Análisis de presencias con procesos de puntos

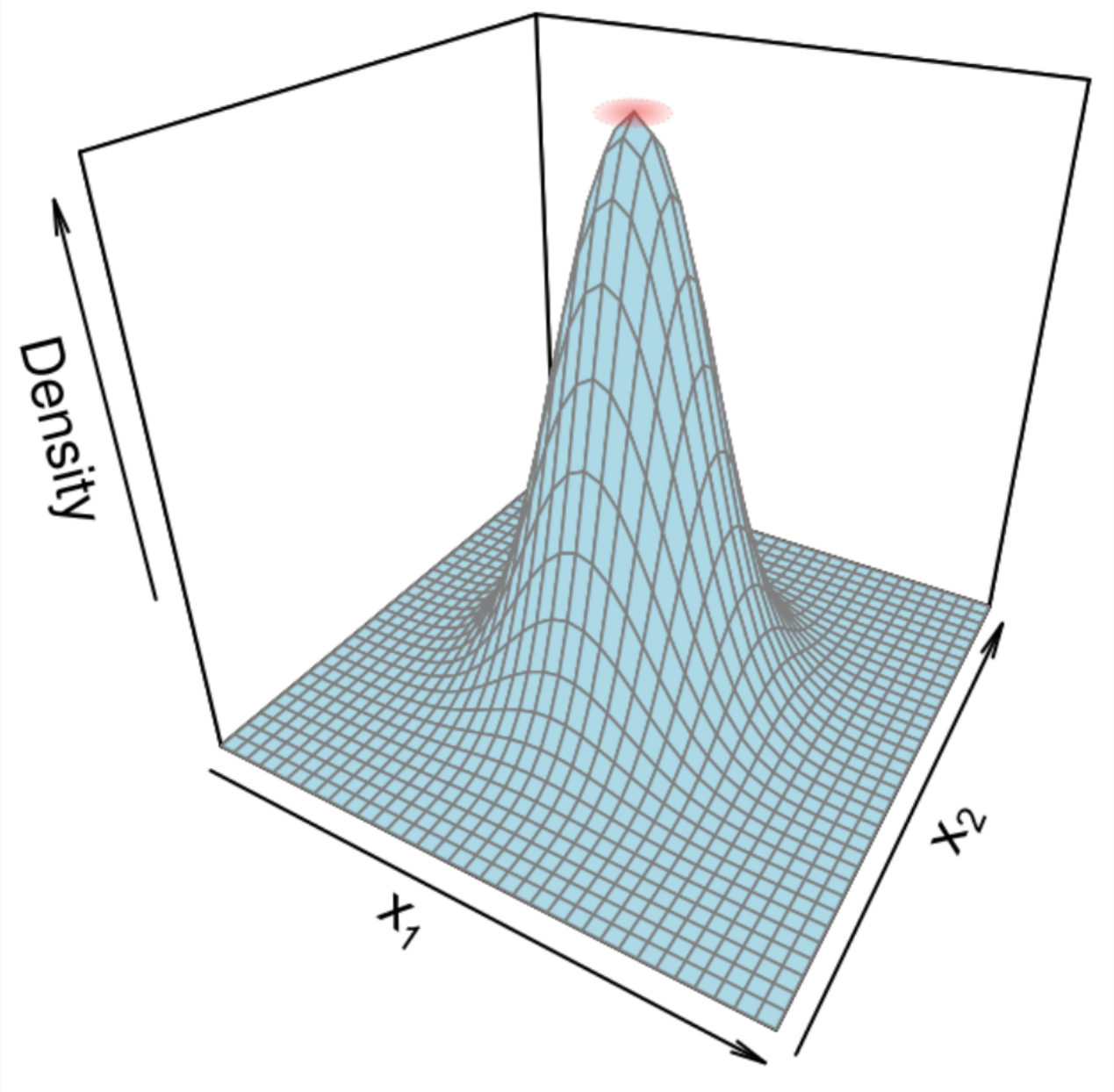
Tutorial intermedio de spatstat

Gerardo Martín

2022-06-29

Simulación de presencias

Especificación de un centroide

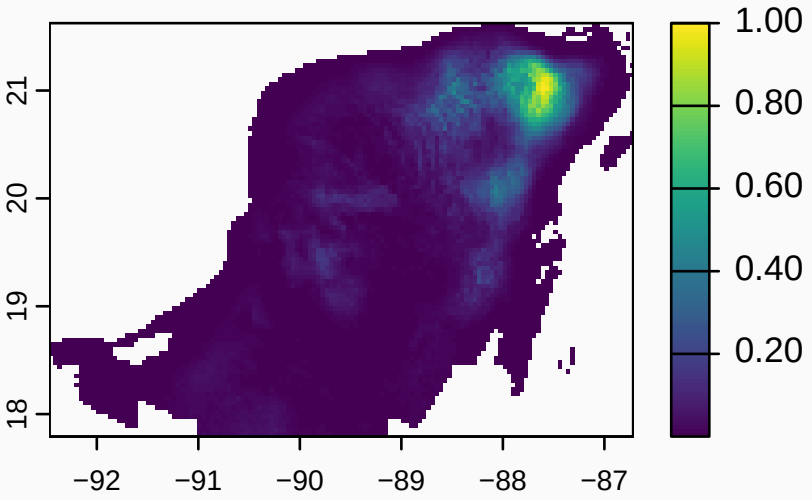


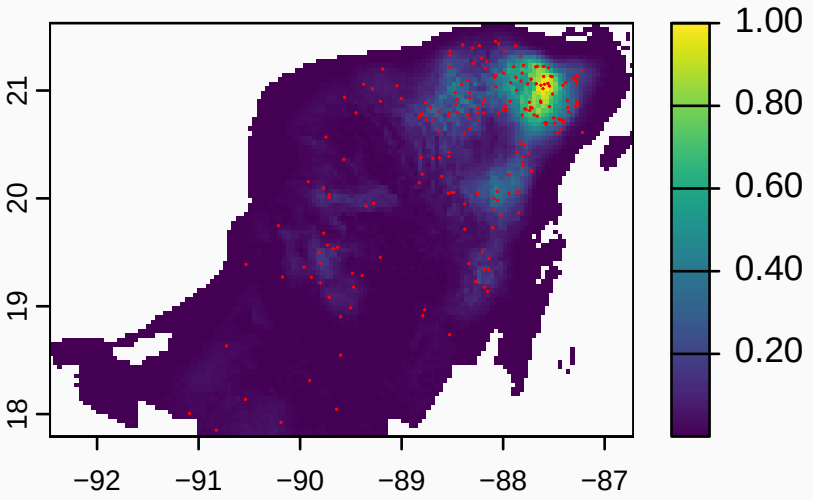
Análisis 2

Importando y escalando datos y covariables

- `aggregate` disminuye la resolución por el factor indicado
- `round` redondea los valores con el número de decimales
- Estos pasos no son enteramente necesarios en un análisis real, los hacemos para disminuir tiempo de cómputo

Código - viendo la favorabilidad





