oSCR - an R package for SCR analyses

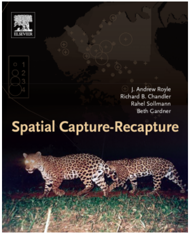
Chris Sutherland
University of Massachusetts – Amherst
csutherland@umass.edu

Analyzing SPARCnet data

What is oSCR?

oSCR → opensource SCR

- built upon various functions in `scrbook`
- an editable and evolving code base
- focus on increased transparency and accessibility
- a community of contributors
  - *contributing* ideas
  - *contributing* problems
  - *contributing* solutions
  - *contributing* code



Where is oSCR?

oSCR - getting the package

- hosted on github
- download directly from github for most up-to-date version

```
install.packages("githubinstall")
install.packages("devtools")
library(githubinstall)
library(devtools)
#install oSCR directly from github
install_github("jaroyle/oSCR")
#load oSCR
library(oSCR)
```

SCR analysis with oSCR - the workflow

Every analysis in oSCR involves the following steps:

1. Format the sampling data
   - spatial encounter history data
   - trapping information data
   - etc. . .

SCR analysis with oSCR - the workflow

Every analysis in oSCR involves the following steps:

1. Format the sampling data
2. Format the spatial extent data
   - extent of the *state space*
   - spatial covariates for density
   - state space constraints

SCR analysis with oSCR - the workflow

Every analysis in oSCR involves the following steps:

1. Format the sampling data
2. Format the spatial extent data
3. Analyze the data data - model fitting

SCR analysis with oSCR - the workflow

Every analysis in oSCR involves the following steps:

1. Format the sampling data
2. Format the spatial extent data
3. Analyze the data data - model fitting
4. Post processing model output to do inference

SCR analysis with oSCR - the workflow

Every analysis in oSCR involves the following steps:

1. Format the sampling data
2. Format the spatial extent data
3. Analyze the data data - model fitting
4. Post processing model output to do inference

Each step has helpful oSCR helper functions

- repeatable, transparent & sharable workflow

Reminder: Spatially explicit observation model

$$y_{ijk} | s_i \sim \text{Bernoulli}(p[x_j, s_i])$$

$$p[x_j, s_i] = p_0 \times \exp\left(-\frac{\text{dist}(x_j, s_i)^2}{2\sigma^2}\right)$$

- data requirements:
  - y_{ijk} observations (binary or frequencies)
  - x_j trap locations
- in `oSCR` the *data* are organized in an `scrFrame`

Making the `scrFrame` from SPARCnet-style data

Data are naturally entered in two basic spreadsheets:

1. the encounter data file `edf` (a single data frame):

- a row per detection
- several key columns with capture information
  - session ID (required)
  - individual ID (required)
  - occasion ID (required)
  - trap ID (required)
  - sex (optional)

Making the `scrFrame` from traditional data

Data are naturally entered in two basic spreadsheets:

1. the encounter data file `edf` (a single data frame):

- a row per detection
- several key columns with capture information

```
# the rbs edf:
rbs.edf <- read.csv("rbsNY.csv",h=T)
head(rbs.edf)
  Site     Ind Occasion Board Sex
1  P1A BxxYP1A        1    A2   M
2  P1A YxYBP1A        1    A3   M
3  P1A xxBYP1A        1    A6   M
4  P1A YYxBP1A        1    A8   M
5  P1A xBBYP1A        1   B10   U
6  P1A xBxxP1A        1    B5   U
```

Making the `scrFrame` from traditional data

Data are naturally entered in two basic spreadsheets:

2. the trap deployment data file `tdf` (a list):

- a list containing a data frame for each session
- a row per trap
- several key columns with trap information
    - detector name (required)
    - `X` coordinates (required)
    - `Y` coordinates (required)
    - other data typically stored but not required

Making the `scrFrame` from traditional data

Data are naturally entered in two basic spreadsheets:

