secrBVN - simulation of spatially explicit capture-recapture with bivariate normal home ranges

Murray Efford

2018-08-16

Contents

The small package **secrBVN** is used to evaluate the performance of SECR estimators when home ranges are bivariate normal (BVN) or uniform (flat-topped) ellipses. We assume detection hazard is directly proportional to home range utilisation (activity). Code to use **secrBVN** for the simulations of Efford (in prep.) is provided in the Appendix.

The key user-visible functions are `simpopn.bvn`, `plotpopn.bvn`, `simcapt.bvn`, `runEllipseSim`, `simsum`, `simplot` and `anisotropic.fit`.

Generating and plotting elliptical home ranges

`simpopn.bvn` is a wrapper for the **secr** function `sim.popn` that adds attributes specifying a bivariate normal home range shape, size and orientation for each individual. By default, shape and size are the same for all individuals, but a mechanism is provided to vary them individually (see Heterogeneous elliptical home ranges).

First, load the package.

```
library(secrBVN)
simfolder <- "c:/density communication/noncircularity/paper/simulations/"
runall <- FALSE  # skip lengthy simulations
```

```
tempgrid <- make.grid(nx = 10, ny = 10)
par(mfrow = c(2,4), mar = c(2,2,2.6,2), xpd = TRUE)
for (i in 1:4) {
    s2xy <- 25^2 * c(1/i, i)
    random.pop <- simpopn.bvn(s2xy = s2xy, core = tempgrid, buffer = 100, D = 1)
    plotpopn.bvn(random.pop, col = 'lightblue')
    mtext(side=3, line=1.5, i)
}
for (i in 1:4) {
    s2xy <- 25^2 * c(1/i, i)
    aligned.pop <- simpopn.bvn(s2xy = s2xy, core = tempgrid, buffer = 100, D = 1, theta = -1)
    plotpopn.bvn(aligned.pop, col = 'lightblue')
}
```

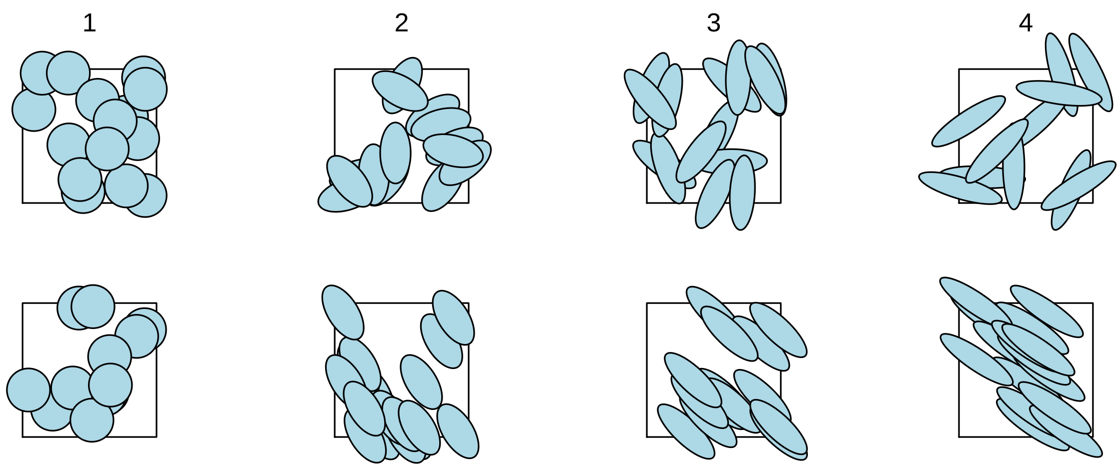


Fig. 1. Elliptical home ranges with varying ratio of major to minor axes as shown. Upper row oriented randomly and independently, lower row with shared random orientation ('randomly aligned').

Simulating elliptical detection data

Generating detection histories

Normally in **secr** we use `sim.capthist` to generate capture histories, but that is limited to circular detection functions. The **secrBVN** function `simcapt.bvn` is a partial replacement for `sim.capthist` that models detection with a bivariate normal. Specifically, the cumulative hazard of detection is a constant times the bivariate normal probability density at the detector. The constant is $\lambda_0 2\pi\sigma_X\sigma_Y$. The user provides a 'popn' object that includes the BVN parameter values $(\sigma_x^2, \sigma_y^2, \theta)$ for each animal (row), as generated above by `simpopn.bvn`. The constant scales the BVN density so that the maximum hazard is λ_0.

The preceding paragraph describes the default behaviour of `simcapt.bvn`. If `type = uniform` is selected then a uniform (flat-topped) elliptical home range is simulated. The uniform probability of detection within the ellipse is controlled by `g0`, not `lambda0`.

Wrapper function to generate BVN data and fit SECR model

The function `runEllipseSim` is a wrapper for the preceding steps (`simpopn.bvn`, `simcapt.bvn`) that also fits a standard (circular) SECR model with `secr.fit`[1]. The default population has a fixed number of individuals within the rectangular buffered area around the detectors (`Ndist = 'fixed'`).

For this example we use a 6×6 grid of binary proximity detectors with 50 metre spacing. The code in **secrBVN** does not allow for competition among detectors (secr detector type 'multi') or other other secr detector types. A density of 4/ha gives exactly 169 animals in the buffered area. Conditional likelihood is used for speed; the default `extractfn = derived` is compatible with both `CL = TRUE` and `CL = FALSE`. The 200-m buffer allows for the longest ranges ($\sigma_y = 50$ m).

```
tr <- make.grid(6, 6, spacing = 50, detector = 'proximity')
simrandomBVN36 <- vector('list', 4)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    simrandomBVN36[[i]] <- runEllipseSim (nrepl = 500, sigmaX, sigmaY, buffer = 200,
        ncores = 20, traps = tr, lambda0 = 0.4, D = 4, type = 'BVN',
        CL = TRUE, detectfn = 'HHN', details = list(distribution = 'binomial'))
}
```

The **secrBVN** function `simplot` is used to summarize the results

```
par(xpd = FALSE, mar = c(4,4,4,4))
simplot(list(BVN = simrandomBVN36))
```

[1] An elliptical SECR model also may be fitted - see Anisotropic home ranges.

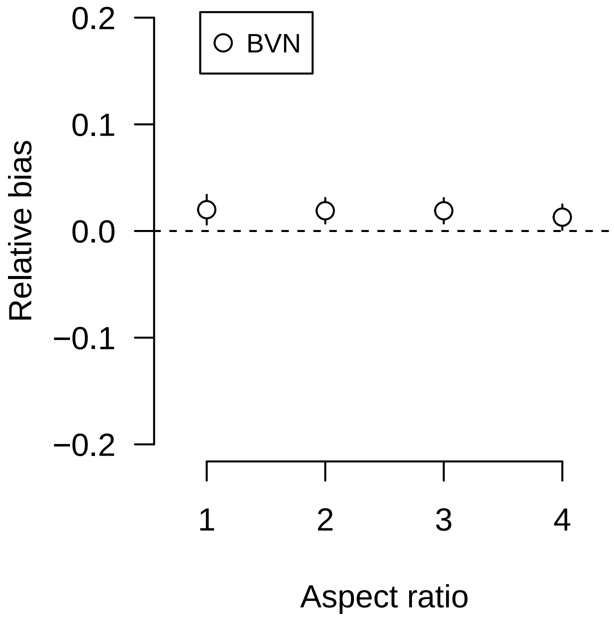


Fig. 2. Relative bias of density estimated by fitting circular SECR detection model to elliptical data, with 95% confidence limit for simulated values.

Next print a summary of the simulation results. There is no apparent effect of range elongation itself on the bias of the estimates.

```
simsum(list(BVN = simrandomBVN36))
```

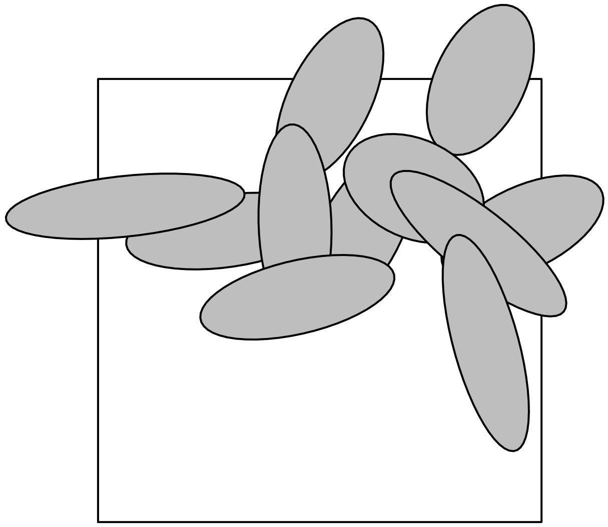
```
## $BVN
##      av.nCH    RB   RSE rRMSE   COV
## [1,] 42.184 0.020 0.140 0.149 0.940
## [2,] 42.942 0.019 0.139 0.146 0.934
## [3,] 43.950 0.019 0.139 0.143 0.946
## [4,] 44.704 0.013 0.139 0.142 0.948
```

The Appendix has many other examples.

