n, ncol=m)
directions[cbind(Row, Col)] <- in.direction # matrix of organized directions
U <- cos(directions) # matrix of organized cosines of directions
V <- sin(directions) # matrix of organized sines of directions

n <- length(out.x)

CosOut <- rep(NA, n) # for interpolated cosine
SinOut <- CosOut # for interpolated sin

p <- 1:length(X)
q <- 1:length(Y)

for(i in 1:n)
{
        xx <- out.x[i]
        yy <- out.y[i]
        Vert=FALSE; Horiz=FALSE
        if(sum(X==xx)==1) Vert=TRUE
        if(sum(Y==yy)==1) Horiz=TRUE

        if(Vert==FALSE & Horiz==FALSE)
        {
                west <- max(p[X <= xx])
                east <- west + 1
                south <- min(q[Y <= yy])
                north <- south - 1
                x.west <- X[west]
                x.east <- X[east]
                y.south <- Y[south]
                y.north <- Y[north]
                cos.nw <- U[north,west]
                cos.ne <- U[north,east]
```

```
cos.sw <- U[south,west]
cos.se <- U[south,east]
sin.nw <- V[north,west]
sin.ne <- V[north,east]
sin.sw <- V[south,west]
sin.se <- V[south,east]

m <- (y.north-y.south)/(x.east-x.west) # 1 if vert res=horiz res
b <- y.north-m*x.east
ydiag <- m*xx+ b
if(yy <= ydiag) # On diagonal or in lower triangular
{
        # Fit plane to lower triangular
        AB <- c(x.east-x.west, 0, cos.se-cos.sw)
        AC <- c(x.east-x.west, y.north-y.south, cos.ne-cos.sw)
        # Coefficients of cross product AB X AC
        a <- AB[2]*AC[3]-AB[3]*AC[2]
        b <- AB[3]*AC[1]-AB[1]*AC[3]
        c <- AB[1]*AC[2]-AB[2]*AC[1]
        CosOut[i] <- cos.sw + (a*(x.west-xx) + b*(y.south-yy))/c
        # Fit plane to lower triangular
        AB <- c(x.east-x.west, 0, sin.se-sin.sw)
        AC <- c(x.east-x.west, y.north-y.south, sin.ne-sin.sw)
        # Coefficients of cross product AB X AC
        a <- AB[2]*AC[3]-AB[3]*AC[2]
        b <- AB[3]*AC[1]-AB[1]*AC[3]
        c <- AB[1]*AC[2]-AB[2]*AC[1]
        SinOut[i] <- sin.sw + (a*(x.west-xx) + b*(y.south-yy))/c
}
else
{
        # In upper triangular
        AC <- c(x.east-x.west, y.north-y.south, cos.ne-cos.sw)
        AD <- c(0,y.north-y.south, cos.nw-cos.sw)
        a <- AC[2]*AD[3]-AC[3]*AD[2]
        b <- AC[3]*AD[1]-AC[1]*AD[3]
```

```
                c <- AC[1]*AD[2]-AC[2]*AD[1]
                CosOut[i] <- cos.sw + (a*(x.west-xx) + b*(y.south-yy))/c
                AC <- c(x.east-x.west, y.north-y.south, sin.ne-sin.sw)
                AD <- c(0,y.north-y.south, sin.nw-sin.sw)
                a <- AC[2]*AD[3]-AC[3]*AD[2]
                b <- AC[3]*AD[1]-AC[1]*AD[3]
                c <- AC[1]*AD[2]-AC[2]*AD[1]
                SinOut[i] <- sin.sw + (a*(x.west-xx) + b*(y.south-yy))/c
        }
}
else if(Vert==TRUE & Horiz==FALSE)
{
        p1 <- p[X==xx] # Column of vert grid line
        q1 <- min(q[Y < yy]) # Even spacing is not assumed
        q2 <- q1 - 1
        cos1 <- U[q1, p1]
        cos2 <- U[q2, p1]
        CosOut[i] <- cos1 + (cos2-cos1)*(yy-Y[q1])/(Y[q2]-Y[q1])
        sin1 <- V[q1, p1]
        sin2 <- V[q2, p1]
        SinOut[i] <- sin1 + (sin2- sin1)*(yy-Y[q1])/(Y[q2]-Y[q1])
}
else if(Vert==FALSE & Horiz==TRUE)
{
        q1 <- q[Y==yy] # Row of horiz grid line
        p1 <- max(p[X < xx])
        p2 <- p1 + 1
        cos1 <- U[q1, p1]
        cos2 <- U[q1, p2]
        CosOut[i] <- cos1 + (cos2-cos1)*(xx-X[p1])/(X[p2]-X[p1])
        sin1 <- V[q1, p1]
        sin2 <- V[q1, p2]
        SinOut[i] <- sin1 + (sin2-sin1)*(xx-X[p1])/(X[p2]-X[p1])
}
else # Vert==TRUE & Horiz==TRUE
{
```

```
                    CosOut[i] <- U[q[Y==yy], p[X==xx]]
                    SinOut[i] <- V[q[Y==yy], p[X==xx]]
                }
        }
        dir <- atan2(SinOut, CosOut)
        return(list(x=out.x, y=out.y, direction=dir))
}
```

K.12 CircMedianPolish

```
CircMedianPolish <- function(x, y, h, v, delta, MaxIter=20, tol=0.01)
{
	# 2007-07-06
	# Inputs:
	#	x = vector of longitudes (x coordinates); y=vector of latitudes (y coordinates).
	#	h = vector of Horizontal components of vectors; v=vector of Vertical component of vectors.
	#	delta is lattice horizontal and vertical spacing.  Why should the horiz & vert spacings be different?
	# Assumptions:
	#	Direction measured on regular lattice, but doesn't have a specific order.  Outputs are ordered in the code.
	# Process:
	#	1) Vector h is organized into matrix H.  Vector v is organized into matrix V.
	#	2) Median polish is performed on matrices H and V separately.
	#	3) Polished direction is determined by applying atan2 to (V, H).
	# Outputs:
	#	X = vector of ordered longitudes
	#	Y = vector of ordered latitudes
	#	polished.dir = vector of ordered polished directions
	#	raw.dir = vector of ordered raw directions, which may have NAs

	MinX <- min(x); MaxX <- max(x); MinY <- min(y); MaxY <- max(y)
	Cols <- 1 + (x -MinX)/delta # For indexing h and v into matrix format
	Rows <- 1 + (MaxY - y)/delta # For indexing h and v into matrix format
	H <- matrix(data = NA, nrow=1 +(MaxY-MinY)/delta, ncol=1+ (MaxX-MinX)/delta); V <- H; Dirs <- H
	# Organize the components and data
	H[cbind(Rows, Cols)] <- h
	V[cbind(Rows, Cols)] <- v
	Dirs[cbind(Rows, Cols)] <- atan2(v,h)

	polish.H <- medpolish(H, maxiter=MaxIter, trace.iter=TRUE, na.rm=T, eps=tol) # Convergence progress reported
	polish.V <- medpolish(V, maxiter=MaxIter, trace.iter=TRUE, na.rm=T, eps=tol)
	# polished.H1 <- H - polish.H$residuals; polished.V1 <- V - polish.V$residuals # NAs
	polished.H <- polish.H$overall + outer(polish.H$row, polish.H$col, FUN="+") # no NAs
	polished.V <- polish.V$overall + outer(polish.V$row, polish.V$col, FUN="+" )
	polished.dir <- atan2(polished.V, polished.H)
```

```
	polished.dir <- as.vector(polished.dir)
	polished.dir[polished.dir < 0] <- polished.dir[polished.dir < 0] + 2*pi
	polished.dir[polished.dir > 2*pi] <- polished.dir[polished.dir > 2*pi] - 2*pi

	raw.dir <- as.vector(Dirs) # has NAs, is organized
	filterNA <- is.na(raw.dir)
	raw.dir2 <- raw.dir[!filterNA]
	raw.dir2[raw.dir2 < 0] <- raw.dir2[raw.dir2 < 0] + 2*pi
	raw.dir2[raw.dir2 > 2*pi] <- raw.dir2[raw.dir2 > 2*pi] - 2*pi
	raw.dir[!filterNA] <- raw.dir2

	x <- rep(seq(MinX, MaxX, by=delta), each=1 + (MaxY-MinY)/delta)
	y <- rep(seq(MaxY, MinY, by=-delta), 1 + (MaxX-MinX)/delta)
	return(list(x=x, y=y, polished.dir=polished.dir))
}
```

K.13 AssessStandardization

```
AssessStandardization <- function(CircDistr2="vM", Rho2=.75, CovModel2="spherical", Range2=10, Ext2=3, nSim=30, nPoints=400,
        OVERFIT=TRUE, ZLevels, CircLevels)
{
        # 2008-2-15.0600
        # Generate a nPoints x nSim and compute bivkde of QQ plots Standard Norm, Circular, Unif
        # CircDistr2 = circular probability distribution in {"U", "vm", "Tri", "WrC", "Card"}
        #     to visually compare mean and variability with and without standardization
        # Range of parameter Rho2
        #     For triangular, 0 < Rho2 <= 4/pi^2
        #     For cardioid, 0 < Rho2 <= 0.5
        #     For vM and wrapped Cauchy, 0 < Rho2 < 1, 1== degenerate
        #     For uniform, Rho2 = 0
        # CovModel2 = Any covariance model valid in geoR or RandomFields
        # Range2 = Distance at which RV are not correlated
        # Ext2 = Multiplies Range2.  Range2 x Ext2 is width and height of RF.  Default protects sill against edge effects.
        # nSim = Number of simulations
        # ZLevels = Number of color bins for filled.contour plots of QQ norm density
        # CircLevels = Number of color bins for filled.contour plots of QQ circular density
        # nPoints = Number of points per simulation
        # OverFit=TRUE, or standardization (centering and rescaling realization of the GRV to mean 0 sd 1) results in closer fit
        #     for qualitative evaluation of the CRV.  Undesirable effects are loss of independence of the marginal GRVs, biased GRF
        #     covariance, and biased testing.  Standardization is suitable for demonstration with closer fit, visualization, and
        #     illustrations.  Do not standardize for purposes of simulation and testing.  OverFit=FALSE, or non-standardization (default)
        #     includes expected variation from transformation of variation in mean and sd of sample of GRV.