2. the trap deployment data file `tdf`:

- a list containing a data frame for each session
- a row per trap
- several key columns with trap information

```
# the rbs edf:
tdf1 <- read.csv("tdfNY1.csv",h=T)
tdf2 <- read.csv("tdfNY2.csv",h=T)
tdf3 <- read.csv("tdfNY3.csv",h=T)
tdf4 <- read.csv("tdfNY4.csv",h=T)
head(tdf1)
  board x y X1 X2 X3 X4 X5 X6 X7
1    A1 0 0  1  1  1  1  1  1  1
2    A2 0 1  1  1  1  1  1  1  1
3    A3 0 2  1  1  1  1  1  1  1
4    A4 0 3  1  1  1  1  1  1  1
5    A5 0 4  1  1  1  1  1  1  1
6    A6 0 5  1  1  1  1  1  1  1
```

Making the `scrFrame` from traditional data

Data are naturally entered in two basic spreadsheets:

2. the trap deployment data file `tdf`:

- a list containing a data frame for each session
- a row per trap
- several key columns with trap information

```
# the rbs edf:
tdf1 <- read.csv("tdfNY1.csv",h=T)
tdf2 <- read.csv("tdfNY2.csv",h=T)
tdf3 <- read.csv("tdfNY3.csv",h=T)
tdf4 <- read.csv("tdfNY4.csv",h=T)
head(tdf2)
  board x y X1 X2 X3 X4 X5
1    A1 0 0  1  1  1  1  1
2    A2 0 1  1  1  1  1  1
3    A3 0 2  1  1  1  1  1
4    A4 0 3  1  1  1  1  1
5    A5 0 4  1  1  1  1  1
6    A6 0 5  1  1  1  1  1
```

Making the `scrFrame` from traditional data

Data are naturally entered in two basic spreadsheets:

2. the trap deployment data file `tdf`:

- a list containing a data frame for each session
- a row per trap
- several key columns with trap information

```
# the rbs edf:
tdf1 <- read.csv("tdfNY1.csv",h=T)
tdf2 <- read.csv("tdfNY2.csv",h=T)
tdf3 <- read.csv("tdfNY3.csv",h=T)
tdf4 <- read.csv("tdfNY4.csv",h=T)
head(tdf3)
  board x y X1 X2 X3 X4 X5 X6
1    A1 0 0  1  1  1  1  1  1
2    A2 0 1  1  1  1  1  1  1
3    A3 0 2  1  1  1  1  1  1
4    A4 0 3  1  1  1  1  1  1
5    A5 0 4  1  1  1  1  1  1
6    A6 0 5  1  1  1  1  1  1
```

Making the `scrFrame` from traditional data

Data are naturally entered in two basic spreadsheets:

2. the trap deployment data file `tdf`:

- a list containing a data frame for each session
- a row per trap
- several key columns with trap information

```
# the rbs edf:
tdf1 <- read.csv("tdfNY1.csv",h=T)
tdf2 <- read.csv("tdfNY2.csv",h=T)
tdf3 <- read.csv("tdfNY3.csv",h=T)
tdf4 <- read.csv("tdfNY4.csv",h=T)
head(tdf4)
  board x y X1 X2 X3 X4
1    A1 0 0  1  1  1  1
2    A2 0 1  1  1  1  1
3    A3 0 2  1  1  1  1
4    A4 0 3  1  1  1  1
5    A5 0 4  1  1  1  1
6    A6 0 5  1  1  1  1
```

Making the `scrFrame` from traditional data

Data are naturally entered in two basic spreadsheets:

1. the encounter data file `edf` (a single data frame):
2. the trap deployment data file `tdf` (a list):

Some important points about this data:

- the detector names in `edf` and `tdf` MUST match
  - same names
  - same class (integer/character/factor)