Heterogeneous elliptical home ranges

`simpopn.bvn` by default generates a population with equal-sized home ranges. The size and elongation of each range may be varied by providing a user-defined function as the argument `s2xy`:

```
tr <- make.grid(6,6, spacing = 50, detector = 'proximity')
rs2xy <- function(N, scale = 25) {
    aspectratio <- 1 + runif(N) * 3
    cbind(scale^2 / aspectratio, scale^2 * aspectratio)
}
pop <- simpopn.bvn(s2xy = rs2xy, core = tr, buffer = 100, D = 1)
par(mfrow = c(1,1), mar = c(4,4,4,4), xpd = TRUE)
plotpopn.bvn(pop, col = 'grey')
```



Heterogeneous populations may also be simulated in `runEllipseSim` by passing a function as the argument 'sigmaX'.

```
sims <- runEllipseSim (10, sigmaX = rs2xy, buffer = 200, ncores = 2,
                       traps = tr, g0 = 0.2, D = 4, type = 'uniform')
```

Anisotropic home ranges: a partial solution

In principle, we can deal with elongated ranges by replacing Euclidean distances with distances in a space transformed to render home ranges circular (Murphy et al. 2016). The transformation compresses home ranges along their major axis. The transformation uses two parameters: ψ_A for the shared orientation measured in radians and ψ_R for the compression ratio ($\psi_R \geq 1$; $\psi_R = 1$ corresponds to no compression). The parameter ψ_A (psiA) corresponds to the argument `theta` used by `simpopn.bvn`. Thanks to Ben Augustine for pointing out the **geoR** function `coords.aniso` that lets us do this (Ribeiro and Diggle 2018)[2].

The method does not work if the detector array is strictly linear (2-D data are required to estimate the orientation and compression parameters). It is tested in the Appendix on simulated data from a hollow square array.

Distances in transformed space

The code uses a user-defined distance function that computes a Euclidean distance in the transformed space. In this version both transformation parameters are estimated (cf Murphy et al. 2016). Transformation parameters are handled in the undocumented 'miscparm' feature of `secr.fit`. This allows supernumerary unmodelled parameters to be passed to the distance function. The parameters are included in the vector of coefficients (beta parameters) over which the likelihood is maximised. 'Unmodelled' here means that the

[2] The package **intamap** (Pebesma et al. 2010) also offers anisotropic transformation in function `rotateAnisotropicData`.

parameter takes a single value for all animals, times and places. The link function is ‘identity’ for ψ_A and ‘log’ for $\psi_R - 1$.

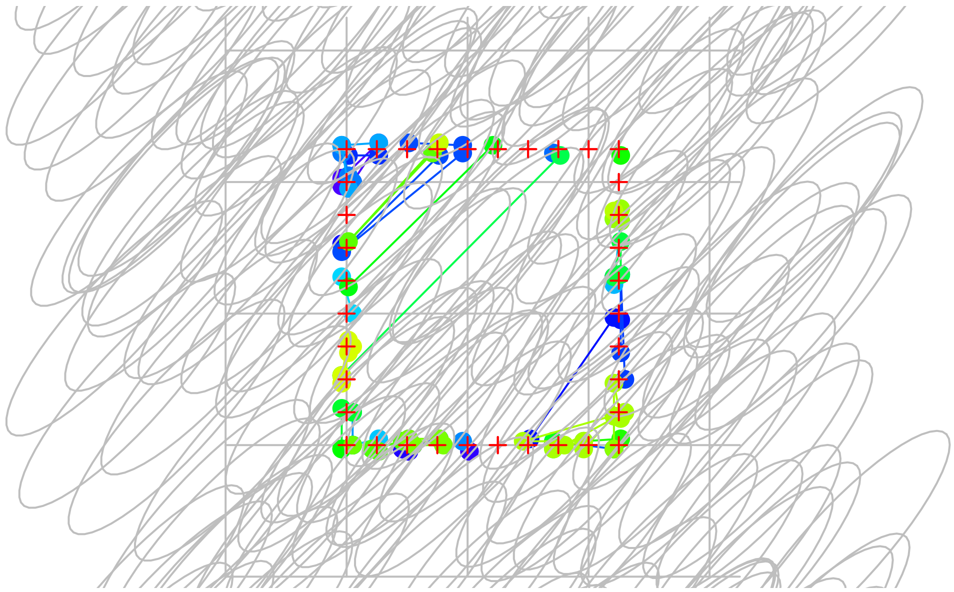
```
anisodistfn <- function (xy1, xy2, mask) {
    if (missing(xy1)) return(character(0))
    xy1 <- as.matrix(xy1)
    xy2 <- as.matrix(xy2)
    miscparm <- attr(mask, 'miscparm')
    psiA <- miscparm[1]           # anisotropy angle; identity link
    psiR <- 1 + exp(miscparm[2])  # anisotropy ratio; log link
    aniso.xy1 <- geoR::coords.aniso(xy1, aniso.pars = c(psiA, psiR))
    aniso.xy2 <- geoR::coords.aniso(xy2, aniso.pars = c(psiA, psiR))
    secr::edist(aniso.xy1, aniso.xy2) # nrow(xy1) x nrow(xy2) matrix
}
```

Example

Here is a simple application with oblique elliptical home ranges (`theta = pi/4`).

```
tr <- make.grid(10, 10, spacing = 25, hollow = TRUE, detector = 'proximity')
pop <- simpopn.bvn(s2xy = (25*c(0.5,2))^2, theta = pi/4, core = tr, buffer = 200,
                   D = 4, Ndist = 'fixed')
CH <- simcapt.bvn(tr, pop, type = 'BVN', lambda = 0.4, noccasions = 5)
```

```
par(mfrow = c(1,1))
plot(CH, tracks = TRUE)
plotpopn.bvn(pop, border='grey', add = TRUE)
plot(tr, add = TRUE)
```



First we fit a naive circular model.

```
fit0 <- secr.fit(CH, buffer = 200, detectfn = 'HHN', trace = FALSE,
                 details = list(distribution = 'binomial'))
predict(fit0)
```

```
##         link   estimate SE.estimate       lcl        ucl
## D        log  3.6504912  0.57084453  2.691832  4.9505631
## lambda0  log  0.2254549  0.04643798  0.151200  0.3361767
## sigma    log 35.3861664  2.91968125 30.110691 41.5859196
```

Next, the model with transformation to isotropy.

```
details <- list(distribution = 'binomial', userdist = anisodistfn,
                miscparm = c(psiA = 0.5, psiR = 1.5))  # initial values
fit1 <- secr.fit(CH, buffer = 200, detectfn = 'HHN', trace = FALSE,
                 details = details)
predict(fit1)
```

```
##         link   estimate SE.estimate       lcl        ucl
## D        log  3.2305502  0.56767427 2.2952952  4.5468900
## lambda0  log  0.3756727  0.06911078 0.2627358  0.5371557
## sigma    log 11.9762898  1.15093956 9.9244963 14.4522718
```

```
coef(fit1)
```

```
##               beta    SE.beta        lcl        ucl
## D        1.1726525 0.17438638  0.8308615  1.5144435
## lambda0 -0.9790369 0.18243679 -1.3366064 -0.6214673
```

```
## sigma     2.4829288 0.09588073  2.2950061  2.6708516
## psiA      0.8001388 0.03903624  0.7236292  0.8766484
## psiR      1.5127331 0.24364892  1.0351900  1.9902762
```

The estimated bearing is 45.84 degrees. The estimated aspect ratio is 5.54, (95% CI 3.82, 8.32).