        a <- ifelse(nPoints <= 10, 3/8, 1/2)
        CumProb <- ((1:nPoints)- a)/(nPoints + 1 - 2*a) # Vector of symmetric cumulative probabilities for QQ plots
        ZQuantiles <- qnorm(CumProb, mean=0, sd=1, lower.tail = TRUE)
        ZQuantiles <- rep(ZQuantiles, nSim)
```

```
# Compute circular quantiles
ThetaQuantiles <- vector(mode="numeric", length=nPoints)
if(CircDistr2=="U") { ThetaQuantiles <- -pi + 2*pi*CumProb } else
if(CircDistr2 == "Tri")
{
	if(Rho2==0 | Rho2 > 4/pi^2) stop("Tri: 0 < Rho <= 4/pi^2")
	filter <- CumProb < 0.5
	u1 <- CumProb[filter]
	a <- Rho2/8
	b <- (4+pi^2*Rho2)/(8*pi)
	c <- 0.5 - u1
	q <- -.5*(b+sqrt(b^2-4*a*c))
	ThetaQuantiles[filter] <- c/q

	u2 <- CumProb[!filter]
	a <- -Rho2/8
	b <- (4+pi^2*Rho2)/(8*pi)
	c <- 0.5 - u2
	q <- -.5*(b+sqrt(b^2-4*a*c))
	ThetaQuantiles[!filter]<- c/q
} else
{
	# For non-uniform circular distributions first get circular CDF in order to get ThetaQuantiles
	CircScale <- seq(-pi, pi, length=2*pi/.01) # Circular support from -pi to +pi, 629 elements, d~.01, CircScale[315] is zero
	n <- length(CircScale)
	if(CircDistr2=="vM")
	{
		if(Rho2==0 | Rho2 >= 1) stop("vM: 0 < Rho < 1")
		require(CircStats)
		CircProb <- rep(-1, n)
		Kappa=A1inv(Rho2) # N. I Fisher, Statistical Analysis of Circular Data, 2000 p. 49
		# As theta increases from -pi, pvm increases from .5
		for(i in 1:n) {CircProb[i] <- pvm(CircScale[i], mu=0, kappa=Kappa)}
		filter <- CircScale < 0
		CircProb[filter] <- CircProb[filter] - 0.5
```

```
			CircProb[!filter] <- CircProb[!filter] + 0.5
		} else
		if(CircDistr2=="Card")
		{
			if(Rho2==0 | Rho2 > 0.5) stop("Cardioid: 0 < Rho <= 0.5")
			CircProb <- (CircScale + pi + 2*Rho2*sin(CircScale))/(2*pi)
		} else
		if(CircDistr2=="WrC")
		{
			if(Rho2==0 | Rho2 >= 1) stop("Wrapped Cauchy: 0 < Rho < 1")
			Angles1 <- CircScale[CircScale < 0] + 2*pi
			Angles2 <- CircScale[CircScale >= 0]
			prob1 <- 0.5 - acos(((1+Rho^2)*cos(Angles1) - 2*Rho)/(1 + Rho^2 - 2*Rho * cos(Angles1)))/(2*pi)
			prob2 <- 0.5 + acos(((1+Rho^2)*cos(Angles2) - 2*Rho)/(1 + Rho^2 - 2*Rho * cos(Angles2)))/(2*pi)
			CircProb <-c(prob1, prob2)
		}
		CircProb[1] <- 0; CircProb[n] <- 1 # For any numerical imprecision

		# Get Quantiles From Inverse Circular CDF for distributions not uniform
		DeltaTh <- CircScale[2] - CircScale[1]
		for(i in 1:nPoints)
		{
			p <- CumProb[i]
			a <- max((1:n)[CircProb <= p]) # Left index
			if(CircProb[a]==p) {r <- 0} else {r <- (p - CircProb[a])/(CircProb[a+1]- CircProb[a])}
			ThetaQuantiles[i] <- CircScale[a] + r*DeltaTh
		}
	}

	ThetaQuantiles <- rep(ThetaQuantiles, nSim)
```

```
        Thetasort <- Zsort <- matrix(data = NA, nrow = nPoints, ncol = nSim)
        for(i in 1:nSim)
        {
                output <- SimulateCRF(N=nPoints, CircDistr=CircDistr2, Rho=Rho2, Range=Range2, Ext=Ext2, CovModel= CovModel2,
                        OverFit=OVERFIT)
                Z <- output$Z
                Zsort[,i] <- sort(Z)

                Theta <- output$direction
                Thetasort[,i] <- sort(Theta)
        }

        require(KernSmooth)
        biv.Z <- bkde2D(x=cbind(ZQuantiles, as.vector(Zsort)), bandwidth=c(0.1,0.1), range.x=list(c(-3,3), c(-3,3)))
        biv.C <- bkde2D(x=cbind(ThetaQuantiles, as.vector(Thetasort)), bandwidth=c(0.1,0.1), range.x=list(c(-pi,pi), c(-pi,pi)))

        # QQ Standard Normmal
        dev.set(which=2)
        par(mgp=c(1.5,.5,0), mar=c(3.2,2.8,1.7,0.1), cex.main=.75, cex.lab=.75, cex.axis=.75)
        filled.contour(biv.Z$x1, biv.Z$x2, biv.Z$fhat, color = terrain.colors, levels=ZLevels,
                plot.title = title(main = "Density of QQ Standard Normal Points", xlab = "Theoretical Quantile", ylab = "Sample Quantile"))

        # QQ Circular
        dev.set(which=3)
        par(mgp=c(1.5,.5,0), mar=c(3.2,2.8,1.7,0.1), cex.main=.75, cex.lab=.75, cex.axis=.75)
        filled.contour(biv.C$x1, biv.C$x2, biv.C$fhat, color = terrain.colors, xlim=c(-pi,pi), ylim=c(-pi,pi), levels=CircLevels,
                plot.title = title(main = "Density of QQ Circular Points", xlab = "Theoretical Quantile", ylab = "Sample Quantile"))
}
```

K.14 MakeCosineData

```
MakeCosineData <-function(nSim=400, N=400, model, Range, Ext=2, CircDistr, Rho, Resolution=0.01, ...)
{
	# 2008-8-14.0550
	# Make cosine data from transformation of standard normal GRF to fit cosine models

	# Input Arguments
	# nSim: Number of simulations
	# N: Number of spatial locations per simulation
	# model: Name of spatial correlation function, see package geoR Help cov.spatial
	# Range: The range parameter of the covariance model
	# Ext: Range*Ext is horizontal and vertical length of sample space
	# CircDistr: Circular distribution in {U, vM, WrC, Tri, Card},
	# Rho: Mean resultant length parameter
	#	For triangular, 0 < Rho <= 4/pi^2
	#	For cardioid, 0 < Rho <= 0.5
	#	For vM and wrapped Cauchy, 0 < Rho < 1, 1== degenerate
	#	For uniform, Rho = 0, not required
	# Resolution: For interpolation of theta on CDF for non-closed form inverse CDFs

	# Values
	# matrix of lag distances in column 1 and cosineogram ordinates in column 2

	require(CircStats)
	require(RandomFields)
	require(KernSmooth)
	x <- c()
	CosineoG <- c()
	mean <- 0
	variance <- 1
	nugget <- 0

	for (i in 1:nSim)
	{
		direction <- vector(mode="numeric", length=N)
```

```
X= runif(N, max=Range*Ext)
Y= runif(N, max=Range*Ext)
GRV <- GaussRF(grid=FALSE, x=X, y=Y, model=model, param=c(mean, variance, nugget, scale=Range, ...))
CumProbZ <- pnorm(GRV, mean=0, sd=1, lower.tail = TRUE)

if(CircDistr=="U") {direction <- -pi + 2*pi*CumProbZ} else
if(CircDistr == "Tri")
{
        if(Rho==0 | Rho > 4/pi^2) stop("Tri: 0 < Rho <= 4/pi^2")
        filter <- CumProbZ < 0.5
        u1 <- CumProbZ[filter]
        a <- Rho/8; b <- (4+pi^2*Rho)/(8*pi); c <- 0.5 - u1
        q <- -.5*(b+sqrt(b^2-4*a*c))
        direction[filter] <- c/q

        u2 <- CumProbZ[!filter]
        a <- -Rho/8; b <- (4+pi^2*Rho)/(8*pi); c <- 0.5 - u2
        q <- -.5*(b+sqrt(b^2-4*a*c))
        direction[!filter]<- c/q
} else
{
        # For OTHER circular distributions compute table of circular CDF and interpolate
        CircScale <- seq(-pi, pi, length=2*pi/Resolution)
        # With resolution=.01, circular support from -pi to +pi has 629 elements, delta ~0.01000507, CircScale[315] = 0
        n <- length(CircScale)
        if(CircDistr == "vM")
        {
                if(Rho==0 | Rho >= 1) stop("vM: 0 < Rho < 1")
                CircProb <- rep(-1, n)
                Kappa=A1inv(Rho) # N. I Fisher, Statistical Analysis of Circular Data, 2000 p. 49
                # As direction increases from -pi, pvm increases from .5
                for(i in 1:length(CircScale)) CircProb[i] <- pvm(CircScale[i], mu=0, kappa=Kappa)
                filter <- CircScale < 0
                CircProb[filter] <- CircProb[filter] - 0.5
                CircProb[!filter] <- CircProb[!filter] + 0.5
        } else
```