Formateo para spatstat

Cargando las funciones

```
source("Funciones-spatstat/imFromStack.R")
source("Funciones-spatstat/plotQuantIntens.R")
source("Funciones-spatstat/findCompatibles.R")
source("Funciones-spatstat/getPolyFormulas.R")
source("Funciones-spatstat/ppmBatchFit.R")
```

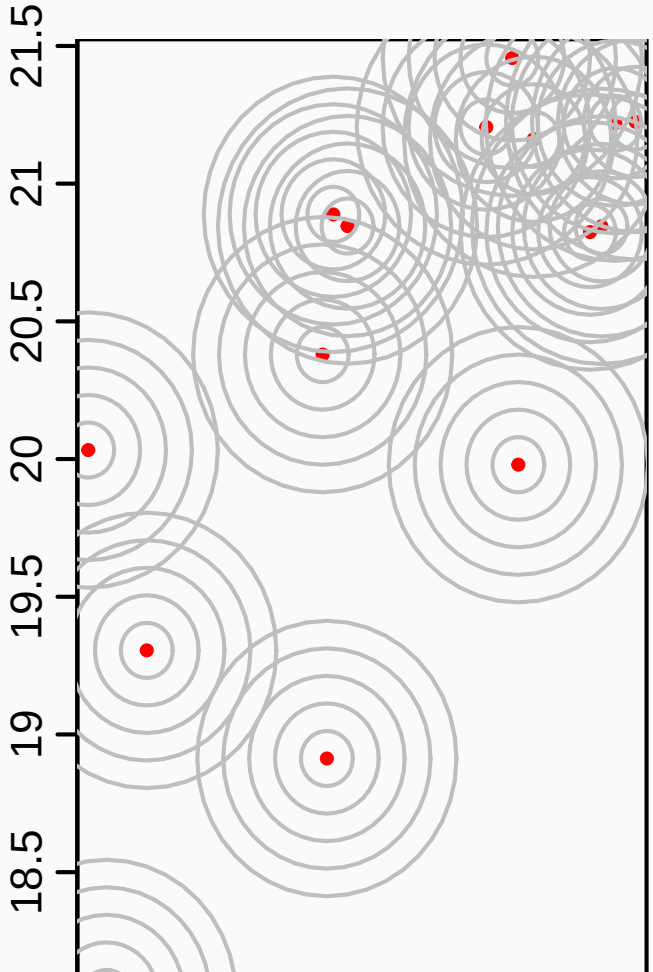
Formateo rápido

```
r.im <- imFromStack(r)
names(r.im) <- names(r)
w <- as.owin(r.im[[1]])
puntos.ppp <- ppp(x = puntos$x,
                  y = puntos$y,
                  window = w,
                  check = F)
Q <- pixelquad(X = puntos.ppp, W = as.owin(w))
```