Function to fit anisotropic model

The **secrBVN** function `anisotropic.fit` streamlines model fitting and does not require `anisodistfn` to be defined externally. Use it as you would use `secr.fit`. For example,

```
fit2 <- anisotropic.fit(CH,  buffer = 200, detectfn = 'HHN', trace = FALSE,
               details = list(distribution = "binomial"))
predictAniso(fit2) # estimates of transformation parameters psiA (in degrees) and psiR
```

```
##       estimate SE.estimate      lcl       ucl
## psiA 45.844562    2.236611 41.46088 50.228240
## psiR  5.539117    1.122569  3.81564  8.317549
```

Fixed orientation

If the orientation is known then there are two solutions:

1. Re-write anisodistfn for miscparm of length 1, with hard-coded value for psiA, or
2. Fix the beta parameter corresponding to psiA.

Here we demonstrate (2), fixing psiA at pi/4 (45 degrees). The `secr.fit` details argument fixedbeta is a vector of values, one per 'beta' parameter, using NA for each beta to be estimated. The first three beta parameters are for D, lambda0 and sigma; the next two are for psiA and psiR. If you model D, lambda0 or sigma there will be extra beta parameters, pushing psiA and psiR back.

```
fit3 <- anisotropic.fit(CH,  buffer = 200, detectfn = 'HHN', trace = FALSE,
            details = list(distribution = "binomial", fixedbeta = c(NA,NA,NA,pi/4,NA)))
predictAniso(fit3) # estimate of psiR only
```

```
##       estimate SE.estimate     lcl      ucl
## psiA 45.000000          NA      NA       NA
## psiR  5.521112    1.119021 3.80342 8.291257
```

```
predict(fit3)
```

```
##         link  estimate SE.estimate      lcl        ucl
## D        log  3.235872  0.56879269 2.298827  4.5548748
## lambda0  log  0.375622  0.06912855 0.262664  0.5371572
## sigma    log 11.985341  1.15169893 9.932175 14.4629345
```

Inadequate data

Detectors in a straight line provide very little information with which to estimate ψ_A and ψ_R. Here is an example.

```
tr <- make.grid(100, 1, spacing = 25, detector = 'proximity')
pop <- simpopn.bvn(s2xy = (25*c(0.5,2))^2, theta = pi/4, core = tr, buffer = 200,
                   D = 4, Ndist = 'fixed')
CH <- simcapt.bvn(tr, pop, type = 'BVN', lambda = 0.4, noccasions = 5)
```

```r
fit4 <- anisotropic.fit(CH,  buffer = 200, detectfn = 'HHN', trace = FALSE,
                details = list(distribution = "binomial"))
predictAniso(fit4) # estimates of transformation parameters psiA (in degrees) and psiR
```

```
##        estimate SE.estimate        lcl       ucl
## psiA 117.571941    8.635190 100.647280 134.49660
## psiR   3.403969    3.717638   1.275583  21.97034
```

```r
predict(fit4)
```

```
##         link  estimate SE.estimate       lcl        ucl
## D        log 8.3250577  1.31038719 6.1266727 11.3122716
## lambda0  log 0.2930468  0.03553798 0.2312532  0.3713524
## sigma    log 9.8707131  4.00729715 4.5906325 21.2238674
```

(Perhaps the parameters are not strictly nonidentifiable).

Package limitations

This package has the limited goal of determining how range elongation affects estimates of density in simple SECR models, and these specific limitations:

1. Only binary proximity detectors are supported.
2. The spatial distribution of activity centres is assumed to be homogeneous Poisson.
3. Ellipses are specified using either 'sigmaX' and 'sigmaY' as separate arguments (`runEllipseSim`) or as a vector of the two values, squared ('s2xy' in `simpopn.bvn`). This is confusing but it's better at this point not to change.

References

Efford, M. G. (2004) Density estimation in live-trapping studies. *Oikos* **106**, 598–610.

Efford, M. G. In prep. Non-circular home ranges and the estimation of population density.

Huggins, R. M. (1989) On the statistical analysis of capture experiments. *Biometrika* **76**, 133–140.

Ivan, J. S., White, G. C. and Shenk, T. M. (2013) Using simulation to compare methods for estimating density from capture–recapture data. *Ecology* **94**, 817–826.

Murphy, S. M., Cox, J. J., Augustine, B. C., Hast, J. T., Guthrie, J. M., Wright, J., McDermott, J., Maehr, S. C. and Plaxico, J. H. (2016) Characterizing recolonization by a reintroduced bear population using genetic spatial capture–recapture. *Journal of Wildlife Management* **80**, 1390–1407.

Pebesma, E., Cornford, D., Dubois, G., Heuvelink, G.B.M., Hristopoulos, D., Pilz, J., Stoehlker, U., Morin, G. and Skoien, J.O. (2010) INTAMAP: the design and implementation of an interoperable automated interpolation web service. *Computers & Geosciences* **37**, 343–352.

Ribeiro, P. J. Jr and Diggle, P. J. (2018) geoR: Analysis of Geostatistical Data. R package version 1.7-5.2.1. https://CRAN.R-project.org/package=geoR.

Appendix. Code for simulations of Efford (in prep)

```
library(secrBVN)
simfolder <- "c:/density communication/noncircularity/paper/simulations/"
nrepl <- 500
ncores <- 20                                 # for machine with at least 20 cores
runsim <- FALSE                              # change to TRUE to execute simulations
details <- list(distribution = 'binomial') # all simulations assume fixed N
```

Functions for heterogeneous aspect ratio.

```
# Uniform on 1-4
rs2xy <- function(N, scale = 25) {
    i <- 1 + runif(N) * 3
    cbind(scale^2 /i, scale^2 * i)
}
# 2 classes 1,4
rs2xy2 <- function(N, scale = 25, prob = c(0.5,0.5)) {
    i <- sample(c(1,4), size = N, replace = TRUE, prob = prob)
    cbind(scale^2 /i, scale^2 * i)
}
```

Main simulations

10 x 10 grid

Elongated ranges are oriented at random with respect to the grid (default `theta = NULL`).