```
				if(CircDistr == "Card")
				{
					if(Rho==0 | Rho > 0.5) stop("Cardioid: 0 < Rho <= 0.5")
					CircProb <- (CircScale + pi + 2*Rho*sin(CircScale))/(2*pi)
				} else
				if(CircDistr == "WrC")
				{
					if(Rho==0 | Rho >= 1) stop("Wrapped Cauchy: 0 < Rho < 1 ")
					Angles1 <- CircScale[CircScale < 0]
					Angles2 <- CircScale[CircScale >= 0]
					prob1 <- 0.5 - acos(((1+Rho^2)*cos(Angles1) - 2*Rho)/(1 + Rho^2 - 2*Rho * cos(Angles1)))/(2*pi)
					prob2 <- 0.5 + acos(((1+Rho^2)*cos(Angles2) - 2*Rho)/(1 + Rho^2 - 2*Rho * cos(Angles2)))/(2*pi)
					CircProb <-c(prob1, prob2)
				}
				CircProb[1] <- 0; CircProb[n] <- 1

				# Interpolation
				DeltaTh <- CircScale[2] + pi
				for(i in 1:N)
				{
					p <- CumProbZ[i]  # Cumulative prob of GRV
					a <- max((1:n)[CircProb <= p]) # Index
					if(a==n) {r <- 0} else
					{
						if(CircProb[a]==p) {r <- 0} else {r <- (p -CircProb[a])/( CircProb[a+1] -CircProb[a])}
					}
					direction[i] <- CircScale[a] + r*DeltaTh
				}
			}
			cosineog.out <- CosinePlots(x=X, y=Y, directions=direction, Lag.n.Adj=1, Lag=seq(0, Range, by=0.25), Plot=FALSE)
			CosineoG <- c(CosineoG, cosineog.out$cosine)
		}

		fitCosineoG <- locpoly(rep(seq(0, Range, by=0.25),nSim), CosineoG, bandwidth = 0.15)
		return(list(fitCosineoG=fitCosineoG, CosineoG=CosineoG ))
}
```

K.15 FitCosineData

```
FitCosineData <- function(output=U.s.5$fitCosineoG, output.distr="U", code="0.5", GRF.model="spherical", ...)
{
	# 2008-11-08.0759

	require(RandomFields)

	rhoMax <- 1 # WrC, vM
	if(output.distr=="card") rhoMax <- .5
	if(output.distr=="tri") rhoMax <- 4/pi^2
	if(output.distr=="U") rhoMax <- 0

		if(code == "0.5")   {Fraction=0.00; Range=5}
	if(code == "0.10")  {Fraction=0.00; Range=10}
	if(code == "05.5")  {Fraction=0.05; Range=5}
	if(code == "05.10") {Fraction=0.05; Range=10}
	if(code == "95.5")  {Fraction=0.95; Range=5}
	if(code == "95.10") {Fraction=0.95; Range=10}

	Xlim=c(0, Range); Ylim=c(0, 1)
	Rho <- Fraction*rhoMax

	u <- output$x/Range # Standardized distance required vy CovarianceFct
	v <- Rho^2+(1-Rho^2)*CovarianceFct(x=u, model=GRF.model, param=c(mean=0,variance=1,nugget=0,scale=1, ...),
		dim=1, fctcall="Covariance")
	par(mfrow=c(2,1))
	plot(u,v, xlim=Xlim, ylim=Ylim, ty="l", col=2, lwd=2, xlab="Distance", ylab="Cosines")
	lines(output, col=1)
	abline(h=1, col="grey"); abline(h=Rho^2, col="grey"); abline(v=0, col="grey"); abline(v=Range, col="grey")
	plot(u, output$y-v, xlim=Xlim, ty="l", main=paste("Mean Abs Difference= ", round(mean(abs(output$y-v)),6)),
		xlab="Distance", ylab="Difference of Cosines")
	abline(h=0, col="grey")
}
```

K.16 FitOceanWind

```
# For Figure 4-8

FitOceanWind <- function(input, Rho, Nugget, Range, GRF.model, ...)
{
	# 2008-11-08.0759
	require(RandomFields)
	delta <- seq(0, .95, by=.05)
	u <- c(delta, input$x)/Range
	v <- Rho^2+(1-Rho^2-Nugget)*CovarianceFct(x=u, model=GRF.model, param=c(mean=0,variance=1,nugget=0,scale=1, ...),
		dim=1, fctcall="Covariance")
	par(mfrow=c(2,1))
	plot(c(delta, input$x), v, ty="l", col=2,lwd=2, xlab="Distance", ylab="Cosines", xlim=c(0, Range), ylim=c(0,1))
	lines(input, col=1)
	abline(h=1, col="grey")
	abline(h=Rho^2, col="grey")
	abline(v=0, col="grey")
	abline(v=Range, col="grey")

	v2 <- v[-c(1:length(delta))]
	filter <- input$x <= Range
	MAD <- round(mean( abs( v2[filter]-input$y[filter] ) ), 6)
	plot(input$x, v2 - input$y, ty="l", main=paste("Mean Abs Difference= ", MAD),
		xlab="Distance", ylab="Difference of Cosines", xlim=c(0,Range))
	abline(h=0, col="grey")
}
```

K.17 3DPolarMainMagnetic

```
MainMagnetic <- function(Data="1900.dat")
{
	# 2008-06-23
	# The data must be sorted first by latitude decreasing and second by longitude increasing: north to south, and west to east.
	#
	# Variables:
	# id	long	lat	direction	magnitude	x.north y.east	z.vertical
	# id is a sequential integer.
	# long is longitude from -180º to +180º by 1º.
	# lat is latitude from -89º to +89º by 1º.  Data are not defined at poles.
	# Direction is from 0 to 2pi radians.
	# Magnitude is the magnitude of the resultant of the east and north components.

	require(rgl)
	require(fields)
	data <- read.table(file=Data,header=T)
	long <-  data[,2]*pi/180 # longitude in radians
	colat <- (90 - data[,3])*pi/180 # colatitude in radians
	Direction <- data[,4]*180/pi # Direction in degrees
	Magnitude <- data[,5]
	MaxMag <- max(Magnitude[!is.na(Magnitude)])

	# VERTICES
	x <- Magnitude*sin(colat)*cos(long)
	y <- Magnitude*sin(colat)*sin(long)
	z <- Magnitude*cos(colat)

	# Color of direction from GYRB color wheel
	c <- vector(mode="character", length=length(x))

	filter <- (Direction < 90); ZeroToOne <- Direction[filter]/90
	c[filter] <- rgb(255*ZeroToOne, 255, 0, maxColorValue=255)

	filter <- (Direction >= 90 & Direction < 180); ZeroToOne <- (Direction[filter]-90)/90
```

```
c[filter] <- rgb(255, 255*(1 - ZeroToOne), 0, maxColorValue=255)

filter <- (Direction >= 180 & Direction < 270); ZeroToOne <- (Direction[filter] - 180)/90
c[filter] <- rgb(255*(1 - ZeroToOne), 0, 255*ZeroToOne, maxColorValue=255)

filter <- (Direction >= 270); ZeroToOne <- (Direction[filter] - 270)/90
c[filter] <- rgb(0, 255*ZeroToOne, 255*(1 - ZeroToOne), maxColorValue=255)

# Matrix rows are lat 89, 88, 87, ..., -88, -89
X <- matrix(data=x, byrow=T, ncol=361)# 179 rows x 361 cols
Y <- matrix(data=y, byrow=T, ncol=361)
Z <- matrix(data=z, byrow=T, ncol=361)
C <- matrix(data=c, byrow=T, ncol=361)

# Add continental profiles for reference
data(world.dat) # range of longitude (x) is -180, +180; range of latitude (y) is -78.5, 83.7
filter <- !is.na(world.dat$x)
xx <- world.dat$x[filter]
yy <- world.dat$y[filter]
C[cbind(90 - round(yy,0), 181 + round(xx,0))] <- rgb(0,0,0)

# q=quadrilateral primitive
Xq <- vector(mode="numeric", len=178*360*4); Yq <- Xq; Zq <- Xq
Cq <- vector(mode="character", len=178*360*4)
QuadFilter <- c( rep(c(1,2,3,4), 178*360) ) # Earth magnetic at 1º resolution + no poles has 64,080 quads.