Making the `scrFrame` using `data2oscr`

The `data2oscr()` function is a *very* useful helper function

- inputs are ‘traditional’ data formats
- returns several data objects
  - an `scrFrame`
  - data formatted for Bayesian analysis

Making the scrFrame using data2oscr

The data2oscr() function is a *very* useful helper function

```
# create general SCR data objects
data <- data2oscr(edf,                  # the edf
                  tdf,                  # the tdf
                  sess.col,             # session col NUMBER (edf)
                  id.col,               # ind ID col NUMBER (edf)
                  occ.col,              # occasion col NUMBER (edf)
                  trap.col,             # detector col NUMBER (edf)
                  sex.col,              # sex col NUMBER
                  sex.nacode,           # character for unknown sex?
                  K,                    # number of occassions
                  ntraps)               # the number of traps
# extract the scrFrame
sf <- data$scrFrame
```

Making the scrFrame using data2oscr

```
# use various objects made in R
rbs <- data2oscr(edf = rbs.edf,
                 tdf = list(tdf1,tdf2,tdf3,tdf4),
                 sess.col = 1,
                 id.col = 2,
                 occ.col = 3,
                 trap.col = 4,
                 K = c(7,5,6,4),
                 ntraps = c(50,50,50,50))
ls(rbs)
[1] "edf"      "scrFrame" "sex"      "trapcovs" "traplocs" "trapopp"  "y3d"
```

```
rbs.sf <- rbs$scrFrame
```

Inspecting the `scrFrame`

Summary functions for the capture data in the `scrFrame`

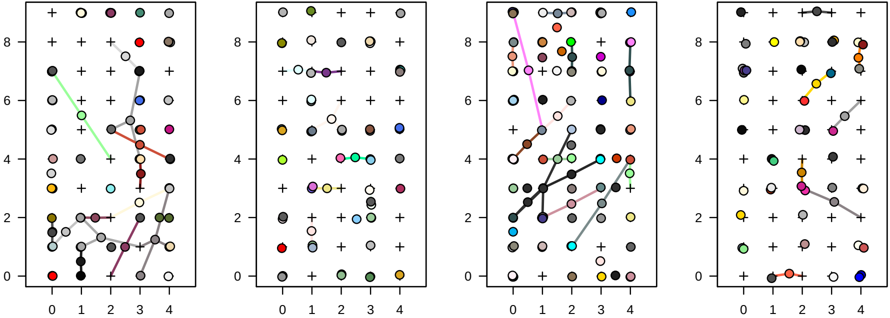
- type object name for a numerical summary: `sf`
  - number of individuals, captures & spatial recaptures
  - mean maximum distance moved (MMDM)
- plot spatial captures: `plot(sf)`
  - spatial average of locations
  - traps captured in

Inspecting the `scrFrame`

```
rbs.sf #print a summary

               S1 S2  S3 S4
n individuals 77 61 107 54
n traps       50 50  50 50
n occasions    7  5   6  4

                    S1   S2   S3   S4
avg caps          1.92 1.48 1.73 1.37
avg spatial caps 1.30 1.16 1.27 1.13
mmdm              2.10 1.05 1.68 1.29

Pooled MMDM:  1.66
```

Exploring the scrFrame

```
plot(rbs.sf) #plot a summary
```



Reminder: Spatially explicit density model

Spatially explicit density model (homogeneous):

- describes how activity centers are distributed in space

$$Pr(s_i) \propto exp(\beta)$$

$$s_i \in \mathcal{S}$$

- $\mathcal{S}$ is the *state space*
- $\mathcal{S}$ is a discretized representation of space
- s_i is a *pixel* centeroid & possible activity center location