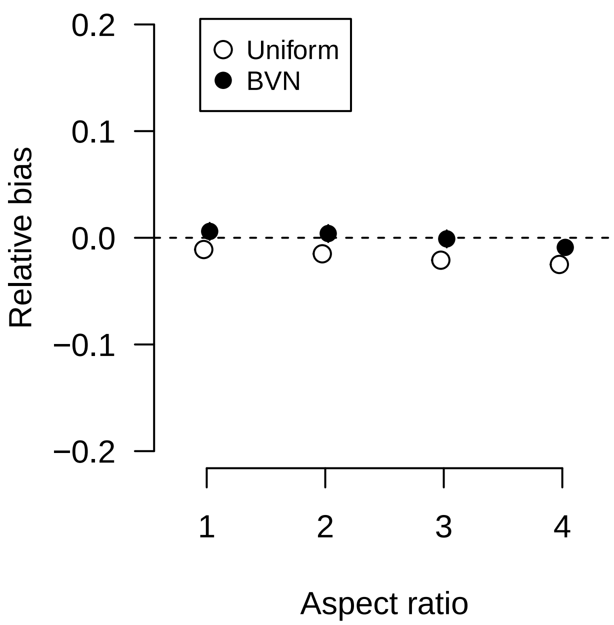
```
tr <- make.grid(10,10, spacing = 50, detector = 'proximity')
simrandom100.3 <- vector('list', 6)
names(simrandom100.3) <- c(1:4,'1-4','1,4')
simrandomBVN100.3 <- simrandom100.3
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr, theta = NULL,
                 D = 4, CL = TRUE, detectfn = 'HHN', details = details, seed = 347)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    # uniform
    args$type <- 'uniform'; args$g0 <- 0.2
    simrandom100.3[[i]] <- do.call(runEllipseSim, args)
    # bvn
    args$type <- 'BVN'; args$lambda0 <- 0.4
    simrandomBVN100.3[[i]] <- do.call(runEllipseSim, args)
    message ('Completed aspect ratio', i)
}
# heterogeneous aspect ratio 1,4, uniform
args$type <- 'uniform'; args$sigmaX <- rs2xy
simrandom100.3[[5]] <- do.call(runEllipseSim, args)
# heterogeneous aspect ratio 1,4, BVN
args$type <- 'BVN'; args$sigmaX <- rs2xy
simrandomBVN100.3[[5]] <- do.call(runEllipseSim, args)
# heterogeneous aspect ratio 1,4, uniform
args$type <- 'uniform'; args$sigmaX <- rs2xy2
```

```r
simrandom100.3[[6]] <- do.call(runEllipseSim, args)
#heterogeneous aspect ratio 1,4, BVN
args$type <- 'BVN'; args$sigmaX <- rs2xy2; args$seed <- 347
simrandomBVN100.3[[6]] <- do.call(runEllipseSim, args)

save(simrandom100.3, file = paste0(simfolder, 'simrandom100.3.RData'))
save(simrandomBVN100.3, file = paste0(simfolder, 'simrandomBVN100.3.RData'))
```

```r
load(file = paste0(simfolder, 'simrandom100.3.RData'))
load(file = paste0(simfolder, 'simrandomBVN100.3.RData'))

par(mfrow = c(1,1), mar = c(4,4,4,4), xpd = FALSE)
# select only first 4 scenarios for plotting
simplot(list(Uniform = simrandom100.3[1:4], BVN = simrandomBVN100.3[1:4]), legend = TRUE)
```




```r
# tabulate all
simsum(list(Uniform = simrandom100.3, BVN = simrandomBVN100.3), compact = NULL, dec = 4)
```

```
## $Uniform
##                    1        2        3        4      1-4      1,4
## av.npop     289.0000 289.0000 289.0000 289.0000 289.0000 289.0000
## av.nCH      121.0620 123.2520 126.3220 129.2880 124.7580 125.4240
## nvalid      500.0000 500.0000 500.0000 500.0000 500.0000 500.0000
## av.parmhat    3.9543   3.9404   3.9174   3.9005   3.9168   3.9210
## md.parmhat    3.9722   3.9454   3.9324   3.9116   3.9189   3.9181
## sd.parmhat    0.2829   0.2852   0.2760   0.2695   0.2689   0.2712
## RB           -0.0114  -0.0149  -0.0207  -0.0249  -0.0208  -0.0197
## seRB          0.0032   0.0032   0.0031   0.0030   0.0030   0.0030
## RSE           0.0686   0.0676   0.0662   0.0650   0.0669   0.0666
```

```
## seRSE        0.0001   0.0001   0.0001   0.0001   0.0001   0.0001
## rRMSE        0.0716   0.0728   0.0720   0.0718   0.0703   0.0706
## COV          0.9400   0.9320   0.9280   0.9000   0.9280   0.9380
##
## $BVN
##                   1        2        3        4      1-4      1,4
## av.npop    289.0000 289.0000 289.0000 289.0000 289.0000 289.0000
## av.nCH     108.3260 109.4820 110.9340 112.0540 110.0220 109.9820
## nvalid     500.0000 500.0000 500.0000 500.0000 500.0000 500.0000
## av.parmhat   4.0257   4.0172   3.9958   3.9652   3.9895   3.9731
## md.parmhat   4.0378   4.0215   3.9992   3.9739   4.0094   3.9591
## sd.parmhat   0.3202   0.3280   0.3247   0.3100   0.3116   0.3089
## RB           0.0064   0.0043  -0.0011  -0.0087  -0.0026  -0.0067
## seRB         0.0036   0.0037   0.0036   0.0035   0.0035   0.0035
## RSE          0.0773   0.0769   0.0765   0.0762   0.0767   0.0767
## seRSE        0.0001   0.0001   0.0001   0.0001   0.0001   0.0001
## rRMSE        0.0802   0.0820   0.0811   0.0779   0.0779   0.0774
## COV          0.9500   0.9420   0.9360   0.9400   0.9500   0.9440
```

```
# compact table for paper
simsum(list(Uniform = simrandom100.3, BVN = simrandomBVN100.3))
```

```
## $Uniform
##       av.nCH     RB   RSE rRMSE   COV
## 1    121.062 -0.011 0.069 0.072 0.940
## 2    123.252 -0.015 0.068 0.073 0.932
## 3    126.322 -0.021 0.066 0.072 0.928
## 4    129.288 -0.025 0.065 0.072 0.900
## 1-4  124.758 -0.021 0.067 0.070 0.928
## 1,4  125.424 -0.020 0.067 0.071 0.938
##
## $BVN
##       av.nCH     RB   RSE rRMSE   COV
## 1    108.326  0.006 0.077 0.080 0.950
## 2    109.482  0.004 0.077 0.082 0.942
## 3    110.934 -0.001 0.077 0.081 0.936
## 4    112.054 -0.009 0.076 0.078 0.940
## 1-4  110.022 -0.003 0.077 0.078 0.950
## 1,4  109.982 -0.007 0.077 0.077 0.944
```

```
# spatial scale parameter
output <- simsum(list(Uniform = simrandom100.3, BVN = simrandomBVN100.3),
    component = 'pred', parm = 'sigma', compact = c("av.parmhat", "sd.parmhat"))
lapply(output, '/', 50)  # in units of detector spacing
```

```
## $Uniform
##     av.parmhat sd.parmhat
## 1      0.59046    0.01422
## 2      0.65116    0.01836
## 3      0.74018    0.02470
## 4      0.82576    0.02966
## 1-4    0.70118    0.02356
## 1,4    0.71320    0.02736
##
## $BVN
##     av.parmhat sd.parmhat
```

```
## 1      0.50072    0.01780
## 2      0.55270    0.02558
## 3      0.62760    0.03130
## 4      0.69782    0.03806
## 1-4    0.59366    0.03278
## 1,4    0.60450    0.03652
```

Straight line 36-detectors

```
tr <- make.grid(36, 1, spacing = 50, detector = 'proximity')
simgridalignlinw1.1 <- vector('list', 6)
names(simgridalignlinw1.1) <- c(1:4,'1-4','1,4')
simgridalignlinw4.1 <- simgridalignlinw3.1 <- simgridalignlinw2.1 <- simgridalignlinw1.1
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 lambda0 = 0.4, D = 4, type = 'BVN',
                 CL = TRUE, detectfn = 'HHN', details = details)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    # bvn parallel to traps
    args$theta <- 0; simgridalignlinw1.1[[i]] <- do.call(runEllipseSim, args)
    # bvn oblique to traps
    args$theta <- pi/4; simgridalignlinw2.1[[i]] <- do.call(runEllipseSim, args)
    # bvn perpendicular to traps
    args$theta <- pi/2; simgridalignlinw3.1[[i]] <- do.call(runEllipseSim, args)
    # bvn random orientation
    args$theta <- NULL; simgridalignlinw4.1[[i]] <- do.call(runEllipseSim, args)
}

args <- c(baseargs, list(sigmaX = rs2xy))
args$theta <- 0; simgridalignlinw1.1[[5]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignlinw2.1[[5]] <- do.call(runEllipseSim, args)
args$theta <- pi/2; simgridalignlinw3.1[[5]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignlinw4.1[[5]] <- do.call(runEllipseSim, args)

args <- c(baseargs, list(sigmaX = rs2xy2))
args$theta <- 0; simgridalignlinw1.1[[6]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignlinw2.1[[6]] <- do.call(runEllipseSim, args)
args$theta <- pi/2; simgridalignlinw3.1[[6]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignlinw4.1[[6]] <- do.call(runEllipseSim, args)

save(simgridalignlinw1.1, file = paste0(simfolder, 'simgridalignlinw1.1.RData'))
save(simgridalignlinw2.1, file = paste0(simfolder, 'simgridalignlinw2.1.RData'))
save(simgridalignlinw3.1, file = paste0(simfolder, 'simgridalignlinw3.1.RData'))
save(simgridalignlinw4.1, file = paste0(simfolder, 'simgridalignlinw4.1.RData'))
```

```
# compact table linear
load(file = paste0(simfolder, 'simgridalignlinw1.1.RData'))
load(file = paste0(simfolder, 'simgridalignlinw2.1.RData'))
load(file = paste0(simfolder, 'simgridalignlinw3.1.RData'))
load(file = paste0(simfolder, 'simgridalignlinw4.1.RData'))
simsum(list(Parallel       = simgridalignlinw1.1,
            Oblique        = simgridalignlinw2.1,
```

```
             Perpendicular = simgridalignlinw3.1,
             Random        = simgridalignlinw4.1))
```

```
## $Parallel
##     av.nCH    RB   RSE rRMSE   COV
## 1   56.452 0.011 0.171 0.179 0.944
## 2   66.474 0.867 0.159 0.926 0.052
## 3   70.976 1.447 0.161    NA 0.000
## 4   73.130 1.791 0.168    NA 0.000
## 1-4 67.740 1.008 0.160 1.057 0.008
## 1,4 64.686 0.643 0.167 0.710 0.206
##
## $Oblique
##     av.nCH    RB   RSE rRMSE   COV
## 1   56.452 0.011 0.171 0.179 0.944
## 2   59.770 0.262 0.168 0.337 0.718
## 3   64.032 0.619 0.162 0.674 0.206
## 4   67.228 0.942 0.157 0.987 0.022
## 1-4 61.960 0.423 0.167 0.483 0.468
## 1,4 61.666 0.402 0.168 0.475 0.488
##
## $Perpendicular
##     av.nCH     RB   RSE rRMSE   COV
## 1   56.452  0.011 0.171 0.179 0.944
## 2   46.480 -0.490 0.167 0.499 0.018
## 3   40.924 -0.661 0.163 0.664 0.000
## 4   37.206 -0.746 0.160 0.748 0.000
## 1-4 44.372 -0.578 0.166 0.583 0.000
## 1,4 46.732 -0.553 0.166 0.559 0.008
##
## $Random
##     av.nCH    RB   RSE rRMSE   COV
## 1   56.400 0.029 0.172 0.181 0.954
## 2   58.290 0.075 0.173 0.207 0.908
## 3   60.340 0.124 0.175 0.277 0.832
## 4   61.762 0.171 0.176 0.343 0.764
## 1-4 59.258 0.106 0.173 0.263 0.816
## 1,4 59.360 0.094 0.173 0.243 0.862
```