Análisis exploratorio

Función de *K* de Ripley

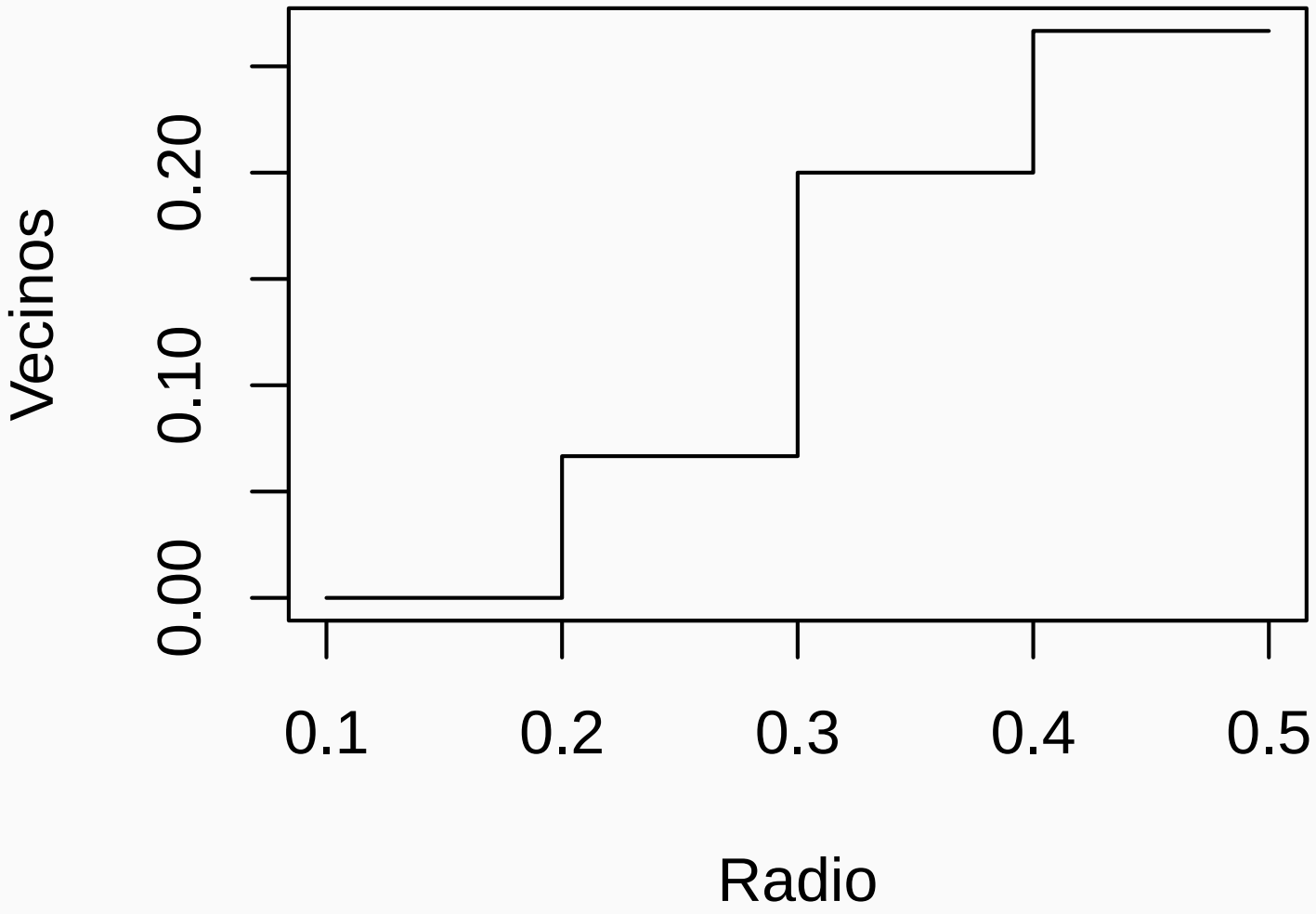
1. Número promedio de vecinos como función de la distancia a cada punto:



Los datos

Radio	Vecinos
0.1	0.0000000
0.2	0.0666667
0.3	0.2000000
0.4	0.2666667
0.5	0.2666667