Xq[QuadFilter == 1] <- X[-179,-361]
Yq[QuadFilter == 1] <- Y[-179,-361]
Zq[QuadFilter == 1] <- Z[-179,-361]
Cq[QuadFilter == 1] <- C[-179,-361]

Xq[QuadFilter == 2] <- X[-179,-1]
Yq[QuadFilter == 2] <- Y[-179,-1]
Zq[QuadFilter == 2] <- Z[-179,-1]
Cq[QuadFilter == 2] <- C[-179,-1]
```

```
        Xq[QuadFilter == 3] <- X[-1,-1]
        Yq[QuadFilter == 3] <- Y[-1,-1]
        Zq[QuadFilter == 3] <- Z[-1,-1]
        Cq[QuadFilter == 3] <- C[-1,-1]

        Xq[QuadFilter == 4] <- X[-1,-361]
        Yq[QuadFilter == 4] <- Y[-1,-361]
        Zq[QuadFilter == 4] <- Z[-1,-361]
        Cq[QuadFilter == 4] <- C[-1,-361]
        rgl.quads(Xq,Yq,Zq,color=Cq,lit=FALSE)
        L <- 1.33*MaxMag
        rgl.lines(c(0,L), c(0,0), c(0,0), color=2, size=4)
        rgl.lines(c(0,0), c(0,L), c(0,0), color=3, size=4)
        rgl.lines(c(0,0), c(0,0), c(0,L), color=4, size=4)
        rgl.lines(c(0,-L), c(0,0), c(0,0), color=2, size=1)
        rgl.lines(c(0,0), c(0,-L), c(0,0), color=3, size=1)
        rgl.lines(c(0,0), c(0,0), c(0,-L), color=4, size=1)

        rgl.viewpoint(fov=1)
}
```

K.18 Circular Kriging Variance

```
PlotCKVar <- function(rho=0, ng=0, range=8, x.legend=1, y.legend=1)
{
	d01 <- seq(0,10,0.05); d02 <- sqrt(d01^2 + 1) # distance between observations=1
	c01.e <- rho^2 + (1-ng-rho^2)*exp(-3*d01/range)
	c02.e <- rho^2 + (1-ng-rho^2)*exp(-3*d02/range)
	c.e <- rbind(c01.e, c02.e)
	k.e <- rho^2 + (1-ng-rho^2)*exp(-3*1/range)
	K.e <- matrix(data=c(1,k.e,k.e,1), ncol=2, byrow=TRUE)
	K.e.inv <-solve(K.e)
	ckvar.e <- 2 - 2*sqrt(  diag(t(c.e) %*% K.e.inv %*% c.e)  )

	c01.g <- rho^2 + (1-ng-rho^2)*exp(-3*(d01/range)^2)
	c02.g <- rho^2 + (1-ng-rho^2)*exp(-3*(d02/range)^2)
	c.g <- rbind(c01.g, c02.g)
	k.g <- rho^2 + (1-ng-rho^2)*exp(-3*(1/range)^2)
	K.g <- matrix(data=c(1,k.g,k.g,1), ncol=2, byrow=TRUE)
	K.g.inv <-solve(K.g)
	ckvar.g <- 2 - 2*sqrt( diag(t(c.g) %*% K.g.inv %*% c.g) )

	c01.s <- 1 - ng - (1-ng-rho^2)*(1.5*d01/range - 0.5*(d01/range)^3); c01.s[d01 > range] <- rho^2
	c02.s <- 1 - ng - (1-ng-rho^2)*(1.5*d02/range - 0.5*(d02/range)^3); c02.s[d02 > range] <- rho^2
	c.s <- rbind(c01.s, c02.s)
	k.s <- 1 - ng - (1-ng-rho^2)*(1.5*1/range - 0.5*(1/range)^3)
	if(1 > range) k.s <- rho^2
	K.s <- matrix(data=c(1,k.s,k.s,1), ncol=2, byrow=TRUE)
	K.s.inv <-solve(K.s)
	ckvar.s <- 2 - 2*sqrt( diag(t(c.s) %*% K.s.inv %*% c.s) )

	plot(d01, ckvar.e, ty="l", col=1, ylim=c(0,2), xlab="Distance",
		ylab="Circular Kriging Variance")
	lines(d01, ckvar.g, col="tan", lwd=3); lines(d01, ckvar.s, col=2, lty=2)
	legend(x=x.legend, y=y.legend, c("Exponential","Gaussian","Spherical"),
		lty = c(1, 1, 2), col=c(1, "tan", 2), lwd=c(1, 3, 1), cex=1.1)
}
```

Appendix L

R Command Line Input

L.1 Figures 3-4 to 3-8

```
out <- SimulateSill()

plot(1:1000, out$Cavg, type="l", xlab="Number of Simulations", ylab="Mean Cosine", main="Cardioid", ylim=c(0.05,0.10))
abline(h=0.25^2, lty=2)

plot(1:1000, out$Tavg, type="l", xlab="Number of Simulations", ylab="Mean Cosine", main="Triangular", ylim=c(0.02,0.07))
abline(h=(.5*4/pi^2)^2, lty=2)

plot(1:1000, out$Uavg, type="l", xlab="Number of Simulations", ylab="Mean Cosine", main="Uniform", ylim=c(-0.01,0.04))
abline(h=0, lty=2)

plot(1:1000,out$VMavg, type="l", xlab="Number of Simulations", ylab="Mean Cosine", main="Von Mises", ylim=c(0.77,0.82))
abline(h=0.798, lty=2)

plot(1:1000,out$WCavg, type="l", xlab="Number of Simulations", ylab="Mean Cosine", main="Wrapped Cauchy", ylim=c(0.1,0.15))
abline(h=exp(-2), lty=2)
```

L.2 Figure 3-13

```
years <- sort(unique(OceanWind[,1]))

# Smoothed average direction from CircDataimage
model.x <- CircDataimageGlobals$x.g[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow]
n.x <- length(model.x)
xmin <- min(model.x); xmax <- max(model.x)
model.y <- CircDataimageGlobals$y.g[CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol]
n.y <- length(model.y)
ymin <- min(model.y); ymax <- max(model.y)
model.dir <- CircDataimageGlobals$Direction[CircDataimageGlobals$StartRow:CircDataimageGlobals$EndRow,
CircDataimageGlobals$StartCol:CircDataimageGlobals$EndCol] # in radians

## Slice matrix and stack into vectors
model.xv <- rep(model.x, n.y); model.yv <- rep(model.y, each=n.x); model.dirv <- as.vector(model.dir)

## subset data to longitude and latitude once
filter1 <- OceanWind[ , 2]>=xmin & OceanWind[ , 2]<= xmax & OceanWind[ , 3]>=ymin & OceanWind[ , 3]<=ymax
data1 <- OceanWind[filter1, ]


SouthPolarCorr <- function(trend)
{
	#  Vectors of the sequence of ordinates of cosineograms are column binded
	Cosines <- c()
	for(year in years)
	{
		## subset the data to year
		filter2 <- data1[,1] == year; data2 <- data1[filter2,-1] # year value omitted

		## Organize data into a matrix using vector expressions
		u <- matrix(data=NA, nrow=n.x, ncol=n.y); v <- u
		Rows    <- round((data2[, 1] - xmin) + 1, digits = 0) # Indexing vector
		Columns <- round((data2[, 2] - ymin) + 1, digits = 0) # Indexing vector
		## test one observation per cell
```

```
            if (max(table(data2[,c(1,2)])) > 1) stop(paste("Duplicate observation in year ", year))

            u[cbind(Rows, Columns)] <- data2[, 3];   v[cbind(Rows, Columns)] <- data2[, 4]

            ## convert u and v to direction in 0 to 2pi, and slice and stack matrix to vector
            data.dir <- atan2(v, u); data.dirv <- as.vector(data.dir)

            residuals <- CircResidual(X=model.xv, Y=model.yv, Raw=data.dirv, Trend=trend, Plot=FALSE)

            cosines <- CosinePlots(x=residuals$x, y=residuals$y, directions=residuals$direction, Lag=1:40, Lag.n.Adj = 1, BinWAdj=1,
                  Plot=FALSE)$cosine
            Cosines <- cbind(Cosines, cosines)
      }
      return(Cosines)
}

Cosineograms <- SouthPolarCorr(trend=model.dirv)
Mean <- function(x) { mean(x, na.rm=TRUE, trim=0.2) }

par(mai=c(0.95, 0.75, 0.25, 0.25))
d <- 1:40
plot(rep(d,28), Cosineograms, xlab=”Distance In Degrees Of Longitude And Latitiude”, ylab=”Cosines”, col=”grey”, xlim=c(0, 15))
lines(c(0,d), c(1, apply(Cosineograms, 1, Mean)), col=2, lw=3)

# Exponential model
d2 <- seq(0, 40, by=0.25)
r = 3.8 # range
rho = sqrt(0.45) # mean resultant length
lines(d2, rho^2 + (1 - rho^2)*exp(-3*d2/r), col=4, lty=2, lwd=3 )
```

L.3 Figure 5-1, Simulated CRF

```
ygrid <- xgrid <- seq(-20,20,length=21)
xgrid <- rep(xgrid, 21)
ygrid <- rep(ygrid, each=21)
set.seed(9) # So Figures 5-1, 5-3, 5-4, and 5-7 will be consistent
crf1 <- SimulateCRF(CircDistr="vM", Rho=0.8, Mu=0, Range=10, CovModel="spherical", Grid=cbind(xgrid, ygrid), OverFit=TRUE)
require(CircSpatial)
par(mgp=c(2,1,0), mar=c(3.1, 3.1, 0.5, 0.5), cex=.8)
PlotVectors(x=crf1$x, y=crf1$y, h=cos(crf1$direction), v=sin(crf1$direction), AdjArrowLength=1.25, AdjHeadLength=.7)
```

L.4 Figure 5-3, Image of GRF

```
ygrid <- xgrid <- seq(-20,20,length=21)
xgrid <- rep(xgrid, 21); ygrid <- rep(ygrid, each=21)
set.seed(9)
crf1 <- SimulateCRF(CircDistr="vM", Rho=0.8, Mu=0, Range=10, CovModel="spherical", Grid=cbind(xgrid, ygrid), OverFit=TRUE)
image(seq(-20,20,length=21), seq(-20,20,length=21), matrix(data=crf1$Z, nrow=21, ncol=21, byrow=FALSE), col = terrain.colors(12),
        xlab=”x”, ylab=”y”)
```

L.5 Figure 5-4, Variogram and Inverted Cosineogram Similar