Hollow square 36 detectors

```
tr <- make.grid(10, 10, spacing = 50, detector = 'proximity', hollow = TRUE)
simgridalignsqw1.1 <- vector('list', 4)  # 'w' for wide
names(simgridalignsqw1.1) <- 1:4
simgridalignsqw3.1 <- simgridalignsqw2.1 <- simgridalignsqw1.1
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 lambda0 = 0.4, D = 4, type = 'BVN',
                 CL = TRUE, detectfn = 'HHN', details = details)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    # bvn parallel to traps
```

```r
    args$theta <- 0; simgridalignsqw1.1[[i]] <- do.call(runEllipseSim, args)
    # bvn oblique to traps
    args$theta <- pi/4; simgridalignsqw2.1[[i]] <- do.call(runEllipseSim, args)
    # bvn random orientation
    args$theta <- NULL; simgridalignsqw3.1[[i]] <- do.call(runEllipseSim, args)
}

args <- c(baseargs, list(sigmaX = rs2xy))
args$theta <- 0;    simgridalignsqw1.1[[5]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignsqw2.1[[5]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignsqw3.1[[5]] <- do.call(runEllipseSim, args)

args <- c(baseargs, list(sigmaX = rs2xy2))
args$theta <- 0;    simgridalignsqw1.1[[6]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignsqw2.1[[6]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignsqw3.1[[6]] <- do.call(runEllipseSim, args)

save(simgridalignsqw1.1, file = paste0(simfolder, 'simgridalignsqw1.1.RData'))
save(simgridalignsqw2.1, file = paste0(simfolder, 'simgridalignsqw2.1.RData'))
save(simgridalignsqw3.1, file = paste0(simfolder, 'simgridalignsqw3.1.RData'))
```

```r
# compact table square
load(file = paste0(simfolder, 'simgridalignsqw1.1.RData'))
load(file = paste0(simfolder, 'simgridalignsqw2.1.RData'))
load(file = paste0(simfolder, 'simgridalignsqw3.1.RData'))
simsum(list(Aligned = simgridalignsqw1.1,
            Oblique = simgridalignsqw2.1,
            Random  = simgridalignsqw3.1))
```

```
## $Aligned
##   av.nCH     RB   RSE rRMSE   COV
## 1 55.580  0.021 0.166 0.177 0.958
## 2 55.792 -0.067 0.165 0.189 0.882
## 3 55.724 -0.191 0.165 0.248 0.658
## 4 55.220 -0.297 0.163 0.336 0.368
##
## $Oblique
##   av.nCH    RB   RSE rRMSE   COV
## 1 55.580 0.021 0.166 0.177 0.958
## 2 58.794 0.198 0.168 0.284 0.800
## 3 62.406 0.421 0.171 0.513 0.492
## 4 65.362 0.616 0.175 0.733 0.306
##
## $Random
##   av.nCH    RB   RSE rRMSE   COV
## 1 55.468 0.020 0.166 0.170 0.940
## 2 57.068 0.061 0.167 0.212 0.906
## 3 59.110 0.108 0.169 0.275 0.834
## 4 60.814 0.140 0.170 0.314 0.808
```

Reduced spacing (σ instead of 2σ)

10 x 10 grid

```
tr <- make.grid(10,10, spacing = 25, detector = 'proximity')
simrandomBVNs100.1 <- vector('list', 4)
names(simrandomBVNs100.1) <- 1:4
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr, theta = NULL,
                 D = 4, CL = TRUE, detectfn = 'HHN', details = details, seed = 347,
                 type = 'BVN', lambda = 0.4)
for (i in 1:4) {
    args <- c(baseargs, list(sigmaX = 25/i^0.5, sigmaY = 25*i^0.5))
    simrandomBVNs100.1[[i]] <- do.call(runEllipseSim, args)
}
save(simrandomBVNs100.1, file = paste0(simfolder, 'simrandomBVNs100.1.RData'))
```

Straight line

```
tr <- make.grid(36, 1, spacing = 25, detector = 'proximity')
simgridalignlinw1.1 <- vector('list', 6)
names(simgridalignlinw1.1) <- c(1:4,'1-4','1,4')
simgridalignlinw4.1 <- simgridalignlinw3.1 <- simgridalignlinw2.1 <- simgridalignlinw1.1
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 lambda0 = 0.4, D = 4, type = 'BVN', theta = 0,
                 CL = TRUE, detectfn = 'HHN', details = details)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    # bvn parallel to traps
    args$theta <- 0; simgridalignlin1.3[[i]] <- do.call(runEllipseSim, args)
    # bvn oblique to traps
    args$theta <- pi/4; simgridalignlin2.3[[i]] <- do.call(runEllipseSim, args)
    # bvn perpendicular to traps
    args$theta <- pi/2; simgridalignlin3.3[[i]] <- do.call(runEllipseSim, args)
    # bvn random orientation
    args$theta <- NULL; simgridalignlin4.3[[i]] <- do.call(runEllipseSim, args)
}

args <- c(baseargs, list(sigmaX = rs2xy))
args$theta <- 0;    simgridalignlin1.3[[5]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignlin2.3[[5]] <- do.call(runEllipseSim, args)
args$theta <- pi/2; simgridalignlin3.3[[5]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignlin4.3[[5]] <- do.call(runEllipseSim, args)

args <- c(baseargs, list(sigmaX = rs2xy2))
args$theta <- 0;    simgridalignlin1.3[[6]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignlin2.3[[6]] <- do.call(runEllipseSim, args)
args$theta <- pi/2; simgridalignlin3.3[[6]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignlin4.3[[6]] <- do.call(runEllipseSim, args)

save(simgridalignlin1.3, file = paste0(simfolder, 'simgridalignlin1.3.RData'))
save(simgridalignlin2.3, file = paste0(simfolder, 'simgridalignlin2.3.RData'))
```

```
save(simgridalignlin3.3, file = paste0(simfolder, 'simgridalignlin3.3.RData'))
save(simgridalignlin4.3, file = paste0(simfolder, 'simgridalignlin4.3.RData'))
```

Hollow square

```
tr <- make.grid(10, 10, spacing = 25, detector = 'proximity', hollow = TRUE)
simgridalignsq1.1 <- vector('list', 4)  # 'w' for wide
names(simgridalignsq1.1) <- 1:4
simgridalignsq3.1 <- simgridalignsq2.1 <- simgridalignsq1.1
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 lambda0 = 0.4, D = 4, type = 'BVN',
                 CL = TRUE, detectfn = 'HHN', details = details)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    # bvn parallel to traps
    args$theta <- 0; simgridalignsq1.1[[i]] <- do.call(runEllipseSim, args)
    # bvn oblique to traps
    args$theta <- pi/4; simgridalignsq2.1[[i]] <- do.call(runEllipseSim, args)
    # bvn random orientation
    args$theta <- NULL; simgridalignsq3.1[[i]] <- do.call(runEllipseSim, args)
}

args <- c(baseargs, list(sigmaX = rs2xy))
args$theta <- 0; simgridalignsq1.1[[5]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignsq2.1[[5]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignsq3.1[[5]] <- do.call(runEllipseSim, args)

args <- c(baseargs, list(sigmaX = rs2xy2))
args$theta <- 0; simgridalignsq1.1[[6]] <- do.call(runEllipseSim, args)
args$theta <- pi/4; simgridalignsq2.1[[6]] <- do.call(runEllipseSim, args)
args$theta <- NULL; simgridalignsq3.1[[6]] <- do.call(runEllipseSim, args)

save(simgridalignsq1.2, file = paste0(simfolder, 'simgridalignsq1.2.RData'))
save(simgridalignsq2.2, file = paste0(simfolder, 'simgridalignsq2.2.RData'))
save(simgridalignsq3.2, file = paste0(simfolder, 'simgridalignsq3.2.RData'))
```