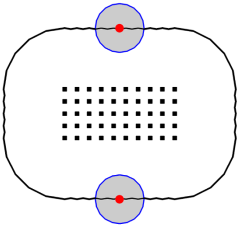
Defining the State space

State space definition is extremely important

Defining the State space

State space definition is extremely important

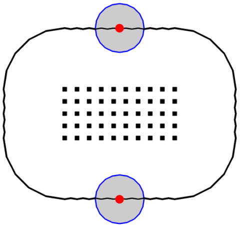
- $\mathcal{S}$ is part of the model!
  - defines where individuals can *live*
  - defines the population of interest
  - includes unsampled parts of the landscape



Defining the State space

State space definition is extremely important

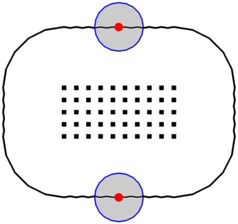
- $\mathcal{S}$ is part of the model!
  - defines where individuals can *live*
  - defines the population of interest
  - includes unsampled parts of the landscape
- should represent activity centers of all detectable individuals
  - a buffer of at least $2\hat{\sigma}$ around traps
  - ensures activity centers of detectable inds. are represented



Defining the State space

State space definition is extremely important

- $\mathcal{S}$ is part of the model!
  - defines where individuals can *live*
  - defines the population of interest
  - includes unsampled parts of the landscape
- should represent activity centers of all detectable individuals
  - a buffer of at least $2\hat{\sigma}$ around traps
  - ensures activity centers of detectable inds. are represented
- discrete approximation space
  - *pixel centroids* $\rightarrow$ activity centers
  - resolution should be $\leq \hat{\sigma}$



`ssDF` - the state space data object

`ssDF`: the state space data object (state space data frame):

- a list containing a data frame
- *at least* the coordinates of the discrete state space
  - must be named `X` and `Y` (upper case)
  - each coordinate represents a *pixel centroid*
- can add named columns of pixel-specific covariate values
  - used to model spatial variation in density
  - continuous or categorical
  - coordinates should be same units as `traps`
  - non-habitat can be removed!

ssDF - a state space data object

Create the ssDF object Using `make.ssDF()`:

```
# 1. use make.ssDF() to make an ssDF
ss <- make.ssDF(scrFrame,  # an scrFrame objects
                buffer,    # the buffer witdh (around traps!)
                res)       # the state space resolution
?make.ssDF()               # look at the help file
```

ssDF - a state space data object

Create the ssDF object Using `make.ssDF()`:

```
# 1. use make.ssDF() to make an ssDF
ss <- make.ssDF(scrFrame,  # an scrFrame objects
                buffer,    # the buffer witdh (around traps!)
                res)       # the state space resolution
?make.ssDF()               # look at the help file
```

A note about specifying values:

- `buffer` should be $\geq 2\hat{\sigma}$
- `res` should be $\leq \hat{\sigma}$
- use $\frac{1}{2}$ `mmdm` as approximation of $\hat{\sigma}$
- always test sensitivity of parameter values to `ssDF` definition

Salamander ssDF

Construct the rbs ssDF

- sf is the scrFrame created earlier
- use it to construct a state space object

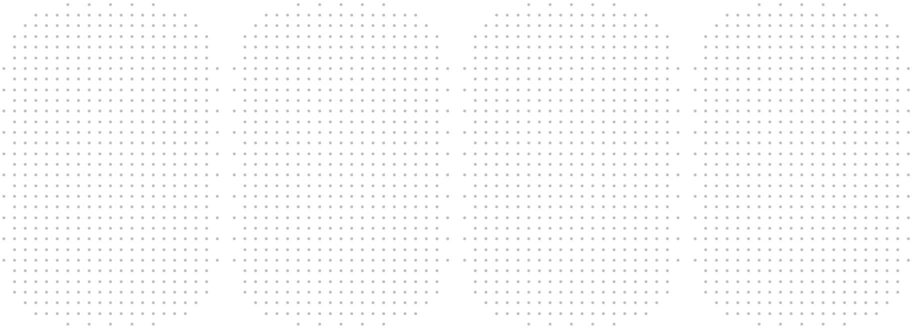
```
rbs.ss <- make.ssDF(scrFrame = rbs.sf,    # the rbs scrFrame
                    buffer = 3,           # 3 m
                    res = 0.5)            # 0.5 m
```