```
ygrid <- xgrid <- seq(-20,20,length=21)
xgrid <- rep(xgrid, 21)
ygrid <- rep(ygrid, each=21)
set.seed(9); CorrelationTransfer(CircDistr2="vM", Rho2=0.8, Range2=10, CovModel2="spherical", GRID=cbind(xgrid, ygrid), OVERFIT=TRUE)
```

L.6 Figure 5-5, Standardization

```
# Figure constructed from GRV and CRV windows

windows(); windows()
set.seed(10); AssessStandardization(CircDistr2="U", ZLevels=seq(0,.8,length=16), CircLevels=seq(0,0.34,length=16), OVERFIT=TRUE)
set.seed(10); AssessStandardization(CircDistr2="U", ZLevels=seq(0,.8,length=16), CircLevels=seq(0,0.34,length=16), OVERFIT=FALSE)
```

L.7 Figure 5-6, Variabiliy vs. ρ

```
# Figure constructed from CRV windows

windows(); windows()
set.seed(20); AssessStandardization(CircDistr2="Tri", Rho2=.95*4/pi^2, ZLevels=seq(0,.8,length=16), CircLevels=seq(0,0.4,length=16),
        OVERFIT=TRUE)
set.seed(20); AssessStandardization(CircDistr2="Tri", Rho2=.50*4/pi^2, ZLevels=seq(0,.8,length=16), CircLevels=seq(0,0.4,length=16),
        OVERFIT=TRUE)
set.seed(20); AssessStandardization(CircDistr2="Tri", Rho2=.25*4/pi^2, ZLevels=seq(0,.8,length=16), CircLevels=seq(0,0.4,length=16),
        OVERFIT=TRUE)
set.seed(20); AssessStandardization(CircDistr2="Tri", Rho2=.25*4/pi^2, ZLevels=seq(0,.8,length=16), CircLevels=seq(0,0.4,length=16),
        OVERFIT=FALSE)
```

L.8 Figure 5-8 and the Figures in Appendices C and D

```
ygrid <- xgrid <- seq(-20,20,length=21) # Source CosinePlots before running the following code
xgrid <- rep(xgrid, 21); ygrid <- rep(ygrid, each=21)
set.seed(9); AssessCRF(CircDistr2="vM",  Rho2=0.8,  Range2=10, CovModel2="spherical", GRID=cbind(xgrid, ygrid), OVERFIT=TRUE)

# Appendix C
AssessCRF(nPoints=400, CircDistr2="Card", Rho2=0.5*0.5, Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=TRUE)
AssessCRF(nPoints=400, CircDistr2="Tri",    Rho2=0.5*4/pi^2, Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=TRUE)
AssessCRF(nPoints=400, CircDistr2="U",      Range2=10, Ext2=3,CovModel2="spherical", OVERFIT=TRUE)
AssessCRF(nPoints=400, CircDistr2="WrC", Rho2=0.5,  Range2=10, Ext2=3,CovModel2="spherical", OVERFIT=TRUE)
```

```
# Appendix D
S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Card", Rho2=0.05,         Range2=10, Ext2=3,  CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Card", Rho2=0.05,         Range2=10, Ext2=3,  CovModel2="spherical", OVERFIT=FALSE)

S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Card", Rho2=0.95*0.5,    Range2=10, Ext2=3,  CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Card", Rho2=0.95*0.5,    Range2=10, Ext2=3,  CovModel2="spherical", OVERFIT=FALSE)

S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Tri",    Rho2=0.05,          Range2=10, Ext2=3,  CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Tri",    Rho2=0.05,          Range2=10, Ext2=3,  CovModel2="spherical", OVERFIT=FALSE)

S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Tri",    Rho2=0.95*4/pi^2, Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="Tri",    Rho2=0.95*4/pi^2, Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=FALSE)
S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="vM",   Rho2=0.05,           Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="vM",   Rho2=0.05,           Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=FALSE)

S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="vM",   Rho2=0.95,           Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="vM",   Rho2=0.95,           Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=FALSE)
```

```
S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="WrC", Rho2=0.05,          Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="WrC", Rho2=0.05,          Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=FALSE)

S=1000*runif(1)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="WrC", Rho2=0.95,          Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=TRUE)
set.seed(S)
AssessCRF(nPoints=400, CircDistr2="WrC", Rho2=0.95,          Range2=10, Ext2=3, CovModel2="spherical", OVERFIT=FALSE)
```

L.9 Figures 6-1 to 6-10

```
require(CircSpatial)

# Figure 6-1, Comprehensive Example - Global Trend Model of 121 Locations

model1.x<- 1:11; model1.y <- 11:1; model1.y <- rep(model1.y, 11); model1.x <- rep(model1.x, each=11)
model1.direction <- matrix(data=c(
   157, 141, 126, 113, 101, 90, 79, 67, 54, 40, 25, 152, 137, 123, 111, 100, 90, 80, 69, 57, 44, 30,
   147, 133, 120, 109,   99, 90, 81, 71, 60, 48, 35, 142, 129, 117, 107,   98, 90, 82, 73, 63, 52, 40,
   137, 125, 114, 105,   97, 90, 83, 75, 66, 56, 45, 132, 121, 111, 103,   96, 90, 84, 77, 69, 60, 50,
   127, 117, 108, 101,   95, 90, 85, 79, 72, 64, 55, 122, 113, 105,   99,   94, 90, 86, 81, 75, 68, 60,
   117, 109, 102,   97,   93, 90, 87, 83, 78, 72, 65, 112, 105,   99,   95,   92, 90, 88, 85, 81, 76, 70,
   107, 101,   96,   93,   91, 90, 89, 87, 84, 80, 75), ncol=11, byrow=TRUE)
model1.direction <- as.vector(model1.direction)*pi/180

par(mai=c(0.5, 0.5, 0.15, 0.15), ps=11)
plot( x=model1.x,y=model1.y, type='n', xlim=c(0,12), ylim=c(0,12), asp=1, xlab='', ylab='')
arrow.plot(a1=model1.x, a2=model1.y, u=cos(model1.direction),v=sin(model1.direction), arrow.ex=.06, length=.07, col='blue3', lwd=1, angle=18)
```

```
# Figure 6-2, Comprehensive Example - Simulated Sample of a Von Mises CRF,  ρ = √0.5

# Compute vM CRF of 121 observations, Rho=sqrt(0.5) so sill about 0.5, from GRF (Range=4, spherical covariance).
set.seed(666)
crf1<- SimulateCRF(CircDistr="vM", Rho=sqrt(0.5), Range=4, CovModel="spherical", Grid=cbind(model1.x, model1.y), OverFit=FALSE)
names(crf1) #  [1] "x"         "y"         "direction" "Z"

# Make sample
sample1.direction <- model1.direction + crf1$direction
par(mai=c(0.5, 0.5, 0.15, 0.15), ps=11)
plot( x=crf1$x,y=crf1$y, type='n', xlim=c(0,12), ylim=c(0,12), asp=1, xlab='', ylab='')
arrow.plot(a1=crf1$x, a2=crf1$y, u=cos(sample1.direction),v=sin(sample1.direction), arrow.ex=.06, length=.07, col=1, lwd=1, angle=18)


# Figure 6-3, Comprehensive Example - Estimate of the Global Trend Model

FitHoriz1 <- lm(cos(sample1.direction) ~ (model1.x + model1.y)^2)
FitVert1 <- lm(sin(sample1.direction)  ~ (model1.x + model1.y)^2)
fitted1.direction <- atan2(FitVert1$fitted.values, FitHoriz1$fitted.values)

plot( x=crf1$x,y=crf1$y, type='n', xlim=c(0,12), ylim=c(0,12), asp=1, xlab="", ylab="")
arrow.plot(a1=model1.x, a2=model1.y, u=cos(fitted1.direction),v=sin(fitted1.direction), arrow.ex=.06, length=.07, col="tan", lwd=3, angle=18)
arrow.plot(a1=model1.x, a2=model1.y, u=cos(model1.direction),v=sin(model1.direction), arrow.ex=.06, length=.07, col="blue3", lwd=1, angle=18)


# Figure 6-4, Comprehensive Example - Enlarged View of the Data, Model, and Residual Rotation

par(mai=c(0.45, 0.45, 0.1, 0.1), ps=11)
CircResidual(X=model1.x, Y=model1.y, Raw=sample1.direction, Trend=fitted1.direction, Plot=TRUE, AdjArrowLength=.9, xlim=c(8.0, 11.0),
     ylim=c(8, 11))

resids1 <- CircResidual(X=model1.x, Y=model1.y, Raw=sample1.direction, Trend=fitted1.direction, Plot=FALSE)
```

```
# Figure 6-5, Comprehensive Example - Points of the Cosineogram, and the Exponential, Gaussian,
# and Spherical Cosine Models of Spatial Correlation

par(mai=c(0.85, 0.75, 0.15, 0.15), ps=11)
CosinePlots(x=resids1$x, y=resids1$y, directions=resids1$direction, Plot=TRUE, Cloud=FALSE, Model=TRUE, nugget=0, Range=3.07,
sill=0.674, x.legend=0.4, y.legend=0.95, Lag=seq(0, 8, by=.375), BinWAdj=1)