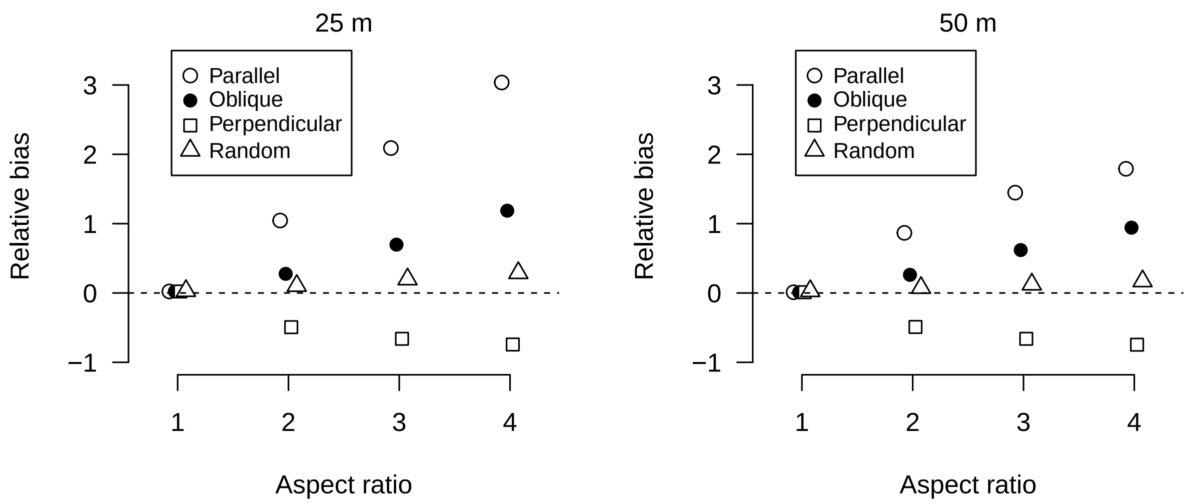
Spacing comparisons

10 x 10 grid

```
load(file = paste0(simfolder, 'simrandomBVN100.3.RData'))
load(file = paste0(simfolder, 'simrandomBVNs100.1.RData'))
par(mfrow=c(1,2))
simplot(list(BVN = simrandomBVNs100.1), legend = FALSE)
mtext(side=3, text = '25 m')
simplot(list(BVN = simrandomBVN100.3[1:4]), legend = FALSE)
mtext(side=3, text = '50 m')
```

Straight line

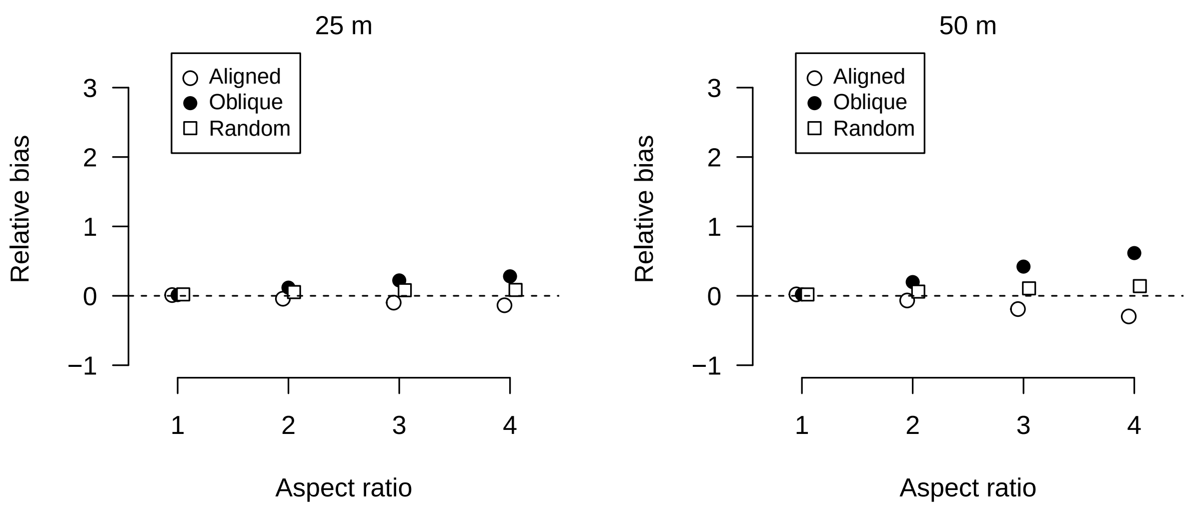
```
load(file = paste0(simfolder, 'simgridalignlin1.3.RData'))
load(file = paste0(simfolder, 'simgridalignlin2.3.RData'))
load(file = paste0(simfolder, 'simgridalignlin3.3.RData'))
load(file = paste0(simfolder, 'simgridalignlin4.3.RData'))
load(file = paste0(simfolder, 'simgridalignlinw1.1.RData'))
load(file = paste0(simfolder, 'simgridalignlinw2.1.RData'))
load(file = paste0(simfolder, 'simgridalignlinw3.1.RData'))
load(file = paste0(simfolder, 'simgridalignlinw4.1.RData'))
par(mfrow=c(1,2))
simplot(list(Parallel      = simgridalignlin1.3[1:4],
             Oblique       = simgridalignlin2.3[1:4],
             Perpendicular = simgridalignlin3.3[1:4],
             Random        = simgridalignlin4.3[1:4]),
        legend = TRUE, ylim = c(-1,3.5))
mtext(side=3, text = '25 m')
simplot(list(Parallel      = simgridalignlinw1.1[1:4],
             Oblique       = simgridalignlinw2.1[1:4],
             Perpendicular = simgridalignlinw3.1[1:4],
             Random        = simgridalignlinw4.1[1:4]),
        legend = TRUE, ylim = c(-1,3.5))
mtext(side=3, text = '50 m')
```



Hollow square

```
load(file = paste0(simfolder, 'simgridalignsq1.2.RData'))
load(file = paste0(simfolder, 'simgridalignsq2.2.RData'))
load(file = paste0(simfolder, 'simgridalignsq3.2.RData'))
load(file = paste0(simfolder, 'simgridalignsqw1.1.RData'))
load(file = paste0(simfolder, 'simgridalignsqw2.1.RData'))
load(file = paste0(simfolder, 'simgridalignsqw3.1.RData'))
par(mfrow =c(1,2))
simplot(list(Aligned  = simgridalignsq1.2[1:4],
             Oblique  = simgridalignsq2.2[1:4],
```

```
             Random   = simgridalignsq3.2[1:4]),
        legend = TRUE, ylim = c(-1,3.5))
mtext(side=3, text = '25 m')
simplot(list(Aligned  = simgridalignsqw1.1[1:4],
             Oblique  = simgridalignsqw2.1[1:4],
             Random   = simgridalignsqw3.1[1:4]),
        legend = TRUE, ylim = c(-1,3.5))
mtext(side=3, text = '50 m')
```