- visualize the state space object using plot() or plot.ssDF()

```
plot(rbs.ss)            # just the state space
plot(rbs.ss,rbs.sf)     # the state space and the traps
```

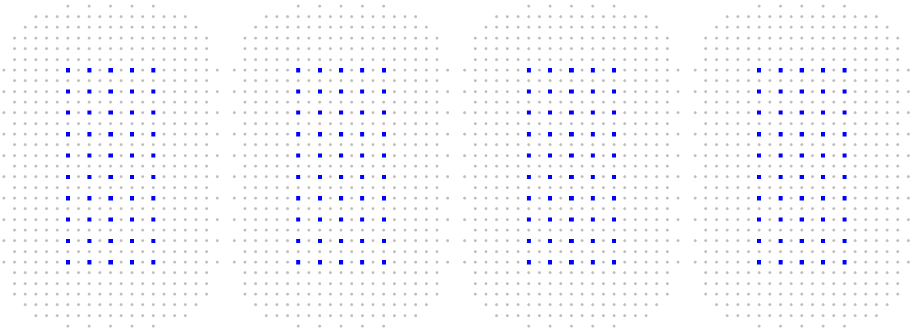
Salamander ssDF

```
plot(rbs.ss)          # just the state space
```



Salamander `ssDF`

```
plot(rbs.ss,rbs.sf)     # the state space and the traps
```



Salamander data preperation

```
# read in the data
rbs.edf <- read.csv("rbsNY.csv",h=T)
tdf1 <- read.csv("tdfNY1.csv",h=T)
tdf2 <- read.csv("tdfNY2.csv",h=T)
tdf3 <- read.csv("tdfNY3.csv",h=T)
tdf4 <- read.csv("tdfNY4.csv",h=T)

# create a 'encounters' data object (scrFrame)
rbs <- data2oscr(edf = rbs.edf, tdf = list(tdf1,tdf2,tdf3,tdf4),
                 sess.col = 1, id.col = 2, occ.col = 3, trap.col = 4,
                 K = c(7,5,6,4), ntraps = c(50,50,50,50))
rbs.sf <- rbs$scrFrame

# create a spatial extent object (ssDF)
rbs.ss <- make.ssDF(scrFrame = rbs.sf,
                    buffer = 3,
                    res = 0.5)
```

- next step, *model fitting*

`oSCR.fit` - the main model fitting function

To fit models in `oSCR`:

- use the fitting function `oSCR.fit()`
- *must* provide an `scrFrame`
- *must* provide an `ssdf`
- specify the model

oSCR.fit - the main model fitting function

To fit models in oSCR:

- use the fitting function oSCR.fit()
- *must* provide an scrFrame
- *must* provide an ssdf
- specify the model

```
mod <- oSCR.fit(model,        # model formulation
                scrFrame,     # the scrFrame object
                ssDF)         # the ssDF object
```

oSCR - Model fitting

```
mod <- oSCR.fit(model,          # model formulation
                scrFrame,       # the scrFrame object
                ssDF,           # the ssDF object
                ...)            # additional arguments
```

`model`:

- a list with 3 model formulations
- `list(D ~ 1, p0 ~ 1, sig ~ 1)`
- `D ~`: model describing variation pixel density ($D(s_i)$)
- `p0 ~`: model describing variation in baseline encounter prob/rate (p_0)
- `sig ~`: model describing variation in sigma (σ)

Model SCR_0 (the *Null* model)

Model SCR_0 in `oSCR` - Density

$$log(D(s_i)) = \beta$$

- inference about *per pixel* density, $D(s_i)$
- log-linear model to ensure positive densities
  - i.e. need to exponentiate estimate!
- intercept only model specification: `D ~ 1`