# Figure 6-6, Enlarged View of The Kriging and the Residual Rotations

x2 <- seq(1,11, by=0.2); n <- length(x2); y2 <- x2; y2 <- rep(y2, n); x2 <- rep(x2, each=n)
krig1 <- KrigCRF(krig.x=x2, krig.y=y2, resid.x=resids1$x, resid.y=resids1$y, resid.direction= resids1$direction, Model="spherical", Nugget=0.0,
    Range=3.07, sill=0.674, Plot=FALSE)
par(mai=c(0.45, 0.45, 0.1, 0.1), ps=11)
plot(x=krig1$x, y=krig1$y, ty="n", xlim=c(8.0, 11.0), ylim=c(8.0, 11.0), asp=1)
arrow.plot(a1=krig1$x, a2=krig1$y, u=cos(krig1$direction), v=sin(krig1$direction), col="light grey", arrow.ex=0.08, xpd=FALSE,
    length = 0.08, angle=15)
arrow.plot(a1=resids1$x, a2=resids1$y, u=cos(resids1$direction), v=sin(resids1$direction), arrow.ex=0.05, xpd=FALSE, length= 0.06,
    col=2, lty=1, lwd=1)


# Figure 6-7, Comprehensive Example – Enlarged View of the Interpolation of the Global Trend Model

interp1 <- InterpDirection(in.x=model1.x, in.y=model1.y, in.direction=fitted1.direction, out.x=krig1$x, out.y=krig1$y)
plot(x=interp1$x, y=interp1$y, ty="n", xlim=c(8, 11.0), ylim=c(8.0, 11.0), asp=1)
arrow.plot(a1=model1.x, a2=model1.y, u=cos(fitted1.direction), v=sin(fitted1.direction), arrow.ex=0.09, xpd=FALSE, length = 0.09,
    col="tan", lwd=2, angle=17)
arrow.plot(a1=interp1$x, a2=interp1$y, u=cos(interp1$direction), v=sin(interp1$direction), col="purple", arrow.ex=0.05, xpd=FALSE,
    length = 0.06, angle=17)
```

```
# Figure 6-8, Comprehensive Example – Arrow Plot of the Circular Spatial Data Estimate, Enlarged View

estimate1.direction=interp1$direction + krig1$direction
par(mai=c(0.45, 0.45, 0.1, 0.1), ps=11)
plot(x=interp1$x, y=interp1$y, ty="n", xlim=c(8.0, 11.0), ylim=c(8.0, 11.0), asp=1)
arrow.plot(a1=interp1$x, a2=interp1$y, u=cos(estimate1.direction), v=sin(estimate1.direction), arrow.ex=0.09, xpd=FALSE,
     length = 0.09,col="gold", lwd=1, angle=17)
arrow.plot(a1=model1.x, a2=model1.y, u=cos(sample1.direction), v=sin(sample1.direction), col=1, arrow.ex=0.05, xpd=FALSE,
     length = 0.06, angle=17)


# Figure 6-9, Comprehensive Example – Circular Dataimage of the Circular Spatial Data Estimate

output1 <- data.frame(x=interp1$x, y=interp1$y, u=cos(estimate1.direction), v=sin(estimate1.direction), check.rows=TRUE, check.names=TRUE)
CircDataimage() # Use HSV Color Wheel, -105 rotation, arrow length multiplier 0.8, arrow spacing in pixels 3


# Figure 6-10, Comprehensive Example - Circular Kriging Variance with Observations on a Regular Grid

KrigCRF(krig.x=x2, krig.y=y2, resid.x=resids1$x, resid.y=resids1$y, resid.direction= resids1$direction, Model="spherical", Nugget=0.0,
        Range=3.07, sill=0.674, Plot=TRUE, PlotVar=TRUE)
```

L.10 Make Cosine Datasets

```
range <- 5

U.s.5 <- MakeCosineData(model="spherical",   CircDistr="U", Range=range)
U.e.5 <- MakeCosineData(model="exponential", CircDistr="U", Range=range)
U.g.5 <- MakeCosineData(model="gauss",    CircDistr="U", Range=range)
gc()

RhoMax <- 0.5
Distr <- "Card"
card.s.05.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
card.s.25.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
card.s.50.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
card.s.75.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
card.s.95.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
card.e.05.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
card.e.25.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
card.e.50.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
card.e.75.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
card.e.95.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
card.g.05.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
card.g.25.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
card.g.50.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
card.g.75.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
card.g.95.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()

RhoMax <- 4/pi^2
Distr <- "Tri"
tri.s.05.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
tri.s.25.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
tri.s.50.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
tri.s.75.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
tri.s.95.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
tri.e.05.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
```

```
tri.e.25.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
tri.e.50.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
tri.e.75.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
tri.e.95.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
tri.g.05.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
tri.g.25.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
tri.g.50.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
tri.g.75.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
tri.g.95.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()

RhoMax <- 1
Distr <- "vM"
vM.s.05.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
vM.s.25.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
vM.s.50.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
vM.s.75.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
vM.s.95.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
vM.e.05.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
vM.e.25.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
vM.e.50.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
vM.e.75.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
vM.e.95.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
vM.g.05.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
vM.g.25.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
vM.g.50.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
vM.g.75.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
vM.g.95.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()

RhoMax <- 1
Distr <- "WrC"
WrC.s.05.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
WrC.s.25.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
WrC.s.50.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
WrC.s.75.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
```

```
WrC.s.95.5 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
WrC.e.05.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
WrC.e.25.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
WrC.e.50.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
WrC.e.75.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
WrC.e.95.5 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
WrC.g.05.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
WrC.g.25.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
WrC.g.50.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
WrC.g.75.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
WrC.g.95.5 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()

range <- 10

U.s.10 <- MakeCosineData(model="spherical",   CircDistr="U", Range=range)
U.e.10 <- MakeCosineData(model="exponential", CircDistr="U", Range=range)
U.g.10 <- MakeCosineData(model="gauss",    CircDistr="U", Range=range)
gc()

RhoMax <- 0.5
Distr <- "Card"
card.s.05.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
card.s.25.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
card.s.50.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
card.s.75.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
card.s.95.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
card.e.05.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
card.e.25.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
card.e.50.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
card.e.75.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
card.e.95.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
card.g.05.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
card.g.25.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
card.g.50.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
card.g.75.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
```

```
card.g.95.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()

RhoMax <- 4/pi^2
Distr <- "Tri"
tri.s.05.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
tri.s.25.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
tri.s.50.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
tri.s.75.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
tri.s.95.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
tri.e.05.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
tri.e.25.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
tri.e.50.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
tri.e.75.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
tri.e.95.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
tri.g.05.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
tri.g.25.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
tri.g.50.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
tri.g.75.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
tri.g.95.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()

RhoMax <- 1
Distr <- "vM"
vM.s.05.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
vM.s.25.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
vM.s.50.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
vM.s.75.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
vM.s.95.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
vM.e.05.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
vM.e.25.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
vM.e.50.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
vM.e.75.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
vM.e.95.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
vM.g.05.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
vM.g.25.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
```

```
vM.g.50.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
vM.g.75.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
vM.g.95.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()

RhoMax <- 1
Distr <- "WrC"
WrC.s.05.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.05*RhoMax, Range=range)
WrC.s.25.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.25*RhoMax, Range=range)
WrC.s.50.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.50*RhoMax, Range=range)
WrC.s.75.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.75*RhoMax, Range=range)
WrC.s.95.10 <- MakeCosineData(model="spherical",   CircDistr=Distr, Rho=.95*RhoMax, Range=range)
WrC.e.05.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.05*RhoMax, Range=range)
WrC.e.25.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.25*RhoMax, Range=range)
WrC.e.50.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.50*RhoMax, Range=range)
WrC.e.75.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.75*RhoMax, Range=range)
WrC.e.95.10 <- MakeCosineData(model="exponential", CircDistr=Distr, Rho=.95*RhoMax, Range=range)
WrC.g.05.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.05*RhoMax, Range=range)
WrC.g.25.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.25*RhoMax, Range=range)
WrC.g.50.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.50*RhoMax, Range=range)
WrC.g.75.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.75*RhoMax, Range=range)
WrC.g.95.10 <- MakeCosineData(model="gauss",    CircDistr=Distr, Rho=.95*RhoMax, Range=range)
gc()
```

L.11 Figure M-1, Fitted Covariogram an Unbiased Estimator

```
MakeVariogData <- function(nSim=400, nPoints=400, Range=10, Ext=3, CovModel="spherical", Grid=NULL)
{
	# 2008-7-26.1112
	# Generate a nPoints x nPoints, fit variograms

	require(geoR)
	require(KernSmooth)
	VarioG.x <- c()
	VarioG.y <- c()
	for (i in 1:nSim)
	{
		if(is.null(Grid))
		{
			GRF <- grf(n=nPoints, xlims=c(0, Range*Ext), ylims=c(0, Range*Ext), cov.model=CovModel,
				nugget=0, cov.pars=c(1, Range), aniso.pars=Anisotropy, RF=TRUE, messages=FALSE) } else {
			GRF <- grf(grid=Grid, cov.model=CovModel,
				nugget=0, cov.pars=c(1, Range), aniso.pars=Anisotropy, RF=TRUE, messages=FALSE)
		}
		# GRF variogram
		vario.i <- variog(coords = GRF$coords, data = GRF$data, option = "bin", breaks=seq(0,Range, by=.2))
		VarioG.x <- c(VarioG.x, vario.i$u)
		VarioG.y <- c(VarioG.y, vario.i$v)
	}
	return(list(x=VarioG.x,y=VarioG.y))
	}
```

```
PlotFittedVaroG <- function(Data=fittedXY, Degree=1, BandW=.83, Range=10)
{
	# Transform MakeVariogData output to covariance
	x <- fittedXY$x
	y <- fittedXY$y
	fitVarioG <- locpoly(x, y, bandwidth = BandW, range.x=c(0,Range), degree=Degree)
	par(mfrow=c(2,1))
	plot(fitVarioG$x, 1-fitVarioG$y, ty="l", lwd=8, xlab="Distance Between Observations", ylab="Spherical Covariance")
	u <- fitVarioG$x
	v <- 1-(1.5*u/Range - 0.5*(u/Range)^3)
	lines(u,v, col=2)
	Difference <- (1-fitVarioG$y) - v
	plot(fitVarioG$x,Difference, ty="l", xlab="Distance Between Observations", ylab="Vertical Distance Between Curves",
		sub=paste("Average Vertical Distance between Curves", round(mean(abs(Difference)),4)))
	abline(h=0,col="grey80")
}
```

L.12 Plot Figures M-2, M-3, and M-4, Families of Curves