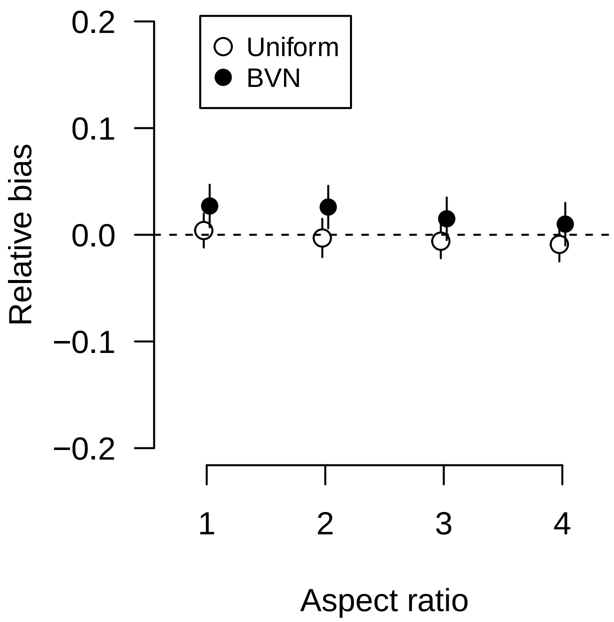
Variations

Low density (0.5/ha)

```
tr <- make.grid(10, 10, spacing = 50, detector = 'proximity')
simrandom100low.2 <- vector('list', 4)
names(simrandom100low.2) <- 1:4
simrandomBVN100low.2 <- simrandom100low.2
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 D = 0.5, CL = TRUE, detectfn = 'HHN', details = details)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    args$type <- 'uniform'; args$g0 <- 0.2
    simrandom100low.2[[i]] <- do.call(runEllipseSim, args)
    args$type <- 'BVN'; args$lambda0 <- 0.4
    simrandomBVN100low.2[[i]] <- do.call(runEllipseSim, args)
}
save(simrandom100low.2, file = paste0(simfolder, 'simrandom100low.2.RData'))
save(simrandomBVN100low.2, file = paste0(simfolder, 'simrandomBVN100low.2.RData'))
```

```
load(file = paste0(simfolder, 'simrandom100low.2.RData'))
load(file = paste0(simfolder, 'simrandomBVN100low.2.RData'))
```

```r
par(mfrow = c(1,1), mar = c(4,4,4,4), xpd = FALSE)
simplot(list(Uniform = simrandom100low.2, BVN = simrandomBVN100low.2),
    trueval = 0.5, legend = TRUE)
```




```r
simsum(list(Uniform = simrandom100low.2, BVN = simrandomBVN100low.2),
    trueval = 0.5, compact = NULL)
```

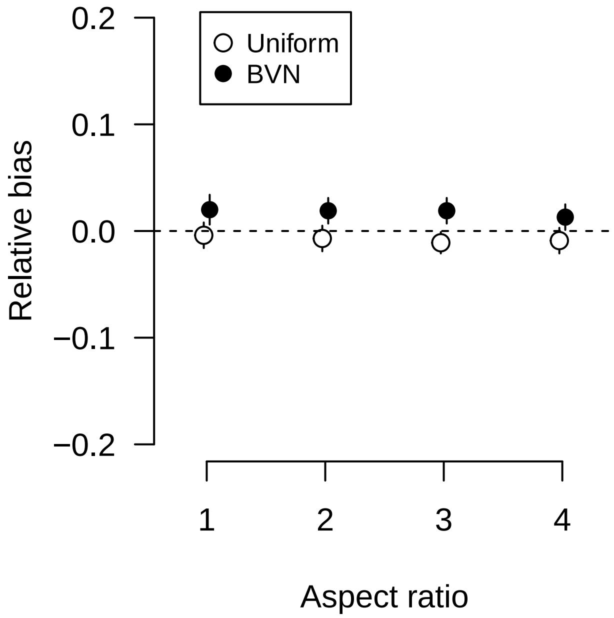
```
## $Uniform
##                  1       2       3       4
## av.npop     36.122  36.122  36.122  36.122
## av.nCH      15.310  15.488  15.906  16.318
## nvalid     500.000 500.000 500.000 500.000
## av.parmhat   0.502   0.498   0.497   0.495
## md.parmhat   0.495   0.496   0.499   0.497
## sd.parmhat   0.095   0.096   0.094   0.093
## RB           0.004  -0.003  -0.006  -0.009
## seRB         0.008   0.009   0.008   0.008
## RSE          0.196   0.194   0.190   0.186
## seRSE        0.001   0.001   0.001   0.001
## rRMSE        0.189   0.191   0.188   0.186
## COV          0.952   0.954   0.932   0.938
##
## $BVN
##                  1       2       3       4
## av.npop     36.122  36.124  36.122  36.122
## av.nCH      13.630  13.722  13.904  14.072
## nvalid     500.000 499.000 500.000 500.000
## av.parmhat   0.513   0.513   0.507   0.505
## md.parmhat   0.509   0.505   0.504   0.497
```

```
## sd.parmhat   0.109   0.110   0.110   0.111
## RB           0.027   0.026   0.015   0.010
## seRB         0.010   0.010   0.010   0.010
## RSE          0.224   0.224   0.222   0.221
## seRSE        0.001   0.001   0.001   0.001
## rRMSE        0.219      NA   0.220   0.223
## COV          0.946   0.946   0.944   0.954
```

Small array (6 x 6 grid)

```
tr <- make.grid(6, 6, spacing = 50, detector = 'proximity')
simrandom36.2 <- vector('list', 4)
names(simrandom36.2) <- 1:4
simrandomBVN36.2 <- simrandom36.2
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 D = 4, CL = TRUE, detectfn = 'HHN', details = details)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    args$type <- 'uniform'; args$g0 <- 0.2
    simrandom36.2[[i]] <- do.call(runEllipseSim, args)
    args$type <- 'BVN'; args$lambda0 <- 0.4
    simrandomBVN36.2[[i]] <- do.call(runEllipseSim, args)
}
save(simrandom36.2, file = paste0(simfolder, 'simrandom36.2.RData'))
save(simrandomBVN36.2, file = paste0(simfolder, 'simrandomBVN36.2.RData'))
```

```
load(file = paste0(simfolder, 'simrandom36.2.RData'))
load(file = paste0(simfolder, 'simrandomBVN36.2.RData'))
par(mfrow = c(1,1), mar = c(4,4,4,4), xpd = FALSE)
simplot(list(Uniform = simrandom36.2[1:4], BVN = simrandomBVN36.2[1:4]), legend = TRUE)
```




```
simsum(list(Uniform = simrandom36.2, BVN = simrandomBVN36.2), compact = NULL)
```

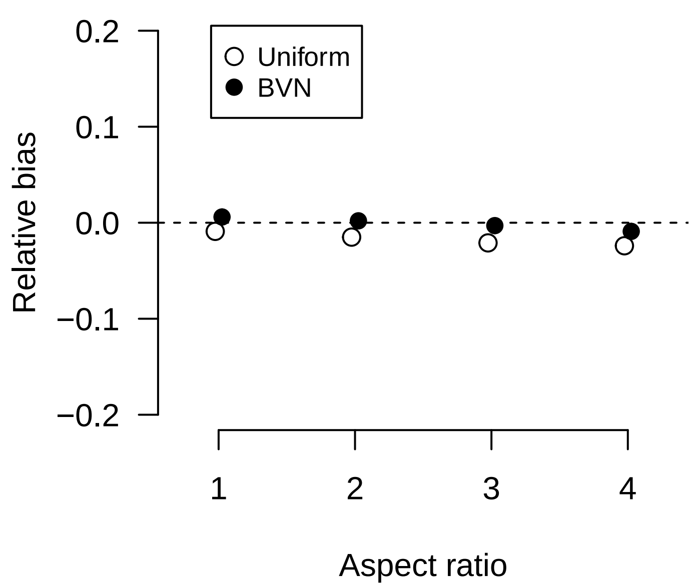
```
## $Uniform
##                  1       2       3       4     1-4     1,4
## av.npop     169.000 169.000 169.000 169.000 169.000 169.000
## av.nCH       49.310  50.536  52.198  53.946  50.890  51.102
## nvalid      500.000 500.000 500.000 500.000 500.000 500.000
## av.parmhat    3.986   3.972   3.954   3.963   3.912   3.927
## md.parmhat    3.979   3.951   3.939   3.943   3.901   3.937
## sd.parmhat    0.497   0.497   0.490   0.501   0.486   0.466
## RB           -0.004  -0.007  -0.011  -0.009  -0.022  -0.018
## seRB          0.006   0.006   0.005   0.006   0.005   0.005
## RSE           0.122   0.120   0.118   0.117   0.120   0.120
## seRSE         0.000   0.000   0.000   0.000   0.000   0.000
## rRMSE         0.124   0.124   0.123   0.126   0.123   0.118
## COV           0.944   0.942   0.952   0.928   0.942   0.956
##
## $BVN
##                  1       2       3       4     1-4     1,4
## av.npop     169.000 169.000 169.000 169.000 169.000 169.000
## av.nCH       42.184  42.942  43.950  44.704  42.994  32.224
## nvalid      500.000 500.000 500.000 500.000 500.000 500.000
## av.parmhat    4.081   4.076   4.074   4.052   4.019   4.112
## md.parmhat    4.051   4.067   4.067   4.028   4.018   4.027
## sd.parmhat    0.593   0.581   0.569   0.565   0.573   0.837
## RB            0.020   0.019   0.019   0.013   0.005   0.028
## seRB          0.007   0.006   0.006   0.006   0.006   0.009
## RSE           0.140   0.139   0.139   0.139   0.139   0.202
## seRSE         0.000   0.000   0.000   0.000   0.000   0.001
```

```
## rRMSE        0.149   0.146   0.143   0.142   0.143   0.211
## COV          0.940   0.934   0.946   0.948   0.946   0.942
```