Model SCR_0 (the *Null* model)

Model SCR_0 in `oSCR` - baseline encounter probability

$$p[x_j, s_i] = p_0 \times \exp\left(-\frac{\text{dist}(x_j, s_i)^2}{2\sigma^2}\right)$$

$$logit(p_0) = \alpha_0$$

- inference about encounter probability, p_0
- constant across all individuals
- logit model for probabilities (ensures 0-1 bounds)
- intercept only model specification: `p0 ~ 1`

Model SCR_0 (the *Null* model)

Model SCR_0 in `oSCR` - spatial scale parameter

$$p[x_j, s_i] = p_0 \times \exp\left(-\frac{\text{dist}(x_j, s_i)^2}{2\sigma^2}\right)$$

$$log(\sigma) = \gamma_0$$

- inference about the spatial scale of detection, σ
- constant across all individuals
- log-linear model to ensure positive distances
  - i.e. need to exponentiate estimate!
- intercept only model specification: `sig ~ 1`

Model SCR_0 (the *Null* model) in oSCR

Fitting model SCR_0 is as simple as:

```
mod <- oSCR.fit(list(D ~ 1, p0 ~ 1, sig ~ 1), # model formulation
                scrFrame,                      # the scrFrame object
                ssDF,                          # the ssDF object
                ...)                           # additional arguments
```

So what are the `...` additional arguments?

- oSCR is *very* flexible and has many options/setting
- check the help file `?oSCR.fit()`
- BUT for salamnders with ACO, must use `multicatch=TRUE`

Model SCR_0 (the *Null* model) in `oSCR`

Fitting model SCR_0 is as simple as:

```
mod <- oSCR.fit(list(D ~ 1, p0 ~ 1, sig ~ 1), # model formulation
                scrFrame,                     # the scrFrame object
                ssDF,                         # the ssDF object
                multicatch=T)                 # additional arguments
```

So what are the . . . additional arguments?

- `oSCR` is *very* flexible and has many options/setting
- check the help file `?oSCR.fit()`
- BUT for salamnders with ACO, must use `multicatch=TRUE`

Model SCR_0 (the *Null* model) in oSCR

Let's fit SCR_0 to the salamander data

```
rbs.scr0 <- oSCR.fit(list(D ~ 1, p0 ~ 1, sig ~ 1), # SCR0
                     rbs.sf,                       # rbs encounter data
                     rbs.ss,                       # rbs study area
                     multicatch = TRUE)

Fitting model: D~1, p0~1, sigma~1, asu~1
Using ll function 'msLL.nosex'
Hold on tight!
2017-07-19 00:48:28
p0.(Intercept) |  sig.(Intercept) |  d0.(Intercept) |
```

Model SCR_0 (the *Null* model) in `oSCR`

Type the name of the model for a model summary:

```
rbs.scr0
 Model:  D ~ 1 p0 ~ 1 sig ~ 1
 Run time:  3.875  minutes
 AIC:  3119.959

Summary table:
                Estimate    SE       z P(>|z|)
p0.(Intercept)    -1.730 0.120 -14.406       0
sig.(Intercept)   -0.468 0.040 -11.690       0
d0.(Intercept)    -0.946 0.069 -13.735       0
*Density intercept is log(individuals per pixel)
  Nhat(state-space) = exp(d0.)*nrow(ssDF)
  (caution is warranted when model contains density covariates)
```

- model took quite a long time to run (~6.5 mins)!
- can reduce run time using (see `?oSCR.fit()`)