```
Plot1 <- function(x, Main, Pch, off1=0.3, off2=0.0)
{
	plot(x, ty="l", xlim=c(0,10.5), ylim=c(0,1), xlab="X", ylab="Cosine(X)", main=Main, col="tan")
	points(x$x[401] + off1, x$y[401] + off2, pch=Pch, col=2)
}

Plot2 <- function(x, Pch, off1=0.3, off2=0.0)
{
	lines(x, ty="l", col="tan")
	points(x$x[401] + off1, x$y[401] + off2, pch=Pch, col=2)
}
```

```
Plot3 <- function(x, Pch, off1=0.3, off2=0.0)
{
	lines(x, ty="l", col=1)
	points(x$x[401] + off1, x$y[401] + off2, pch=Pch, col=2)
}


# Figure M-2

par(mfrow=c(3,2), bty="n", mgp=c(2,1,0), mar=c(4.0, 3.5, 2.0, 0.0))
Plot1(card.e.05.10$fitCosineoG, Main="Cardioid", Pch="1")

Plot2(card.e.25.10$fitCosineoG, Pch="2")
Plot2(card.e.50.10$fitCosineoG, Pch="3")
Plot2(card.e.75.10$fitCosineoG, Pch="4")
Plot2(card.e.95.10$fitCosineoG, Pch="5")

Plot3(card.e.05.5$fitCosineoG,  Pch="1")
Plot3(card.e.25.5$fitCosineoG,  Pch="2")
Plot3(card.e.50.5$fitCosineoG,  Pch="3")
Plot3(card.e.75.5$fitCosineoG,  Pch="4")
Plot3(card.e.95.5$fitCosineoG,  Pch="5")

Plot1(tri.e.05.10$fitCosineoG, Main="Triangular", Pch="1")

Plot2(tri.e.25.10$fitCosineoG, Pch="2", 0.5, 0.01)
Plot2(tri.e.50.10$fitCosineoG, Pch="3")
Plot2(tri.e.75.10$fitCosineoG, Pch="4")
Plot2(tri.e.95.10$fitCosineoG, Pch="5")

Plot3(tri.e.05.5$fitCosineoG,  Pch="1")
Plot3(tri.e.25.5$fitCosineoG,  Pch="2", 0.5, 0.01)
Plot3(tri.e.50.5$fitCosineoG,  Pch="3")
Plot3(tri.e.75.5$fitCosineoG,  Pch="4")
Plot3(tri.e.95.5$fitCosineoG,  Pch="5")
```

```
Plot1(vM.e.05.10$fitCosineoG, Main="von Mises", Pch="1")
Plot2(vM.e.25.10$fitCosineoG, Pch="2")
Plot2(vM.e.50.10$fitCosineoG, Pch="3")
Plot2(vM.e.75.10$fitCosineoG, Pch="4")
Plot2(vM.e.95.10$fitCosineoG, Pch="5")

Plot3(vM.e.05.5$fitCosineoG,  Pch="1")
Plot3(vM.e.25.5$fitCosineoG,  Pch="2")
Plot3(vM.e.50.5$fitCosineoG,  Pch="3")
Plot3(vM.e.75.5$fitCosineoG,  Pch="4")
Plot3(vM.e.95.5$fitCosineoG,  Pch="5")

Plot1(WrC.e.05.10$fitCosineoG, Main="Wrapped Cauchy", Pch="1")

Plot2(WrC.e.25.10$fitCosineoG, Pch="2")
Plot2(WrC.e.50.10$fitCosineoG, Pch="3")
Plot2(WrC.e.75.10$fitCosineoG, Pch="4")
Plot2(WrC.e.95.10$fitCosineoG, Pch="5")

Plot3(WrC.e.05.5$fitCosineoG,  Pch="1")
Plot3(WrC.e.25.5$fitCosineoG,  Pch="2")
Plot3(WrC.e.50.5$fitCosineoG,  Pch="3")
Plot3(WrC.e.75.5$fitCosineoG,  Pch="4")
Plot3(WrC.e.95.5$fitCosineoG,  Pch="5")

Plot1(U.e.10$fitCosineoG, Main="Uniform", Pch="1")
Plot3(U.e.5$fitCosineoG,  Pch="1")


# Figure M-3

par(mfrow=c(3,2), bty="n", mgp=c(2,1,0), mar=c(4.0, 3.5, 2.0, 0.0))
Plot1(card.g.05.10$fitCosineoG, Main="Cardioid", Pch="1")

Plot2(card.g.25.10$fitCosineoG, Pch="2", 0.5)
Plot2(card.g.50.10$fitCosineoG, Pch="3")
```

```
Plot2(card.g.75.10$fitCosineoG, Pch="4")
Plot2(card.g.95.10$fitCosineoG, Pch="5")
Plot3(card.g.05.5$fitCosineoG,  Pch="1")
Plot3(card.g.25.5$fitCosineoG,  Pch="2", 0.5)
Plot3(card.g.50.5$fitCosineoG,  Pch="3")
Plot3(card.g.75.5$fitCosineoG,  Pch="4")
Plot3(card.g.95.5$fitCosineoG,  Pch="5")

Plot1(tri.g.05.10$fitCosineoG, Main="Triangular", Pch="1")

Plot2(tri.g.25.10$fitCosineoG, Pch="2", 0.5, 0.02)
Plot2(tri.g.50.10$fitCosineoG, Pch="3")
Plot2(tri.g.75.10$fitCosineoG, Pch="4")
Plot2(tri.g.95.10$fitCosineoG, Pch="5")

Plot3(tri.g.05.5$fitCosineoG,  Pch="1")
Plot3(tri.g.25.5$fitCosineoG,  Pch="2", 0.5)
Plot3(tri.g.50.5$fitCosineoG,  Pch="3")
Plot3(tri.g.75.5$fitCosineoG,  Pch="4")
Plot3(tri.g.95.5$fitCosineoG,  Pch="5")

Plot1(vM.g.05.10$fitCosineoG, Main="von Mises", Pch="1")

Plot2(vM.g.25.10$fitCosineoG, Pch="2")
Plot2(vM.g.50.10$fitCosineoG, Pch="3")
Plot2(vM.g.75.10$fitCosineoG, Pch="4")
Plot2(vM.g.95.10$fitCosineoG, Pch="5")

Plot3(vM.g.05.5$fitCosineoG,  Pch="1")
Plot3(vM.g.25.5$fitCosineoG,  Pch="2")
Plot3(vM.g.50.5$fitCosineoG,  Pch="3")
Plot3(vM.g.75.5$fitCosineoG,  Pch="4")
Plot3(vM.g.95.5$fitCosineoG,  Pch="5")

Plot1(WrC.g.05.10$fitCosineoG, Main="Wrapped Cauchy", Pch="1")
```

```
Plot2(WrC.g.25.10$fitCosineoG, Pch="2")
Plot2(WrC.g.50.10$fitCosineoG, Pch="3")
Plot2(WrC.g.75.10$fitCosineoG, Pch="4")
Plot2(WrC.g.95.10$fitCosineoG, Pch="5")

Plot3(WrC.g.05.5$fitCosineoG,  Pch="1")
Plot3(WrC.g.25.5$fitCosineoG,  Pch="2")
Plot3(WrC.g.50.5$fitCosineoG,  Pch="3")
Plot3(WrC.g.75.5$fitCosineoG,  Pch="4")
Plot3(WrC.g.95.5$fitCosineoG,  Pch="5")

Plot1(U.g.10$fitCosineoG, Main="Uniform", Pch="1")
Plot3(U.g.5$fitCosineoG,  Pch="1")


# Figure M-4

par(mfrow=c(3,2), bty="n", mgp=c(2,1,0), mar=c(4.0, 3.5, 2.0, 0.0))
Plot1(card.s.05.10$fitCosineoG, Main="Cardioid", Pch="1")

Plot2(card.s.25.10$fitCosineoG, Pch="2", 0.5, 0.01)
Plot2(card.s.50.10$fitCosineoG, Pch="3")
Plot2(card.s.75.10$fitCosineoG, Pch="4")
Plot2(card.s.95.10$fitCosineoG, Pch="5")

Plot3(card.s.05.5$fitCosineoG,  Pch="1")
Plot3(card.s.25.5$fitCosineoG,  Pch="2", 0.5, 0.01)
Plot3(card.s.50.5$fitCosineoG,  Pch="3")
Plot3(card.s.75.5$fitCosineoG,  Pch="4")
Plot3(card.s.95.5$fitCosineoG,  Pch="5")

Plot1(tri.s.05.10$fitCosineoG, Main="Triangular", Pch="1")

Plot2(tri.s.25.10$fitCosineoG, Pch="2", 0.5, 0.01)
Plot2(tri.s.50.10$fitCosineoG, Pch="3")
Plot2(tri.s.75.10$fitCosineoG, Pch="4")
```

```
Plot2(tri.s.95.10$fitCosineoG, Pch="5")
Plot3(tri.s.05.5$fitCosineoG,  Pch="1")
Plot3(tri.s.25.5$fitCosineoG,  Pch="2", 0.55, 0.01)
Plot3(tri.s.50.5$fitCosineoG,  Pch="3")
Plot3(tri.s.75.5$fitCosineoG,  Pch="4")
Plot3(tri.s.95.5$fitCosineoG,  Pch="5")