Representación gráfica

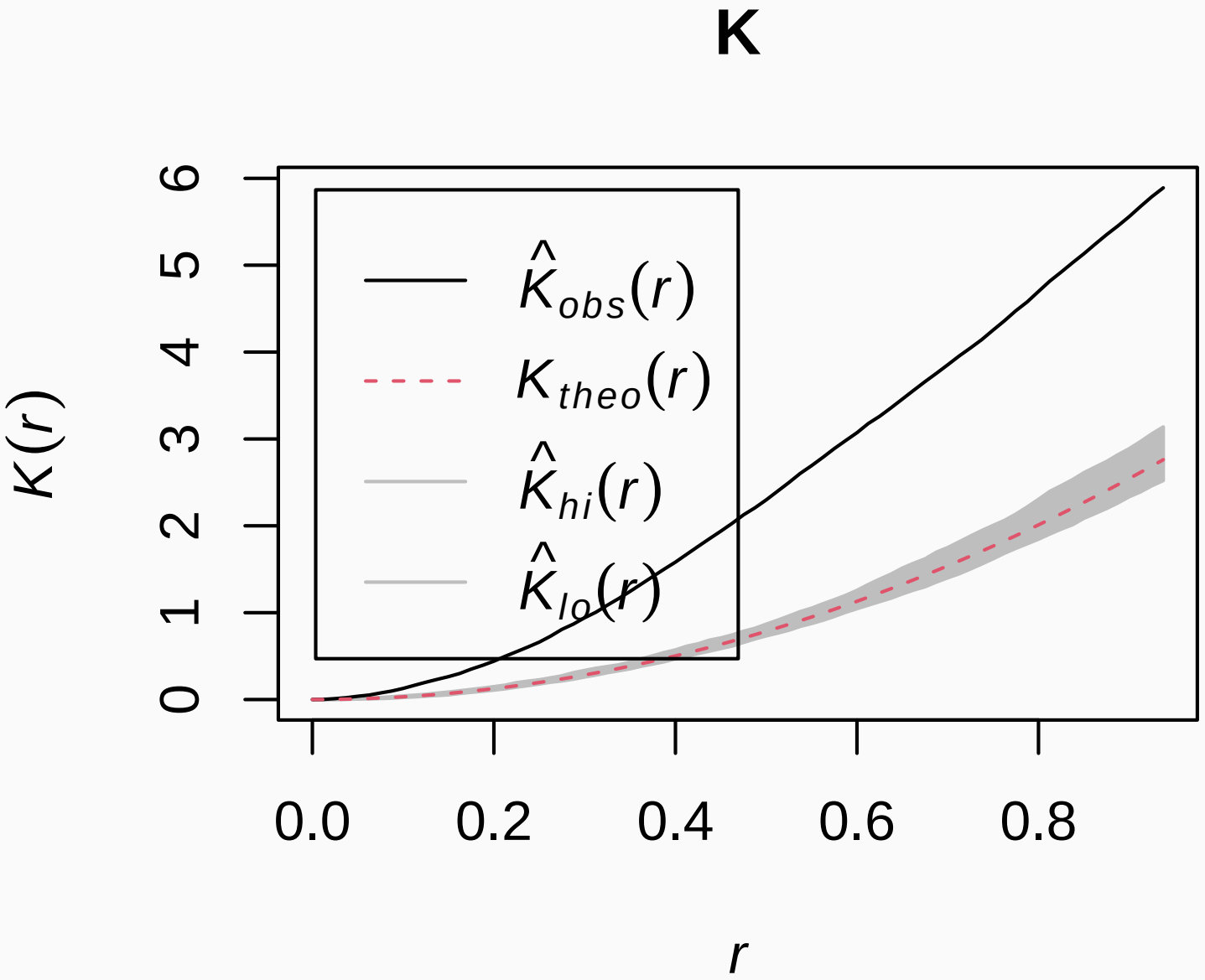


Implementación en spatstat

```
K <- envelope(puntos.ppp, fun = Kest, nsim = 39)
```

Para estimar significancia hace un muestreo aleatorios de punto, de ahí que haya que especificar el número de simulaciones (`nsim = 39`).

Interpretación gráfica



Autocorrelación - notas

1. El proceso está levemente autocorrelacionado
    - Veremos si la correlación presente es explicada por factores ambientales
2. No sabemos de momento si afectará al modelo

Análisis 2.0

Respuestas a variables

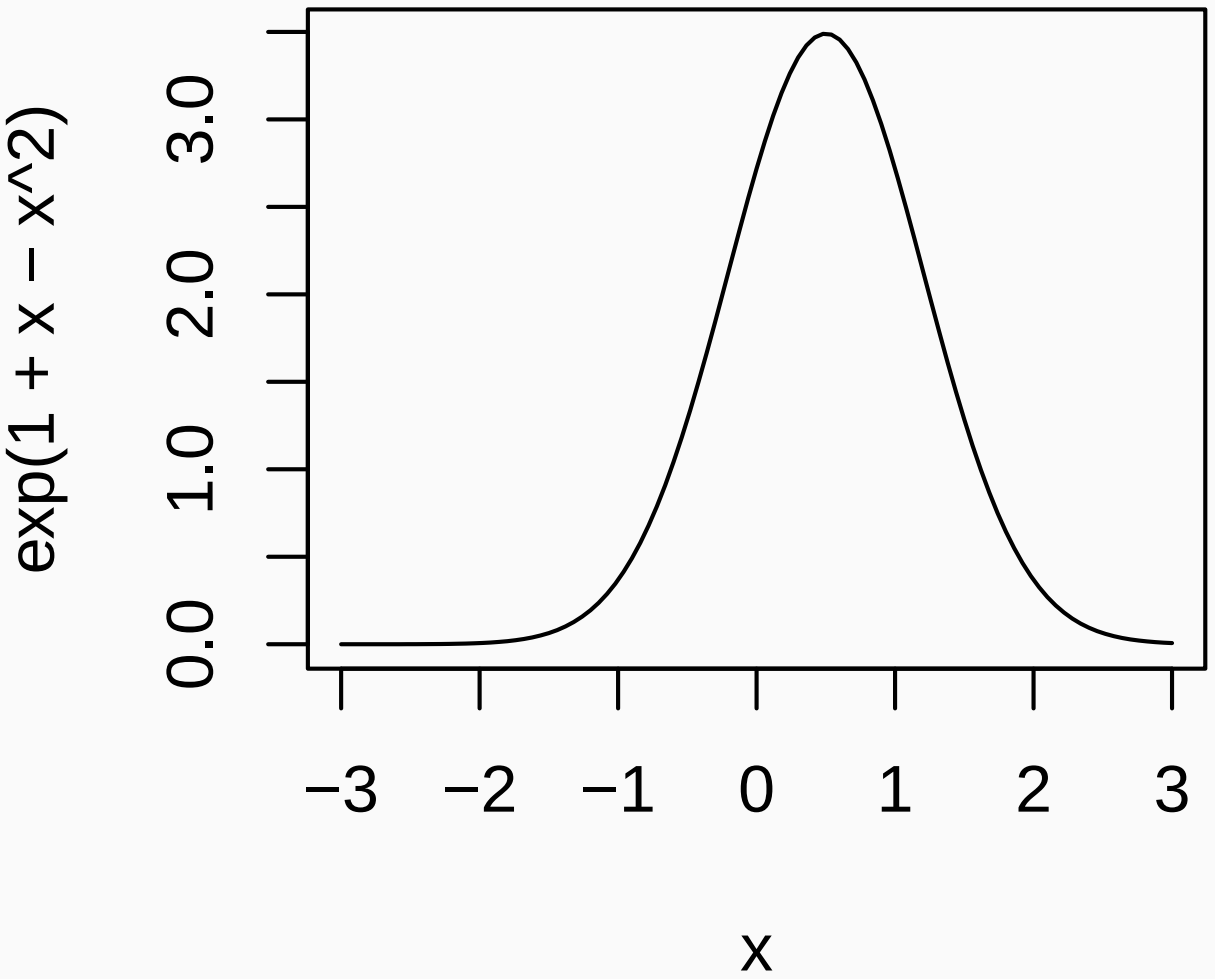
```
plotQuantIntens(imList = r.im,
                noCuts = 5,
                Quad = Q,
                p.pp = puntos.ppp,
                dir = "",
                name = "Respuestas-centroide")
```