Salamander analysis workflow

```r
# read in the data
rbs.edf <- read.csv("rbsNY.csv",h=T)
tdf1 <- read.csv("tdfNY1.csv",h=T)
tdf2 <- read.csv("tdfNY2.csv",h=T)
tdf3 <- read.csv("tdfNY3.csv",h=T)
tdf4 <- read.csv("tdfNY4.csv",h=T)

# create a 'encounters' data object (scrFrame)
rbs <- data2oscr(edf = rbs.edf, tdf = list(tdf1,tdf2,tdf3,tdf4),
                 sess.col = 1, id.col = 2, occ.col = 3, trap.col = 4,
                 K = c(7,5,6,4), ntraps = c(50,50,50,50))
rbs.sf <- rbs$scrFrame

# create a spatial extent object (ssDF)
rbs.ss <- make.ssDF(scrFrame = rbs.sf,
                    buffer = 3,
                    res = 0.5)
# fit model SCR0
rbs.scr0  <- oSCR.fit(list(D~1, p0~1, sig~1), rbs.sf, rbs.ss, trimS=4)
```

- all that's left is to interpret the output - *inference*!

Interpreting and processing oSCR output

What can you say about these results?

```
rbs.scr0
 Model:  D ~ 1 p0 ~ 1 sig ~ 1
 Run time:  3.875  minutes
 AIC:  3119.959

Summary table:
                Estimate    SE       z P(>|z|)
p0.(Intercept)    -1.730 0.120 -14.406       0
sig.(Intercept)   -0.468 0.040 -11.690       0
d0.(Intercept)    -0.946 0.069 -13.735       0
*Density intercept is log(individuals per pixel)
  Nhat(state-space) = exp(d0.)*nrow(ssDF)
  (caution is warranted when model contains density covariates)
```

Interpreting and processing oSCR output

What can you say about these results?

```
rbs.scr0
 Model:  D ~ 1 p0 ~ 1 sig ~ 1
 Run time:  3.875  minutes
 AIC:  3119.959

Summary table:
                Estimate    SE       z P(>|z|)
p0.(Intercept)    -1.730 0.120 -14.406       0
sig.(Intercept)   -0.468 0.040 -11.690       0
d0.(Intercept)    -0.946 0.069 -13.735       0
*Density intercept is log(individuals per pixel)
  Nhat(state-space) = exp(d0.)*nrow(ssDF)
  (caution is warranted when model contains density covariates)
```

- on the *linear predictor*/*link* scale
- need to transform onto the *real* scale
- use `get.real()` function (needs `library(car)`)

Obtaining real scale estimates from oSCR output

```
# function for doing the back transformation
get.real(model,   # a fitted model
         type,    # the sub model to backtrasform (dens,det,sig)
         newdata) # [optional] new data to predict for
```

Obtaining real scale estimates from oSCR output

Session (plot) specific density estimates

```
# function for doing the back transformation
get.real(model = rbs.scr0,
         type = "dens",
         newdata = data.frame(session=factor(1:4)))
   estimate         se       lwr       upr
1 0.3883592 0.02674403 0.3359419 0.4407765
2 0.3883592 0.02674403 0.3359419 0.4407765
3 0.3883592 0.02674403 0.3359419 0.4407765
4 0.3883592 0.02674403 0.3359419 0.4407765
```

- remember, *NULL* model → no variation

Obtaining real scale estimates from oSCR output

Session (plot) specific encounter probability estimates

```
# function for doing the back transformation
get.real(model = rbs.scr0,
         type = "det",
         newdata = data.frame(session=factor(1:4)))
   estimate         se       lwr       upr
1 0.1505349 0.01535998 0.1204299 0.1806399
2 0.1505349 0.01535998 0.1204299 0.1806399
3 0.1505349 0.01535998 0.1204299 0.1806399
4 0.1505349 0.01535998 0.1204299 0.1806399
```

- remember, *NULL* model → no variation

Obtaining real scale estimates from oSCR output

Session (plot) specific sigma estimates

```
# function for doing the back transformation
get.real(model = rbs.scr0,
         type = "sig",
         newdata = data.frame(session=factor(1:4)))
   estimate         se      lwr      upr
1 0.6264615 0.02506298 0.577339 0.675584
2 0.6264615 0.02506298 0.577339 0.675584
3 0.6264615 0.02506298 0.577339 0.675584
4 0.6264615 0.02506298 0.577339 0.675584
```