Plot1(vM.s.05.10$fitCosineoG, Main="von Mises", Pch="1")

Plot2(vM.s.25.10$fitCosineoG, Pch="2")
Plot2(vM.s.50.10$fitCosineoG, Pch="3")
Plot2(vM.s.75.10$fitCosineoG, Pch="4")
Plot2(vM.s.95.10$fitCosineoG, Pch="5")

Plot3(vM.s.05.5$fitCosineoG,  Pch="1")
Plot3(vM.s.25.5$fitCosineoG,  Pch="2")
Plot3(vM.s.50.5$fitCosineoG,  Pch="3")
Plot3(vM.s.75.5$fitCosineoG,  Pch="4")
Plot3(vM.s.95.5$fitCosineoG,  Pch="5")

Plot1(WrC.s.05.10$fitCosineoG, Main="Wrapped Cauchy", Pch="1")

Plot2(WrC.s.25.10$fitCosineoG, Pch="2")
Plot2(WrC.s.50.10$fitCosineoG, Pch="3")
Plot2(WrC.s.75.10$fitCosineoG, Pch="4")
Plot2(WrC.s.95.10$fitCosineoG, Pch="5")

Plot3(WrC.s.05.5$fitCosineoG,  Pch="1")
Plot3(WrC.s.25.5$fitCosineoG,  Pch="2")
Plot3(WrC.s.50.5$fitCosineoG,  Pch="3")
Plot3(WrC.s.75.5$fitCosineoG,  Pch="4")
Plot3(WrC.s.95.5$fitCosineoG,  Pch="5")

Plot1(U.s.10$fitCosineoG, Main="Uniform", Pch="1")
Plot3(U.s.5$fitCosineoG,  Pch="1")
```

L.13 Plot Figures M-6 to M-10

```
# Figure M-6

range=5
x <- seq(0,range,length=101)/range
a1=0.5; a2=0.8; a3=2

PlotCosModels <- function(a)
{
	y <- 2^(1-a) * (gamma(a))^(-1) * x^a * besselK(x, a)
	plot(x, y, ty="l" , col=1, xlab=paste("a=",a), ylim=c(0,1))
}

par(mfrow=c(3,1), mai=c(0.625, 0.50, 0.15, 0.25))
PlotCosModels(a=a1); PlotCosModels(a=a2); PlotCosModels(a=a3)


# Figure M-7

range=5
x <- seq(0,range,length=101)/range
a1=0.1; a2=1; a3=2
b1=0.1; b2=1.5; b3=3
c1=2; c2=4; c3=6

PlotCosModels <- function(a,b)
{
	y <- (1+(1-b/c1)*x^a)*(1+x^a)^(-1-b/a)
	plot(x, y, ty="l" , col=1, xlab=paste("a=",a), ylab=paste("b=", b))
	y <- (1+(1-b/c2)*x^a)*(1+x^a)^(-1-b/a)
	lines(x,y, col="tan")
	y <- (1+(1-b/c3)*x^a)*(1+x^a)^(-1-b/a)
	lines(x,y, col=1, lwd=2, lty=2)
}
```

```
par(mfrow=c(3,3))
PlotCosModels(a=a1,b=b3); PlotCosModels(a=a2,b=b3); PlotCosModels(a=a3,b=b3)
PlotCosModels(a=a1,b=b2); PlotCosModels(a=a2,b=b2); PlotCosModels(a=a3,b=b2)
PlotCosModels(a=a1,b=b1); PlotCosModels(a=a2,b=b1); PlotCosModels(a=a3,b=b1)


# Figure M-8

range=5
x <- seq(0,range,length=101)/range
a1=0.1; a2=1; a3=2
b1=0.1; b2=1.5; b3=3

PlotCosModels <- function(a,b)
{
	y <- (1+x^a)^(-b/a)
	plot(x, y, ty="l" , col=1, xlab=paste("a=",a), ylab=paste("b=", b))
}

par(mfrow=c(3,3))
PlotCosModels(a=a1,b=b3); PlotCosModels(a=a2,b=b3); PlotCosModels(a=a3,b=b3)
PlotCosModels(a=a1,b=b2); PlotCosModels(a=a2,b=b2); PlotCosModels(a=a3,b=b2)
PlotCosModels(a=a1,b=b1); PlotCosModels(a=a2,b=b1); PlotCosModels(a=a3,b=b1)
```

```
# Figure M-9

range=5; x <- seq(0,range,length=101)/range
a1=1; a2=5; a3=10
b1=1; b2=5; b3=10
c1=1; c2=2; c3=3

PlotCosModels <- function(a,b)
{
	y <- c1^(-b) * (besselK(a*c1, b))^(-1) * (c1^2 + x^2)^(0.5*b) * besselK(a*sqrt(c1^2 + x^2), b)
	plot(x, y, ty="l" , col=1, xlab=paste("a=",a), ylab=paste("b=", b), ylim=c(0,1))
	y <- c2^(-b) * (besselK(a*c2, b))^(-1) * (c2^2 + x^2)^(0.5*b) * besselK(a*sqrt(c2^2 + x^2), b)
	lines(x,y, col="tan")
	y <- c3^(-b) * (besselK(a*c3, b))^(-1) * (c3^2 + x^2)^(0.5*b) * besselK(a*sqrt(c3^2 + x^2), b)
	lines(x,y, col=1, lwd=2, lty=2)
}

par(mfrow=c(3,3))
PlotCosModels(a=a1,b=b3); PlotCosModels(a=a2,b=b3); PlotCosModels(a=a3,b=b3)
PlotCosModels(a=a1,b=b2); PlotCosModels(a=a2,b=b2); PlotCosModels(a=a3,b=b2)
PlotCosModels(a=a1,b=b1); PlotCosModels(a=a2,b=b1); PlotCosModels(a=a3,b=b1)


# Figure M-10

range=5; x <- seq(0,range,length=101)/range
a1=0.5; a2=1; a3=2

PlotCosModels <- function(a)
{
	y <- exp(-x^a)
	plot(x, y, ty="l" , col=1, xlab=paste("a=",a))
}

par(mfrow=c(3,1), mai=c(0.625, 0.50, 0.15, 0.25))
PlotCosModels(a=a1); PlotCosModels(a=a2); PlotCosModels(a=a3)
```

L.14 Plot Figures 4-3 and 4-4

```
# Figure 4-3, Cosine Model Behavior around the Observation Location 0

sample4.x <- c(-20, -10, 0, 10, 20); sample4.y <- c(0,0,0,0,0); sample4.direction <- c(pi/2, 0, pi/2, 0, pi/2)
x4 <- seq(-20, 20, length.out=201); y4 <- rep(0, 201)

krig4.e <- KrigCRF(krig.x=x4, krig.y=y4, resid.x=sample4.x, resid.y=sample4.y, resid.direction= sample4.direction, Model="exponential",
    Nugget=0.0, Range=10.0, sill=0, Plot=FALSE)
krig4.g <- KrigCRF(krig.x=x4, krig.y=y4, resid.x=sample4.x, resid.y=sample4.y, resid.direction= sample4.direction, Model="gauss",
    Nugget=0.0, Range=10.0, sill=0, Plot=FALSE)
krig4.s <- KrigCRF(krig.x=x4, krig.y=y4, resid.x=sample4.x, resid.y=sample4.y, resid.direction= sample4.direction, Model="spherical",
    Nugget=0.0, Range=10.0, sill=0, Plot=FALSE)

par(mai=c(0.65, 0.6, 0.05, 0.05), lab=c(5,10,7), mgp=c(2,1,0))
plot( krig4.g$x, krig4.g$direction*180/pi, ty="l", col="tan", lwd=4, xlim=c(-10,1), ylim=c(0, 100),
    xlab="Location", ylab="Kriging Estimate in Degrees")
lines(krig4.s$x, krig4.s$direction*180/pi, col=2, lty=2)
lines( krig4.e$x, krig4.e$direction*180/pi, col=1)

krig5.e <- KrigCRF(krig.x=x4, krig.y=y4, resid.x=sample4.x, resid.y=sample4.y, resid.direction= sample4.direction, Model="exponential",
    Nugget=0.1, Range=10.0, sill=0, Plot=FALSE)
krig5.g <- KrigCRF(krig.x=x4, krig.y=y4, resid.x=sample4.x, resid.y=sample4.y, resid.direction= sample4.direction, Model="gauss",
    Nugget=0.1, Range=10.0, sill=0, Plot=FALSE)
krig5.s <- KrigCRF(krig.x=x4, krig.y=y4, resid.x=sample4.x, resid.y=sample4.y, resid.direction= sample4.direction, Model="spherical",
    Nugget=0.1, Range=10.0, sill=0, Plot=FALSE)

par(mai=c(0.65, 0.05, 0.05, 0.05),yaxt="n")
plot( krig5.g$x, krig5.g$direction*180/pi, ty="l", col="tan", lwd=4, xlim=c(-10,1), ylim=c(0, 100), xlab="Location", ylab="")
lines(krig5.s$x, krig5.s$direction*180/pi, col=2, lty=2)
lines( krig5.e$x, krig5.e$direction*180/pi, col=1)
```

```
# Figure 4-4

PlotCKVar(rho=0,   ng=0,   range=4, x.legend=4.0, y.legend=1.5)
PlotCKVar(rho=0.5, ng=0,   range=4, x.legend=3.5, y.legend=1.0)
PlotCKVar(rho=0,   ng=0.2, range=4, x.legend=3.5, y.legend=1.25)
PlotCKVar(rho=0,   ng=0,   range=8, x.legend=4.5, y.legend=1.0)
```