Ver archivo de gráficas

Consideraciones para proponer modelos

Curvas con forma de campana $\longrightarrow$ fórmula cuadrática

```
curve(exp(1 + x - x^2), from = -3, 3)
```



Consideraciones para proponer modelos

Ecuación lineal:

$$y = \alpha + \beta_1 x_1 + \cdots + \beta_n x_n$$

Ecuación polinomial de 2^o grado

$$y = \alpha + \beta_1 x_1 + \beta_1' x_1^2 + \cdots + \beta_n x_n + \beta_n' x_n^2$$

Recordemos que $y = \log \lambda$

¿Qué variables podemos incluir en el mismo modelo?

Regla de oro: Aquellas que no estén correlacionadas

- Que x_1 no sea predictor de x_2
- No se puede atribuir efecto de x_1 ó x_2 sobre λ
- Necesitamos medir correlación entre pares de variables (`pairs`)

Identificación automática de covariables *compatibles*

```
compatibles <- findCompatibles(r, thres = 0.5,
                               max.comb = 3)
```

Variable_1	Variable_2	Variable_3
bio1	bio12	bio18
bio1	bio12	bio2
bio1	bio12	bio3
bio1	bio12	bio4
bio1	bio12	bio6
bio1	bio12	bio7

Obteniendo las fórmulas

- Necesitamos generar una tabla de exponentes para variables, usando el resultado de `plotQuantIntens`.
- Razonamiento:
    - Identificar exponente máximo que tendrá el modelo para cada variable
    - La función `getPolyFormulas` generará las fórmulas para todas las combinaciones con exponentes $1:n$
- Tabla debe tener dos columnas: `Variable`, `Power`

Uso de getPolyFormulas

```
expon <- read.csv("Datos/Tabla-coefs.csv")
formulas  <- getPolyFormulas(respDF = expon,
                             compatMat = compatibles)
formulas[1:5]
```

```
## [1] "~bio1 + bio12 + I(bio12^2) + bio18 + I(bio18^2) + I(bio18
## [2] "~bio1 + bio12 + I(bio12^2) + bio2 + I(bio2^2)"
## [3] "~bio1 + bio12 + I(bio12^2) + bio3 + I(bio3^2)"
## [4] "~bio1 + bio12 + I(bio12^2) + bio4 + I(bio4^2)"
## [5] "~bio1 + bio12 + I(bio12^2) + bio6 + I(bio6^2)"
```

Ajustando los modelos

- La función `ppmBatchFit` ajustará todos los modelos generados por `getPolyFormulas`
- Algunos modelos no lograrán estimar coeficientes satisfactoriamente
- La implementación presente solamente puede priorizar con base en AIC
- En un futuro, eliminará modelos que con converjan

Uso de ppmBatcchFit

```
modelos <- ppmBatchFit(points = puntos,
                       covariates = r,
                       formulas = formulas[1:10],
                       parallel = F,
                       topModels = 5)
```

Los argumentos

- `points`, tabla de coordenadas con dos columnas, `x` y `y`, en formato `data.frame`
- `covariates`, raster con bandas como covariables, nombres deben coincidir con fórmulas
- `formulas`
- `parallel`, si la rutina se ejeccutará en serie ó paralelo, si `parallel = T`, especificar número de núcleos a usar con `cores = 3` (ajustar para cada máquina)
- `topModels`, cuántos de los "mejores" modelos queremos que nos guarde
- El resultado almacenado en `modelos` es una lista con los 5 mejores con base en el AIC

Analizando el resultado

```
sapply(modelos, AIC)
```

```
## [1]  -936.8556  -953.2552  -945.1438 -1031.7370  -
956.3632
```

Podemos usar los procedimientos habituales para los modelos de regresión en R

```
summary(modelos[[1]])
```

Análisis de residuales

Como en los análisis de regresión, podemos ver el ajuste con los residuales, y siendo un modelo espacial, ver si hemos logrado explicar la correlación espacial con la prueba K de Ripley, tal como en el análisis exploratorio:

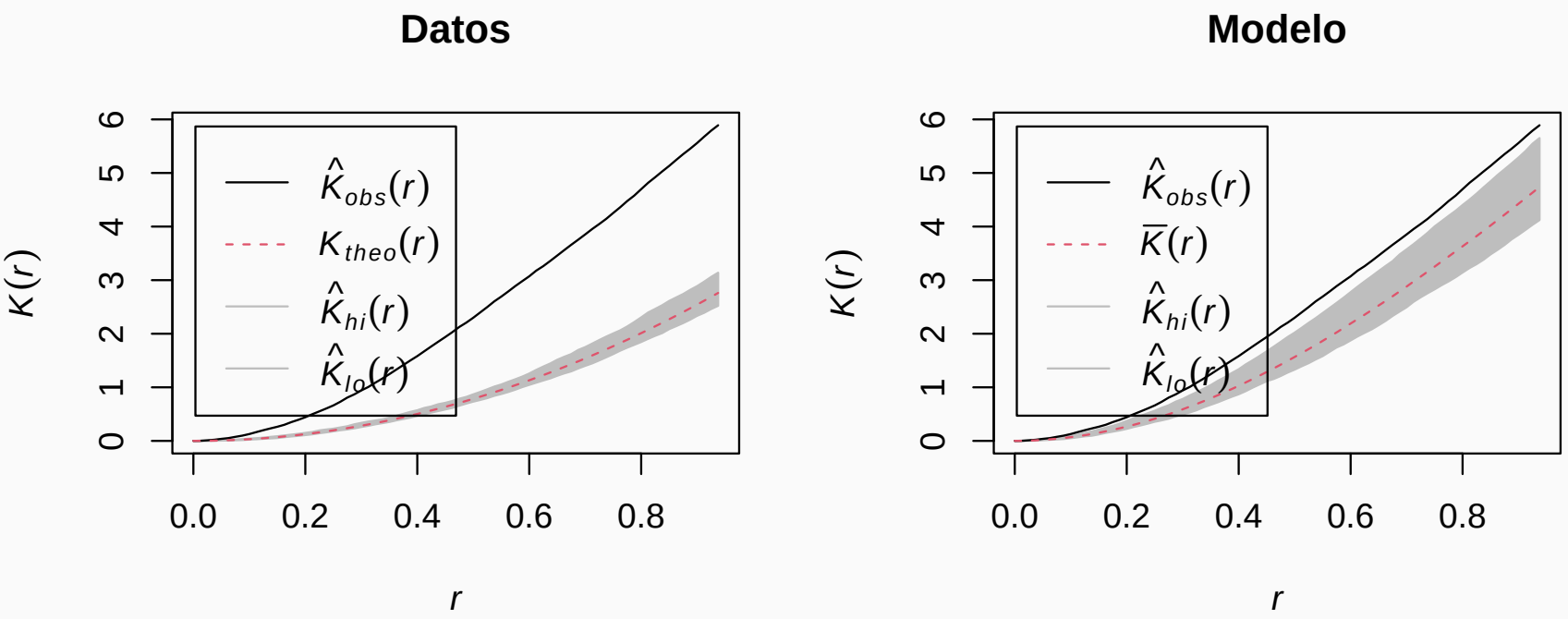
```
K.modelo <- envelope(modelos[[1]], fun = "Kest", nsim = 39)
```

```
## Generating 39 simulated realisations of fitted Poisson model
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1
## 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
## 39.
##
## Done.
```

Esta prueba genera 39 patrones de puntos utilizando el modelo base para calcular la función de Ripley y compara las simulaciones con la base de datos

Gráfica

```
par(mfrow = c(1, 2))
plot(K, main = "Datos")
plot(K.modelo, main = "Modelo")
```



Métodos para residuales

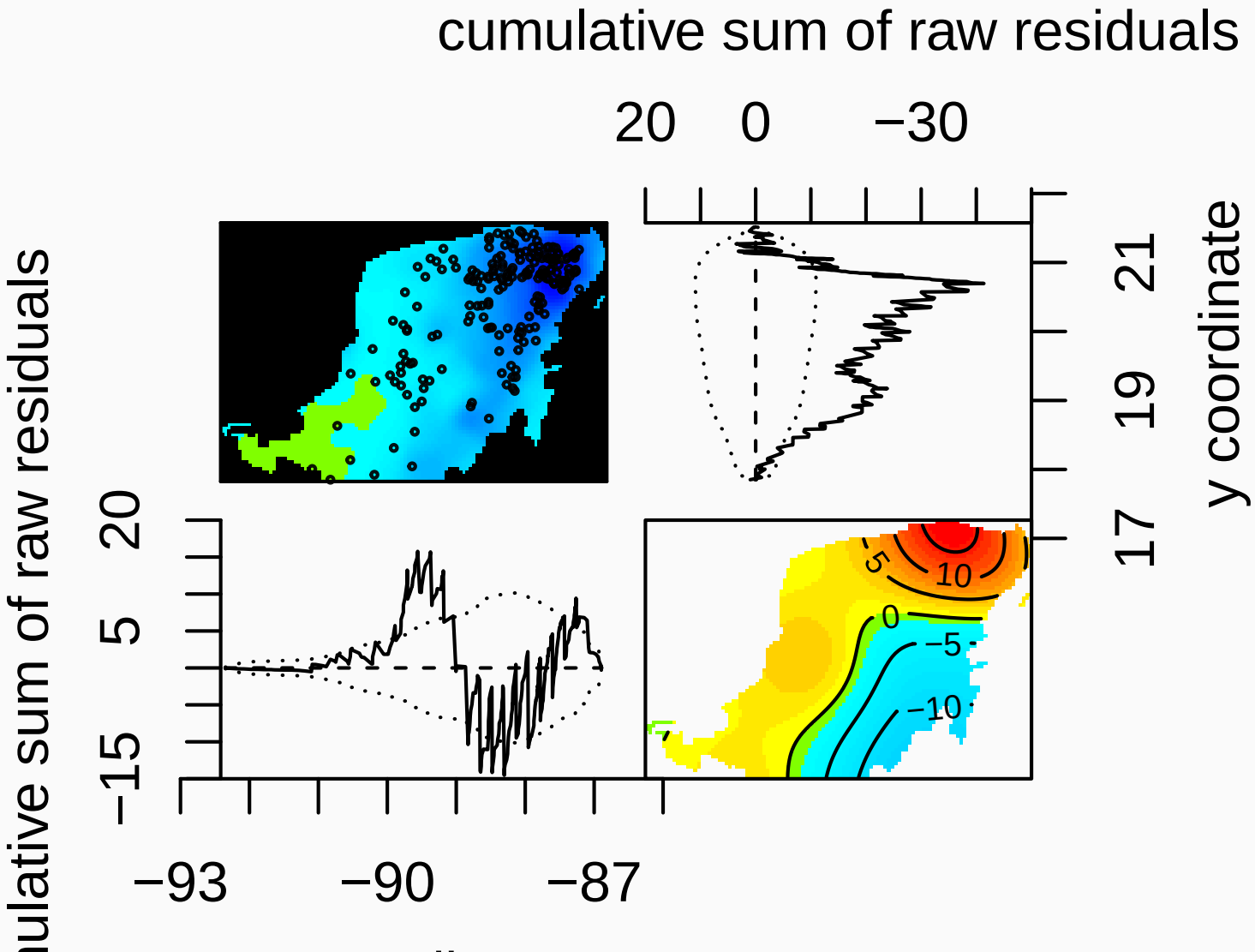
- Gráficas de horizonte
- Muestra 4 páneles:

1. Patrón de puntos
2. Residuales acumulados en cada fila de píxeles
3. Residuales acumulados en cada columna de píxeles
4. Residuales suavizados con contornos:

$$Kernel - Modelo$$

Gráfica de horizonte

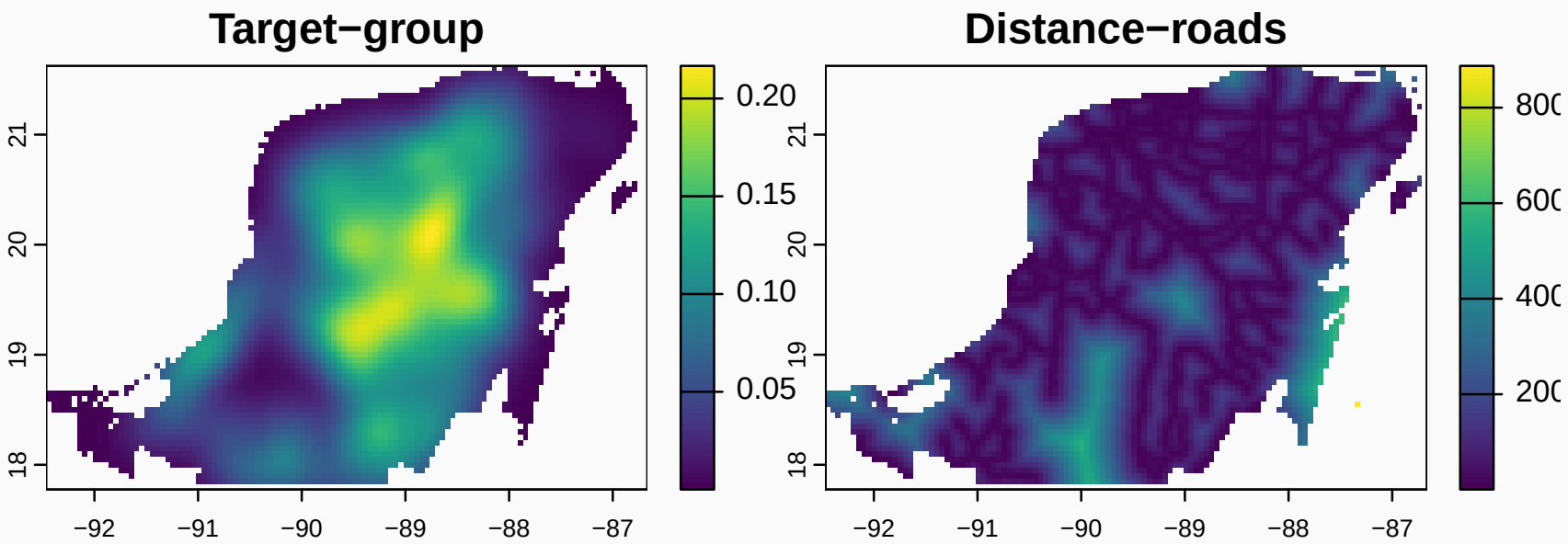
```
par(mar = c(1.5,1,0,0))
diagnose.ppm(modelos[[1]], cex = 0.25, outer = 5)
```



Corrección de sesgo

Definición de escenario de sesgo

```
sesgo <- rast(c("Datos/Target-group.tif",
                "Datos/Distance-roads.tif"))
sesgo <- resample(sesgo, r)
plot(sesgo)
```



Filtrado del entorno

- Usaremos las función `maskBias`

```
source("Funciones-spatstat/maskBias.R")
```

- Necesita los siguientes argumentos:

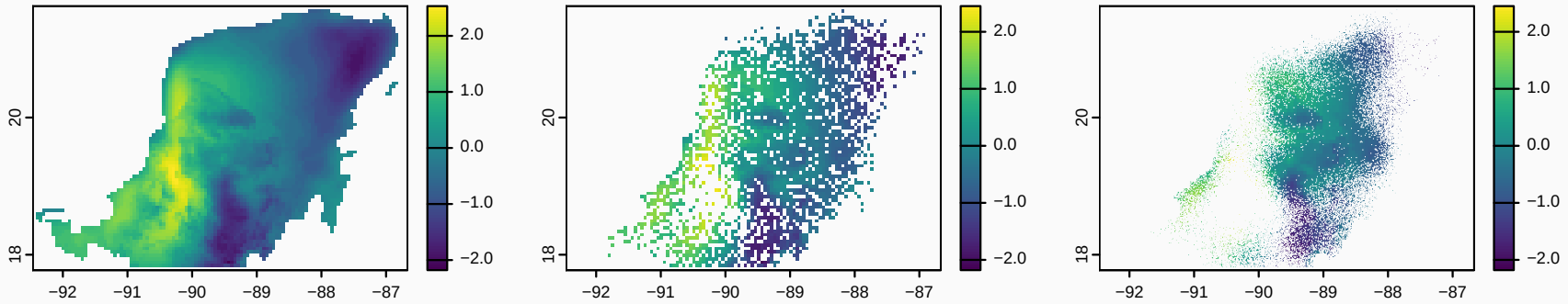
1. `s`, es el raster multi-banda de las covariables
2. `pres.areas`, es la base de datos de presencia con las coordenadas `x` y `y`
3. `bias.lay`, la capa que representa el esfuerzo de muestreo, donde los valores máximos correspondan a aquellos más muestreados.
4. `p.keep`, la proporción de valores a ser retenidos en cada corte de la capa
5. `power`, cuánto queremos que las muestras se concentren en los valores más altos de la capa de sesgo
6. `dis`, un factor de desagregación en caso que querer concentrar más valores en las zonas más muestreadas de lo que la resolución