- remember, *NULL* model $\rightarrow$ no variation

Model $SCR_{session}$ (the session specific model) in oSCR

Let's fit $SCR_{session}$ to the salamander data

```
rbs.scrS <- oSCR.fit(list(D ~ session, p0 ~ session, sig ~ session),
                     rbs.sf,                              # rbs encounter data
                     rbs.ss,                              # rbs study area
                     multicatch = TRUE)

Fitting model: D~session, p0~session, sigma~session, asu~1
Using ll function 'msLL.nosex'
Hold on tight!
2017-07-19 00:53:04
p0.(Intercept) |  p0.session2 |  p0.session3 |  p0.session4 |  sig.(Intercept) |
sig.session2 |  sig.session3 |  sig.session4 |  d0.(Intercept) |  d.beta.session2 |
d.beta.session3 |  d.beta.session4 |
```

Model $SCR_{session}$ (the session specific model) in oSCR

Type the name of the model for a model summary:

```
rbs.scrS
 Model:  D ~ session p0 ~ session sig ~ session
 Run time:  42.44167  minutes
 AIC:  3085.642

Summary table:
                Estimate    SE      z P(>|z|)
p0.(Intercept)    -1.795 0.200 -8.981   0.000
p0.session2        1.008 0.436  2.313   0.021
p0.session3       -0.033 0.275 -0.121   0.904
p0.session4        0.642 0.459  1.399   0.162
sig.(Intercept)   -0.338 0.070 -4.832   0.000
sig.session2      -0.672 0.118 -5.708   0.000
sig.session3      -0.047 0.095 -0.493   0.622
sig.session4      -0.484 0.134 -3.599   0.000
d0.(Intercept)    -1.143 0.127 -8.982   0.000
d.beta.session2    0.294 0.210  1.398   0.162
d.beta.session3    0.451 0.170  2.656   0.008
d.beta.session4    0.214 0.232  0.925   0.355
*Density intercept is log(individuals per pixel)
  Nhat(state-space) = exp(d0.)*nrow(ssDF)
  (caution is warranted when model contains density covariates)
```

Obtaining real scale estimates from oSCR output

session (plot) specific density estimates

```
# function for doing the back transformation
get.real(model = rbs.scrS,
         type = "dens",
         newdata = data.frame(session=factor(1:4)))
   estimate         se       lwr       upr
1 0.3187848 0.04057461 0.2392600 0.3983096
2 0.4277774 0.07161124 0.2874219 0.5681328
3 0.5003131 0.05614364 0.3902736 0.6103526
4 0.3950006 0.07654061 0.2449837 0.5450174
```

Obtaining real scale estimates from oSCR output

Session (plot) specific encounter probability estimates

```
# function for doing the back transformation
get.real(model = rbs.scrS,
         type = "det",
         newdata = data.frame(session=factor(1:4)))
   estimate         se        lwr       upr
1 0.1425107 0.02441925 0.09464984 0.1903715
2 0.3129154 0.08329148 0.14966712 0.4761637
3 0.1384872 0.02257794 0.09423521 0.1827391
4 0.2399675 0.07533310 0.09231735 0.3876177
```

Obtaining real scale estimates from oSCR output

Session (plot) specific sigma estimates

```
# function for doing the back transformation
get.real(model = rbs.scrS,
         type = "sig",
         newdata = data.frame(session=factor(1:4)))
   estimate         se       lwr       upr
1 0.6113697 0.05507952 0.5034159 0.7193236
2 0.3357536 0.02998213 0.2769897 0.3945175
3 0.5871868 0.03826876 0.5121814 0.6621921
4 0.3985914 0.04252299 0.3152479 0.4819350
```