Ejemplos

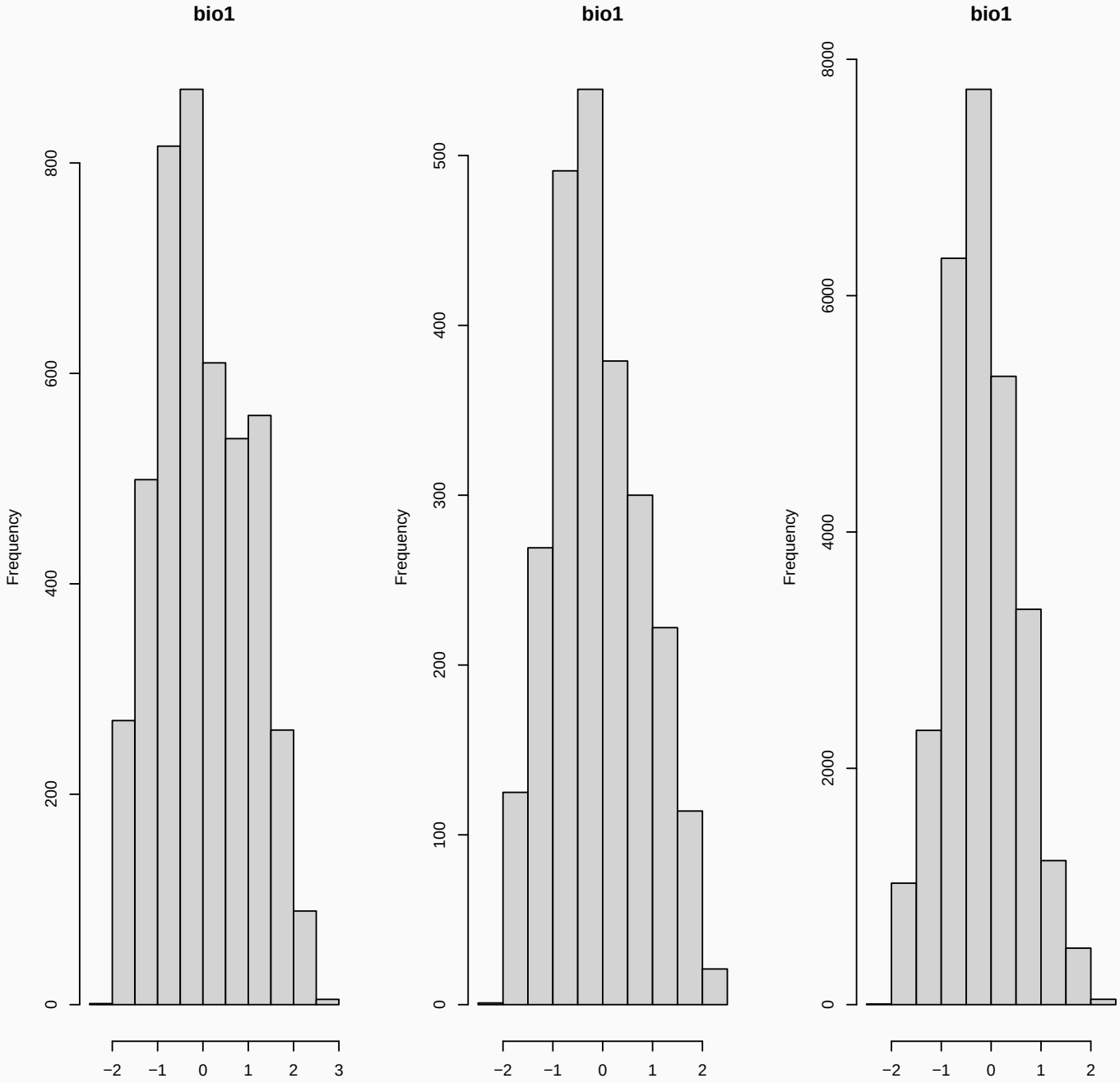
```
r.mask1 <- maskBias(s = r[[1]], #Bio1
                    pres.areas = puntos,
                    bias.lay = sesgo[[1]], #Target group
                    p.keep = 0.1, power = 1)

r.mask2 <- maskBias(s = r[[1]], #Bio1
                    pres.areas = puntos,
                    bias.lay = sesgo[[1]], #Target group
                    p.keep = 0.1, power = 3, dis = 4)
```

Gráfica



Histogramas



El resto de la historia

1. Correr análisis exploratorio con capas filtradas
2. Seleccionar exponentes
3. Generar fórmulas
4. Ajustar modelos
5. Evaluar bondad de ajuste
6. Proyectar a geografía completa (sin filtrado)
7. Validar