Common random orientation

```r
tr <- make.grid(10, 10, spacing = 50, detector = 'proximity')
simrandom100C.2 <- vector('list', 4)
names(simrandom100C.2) <- 1:4
simrandomBVN100C.2 <- simrandom100C.2
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 theta = -1, D = 4, CL = TRUE, detectfn = 'HHN', details = details)
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    args$type <- 'uniform'; args$g0 <- 0.2
    simrandom100C.2[[i]] <- do.call(runEllipseSim, args)
    args$type <- 'BVN'; args$lambda0 <- 0.4
    simrandomBVN100C.2[[i]] <- do.call(runEllipseSim, args)
}
save(simrandom100C.2, file = paste0(simfolder, 'simrandom100C.2.RData'))
save(simrandomBVN100C.2, file = paste0(simfolder, 'simrandomBVN100C.2.RData'))
```

```r
load(file = paste0(simfolder, 'simrandom100C.2.RData'))
load(file = paste0(simfolder, 'simrandomBVN100C.2.RData'))
simplot(list(Uniform = simrandom100C.2, BVN = simrandomBVN100C.2), legend = TRUE)
```

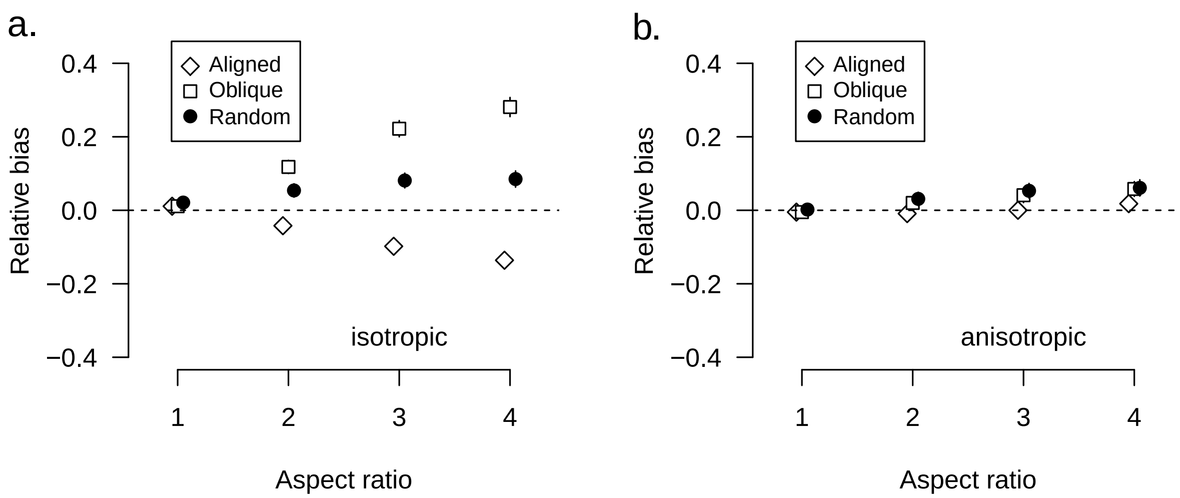


Anisotropic model for data from hollow square array

Using function `anisotropic.fit`. Extract the fitted coefficients corresponding to `psiA` and `psiR` with the `coef` method for secr objects, and rely on direct estimation of density (CL = FALSE).

```
tr <- make.grid(10, 10, spacing = 25, detector = 'proximity', hollow = TRUE)
simaniso1 <- vector('list', 4); names(simaniso1) <- 1:4
simaniso4 <- simaniso3 <- simaniso2 <- simaniso1
baseargs <- list(nrepl = nrepl, buffer = 200, ncores = ncores, traps = tr,
                 lambda0 = 0.4, D = 4, type = 'BVN', CL = FALSE, detectfn = 'HHN',
                 details = list(distribution = 'binomial'), extractfn = coef,
                 secrfn = 'anisotropic.fit')
for (i in 1:4) {
    sigmaX <- 25/i^0.5; sigmaY <- 25*i^0.5
    args <- c(baseargs, list(sigmaX = sigmaX, sigmaY = sigmaY))
    args$theta <- 0     # parallel to detectors
    simaniso1[[i]] <- do.call(runEllipseSim, args)
    args$theta <- pi/4  # oblique to detectors
    simaniso2[[i]] <- do.call(runEllipseSim, args)
    args$theta <- NULL  # random orientation
    simaniso3[[i]] <- do.call(runEllipseSim, args)
    args$theta <- -1    # common random orientation
    simaniso4[[i]] <- do.call(runEllipseSim, args)
}
save(simaniso1, file = paste0(simfolder, 'simaniso1.RData'))
save(simaniso2, file = paste0(simfolder, 'simaniso2.RData'))
save(simaniso3, file = paste0(simfolder, 'simaniso3.RData'))
save(simaniso4, file = paste0(simfolder, 'simaniso4.RData'))
```

```
load(file = paste0(simfolder, 'simgridalignsq1.2.RData'))
load(file = paste0(simfolder, 'simgridalignsq2.2.RData'))
load(file = paste0(simfolder, 'simgridalignsq3.2.RData'))
load(file = paste0(simfolder, 'simaniso1.RData'))
load(file = paste0(simfolder, 'simaniso2.RData'))
load(file = paste0(simfolder, 'simaniso3.RData'))
load(file = paste0(simfolder, 'simaniso4.RData'))
par(mfrow = c(1,2))
simplot(list(Aligned = simgridalignsq1.2[1:4],
             Oblique = simgridalignsq2.2[1:4],
             Random = simgridalignsq3.2[1:4]),
        ylim = c(-0.4,0.45), legend = TRUE, pchi = c(23, 22, 16))
text(-0.4, 0.5, 'a.', cex = 1.45, xpd = TRUE)
text(3, -0.35, 'isotropic')
simplot(list(Aligned = simaniso1,
             Oblique = simaniso2,
             Random = simaniso3),
        component = "pred",
        ylim = c(-0.4,0.45), legend = TRUE, pchi = c(23, 22, 16))
text(-0.4, 0.5, 'b.', cex = 1.45, xpd = TRUE)
text(3, -0.35, 'anisotropic')
```



Tabular summary for anisotropic model.

```
simsum(list(Aligned = simaniso1,
             Oblique = simaniso2,
             Random = simaniso3),
        component = "pred")
```

```
## $Aligned
##    av.nCH     RB   RSE rRMSE   COV
## 1 34.428 -0.005 0.184 0.173 0.962
## 2 35.188 -0.009 0.186 0.168 0.970
## 3 36.572  0.000 0.186 0.165 0.968
## 4 37.898  0.018 0.182 0.171 0.970
##
## $Oblique
##    av.nCH     RB   RSE rRMSE   COV
## 1 34.428 -0.005 0.184 0.173 0.962
## 2 36.468  0.020 0.196 0.191 0.960
## 3 39.392  0.041 0.200 0.221 0.936
## 4 41.842  0.058 0.197 0.228 0.938
##
## $Random
##    av.nCH    RB   RSE rRMSE   COV
## 1 34.626 0.002 0.183 0.184 0.956
## 2 36.086 0.031 0.182 0.198 0.934
## 3 38.148 0.053 0.178 0.234 0.878
## 4 40.000 0.061 0.175 0.248 0.880
```

Tabular comparison of isotropic and anisotropic models for hollow grid.

```
tab1 <- simsum(list(Aligned = simgridalignsq1.2[1:4],
             Oblique = simgridalignsq2.2[1:4],
             Random = simgridalignsq3.2[1:4]))  # find D-hat in the 'fit' component of the output
tab2 <- simsum(list(Aligned = simaniso1,
             Oblique = simaniso2,
```

```r
             Random = simaniso3),
        component = "pred")                            # find D-hat in the 'pred' component of the output
fn <- function(t1,t2) cbind(t1[,-1],t2[,-1])
mapply(fn, tab1, tab2, SIMPLIFY = FALSE)
```

```
## $Aligned
##       RB   RSE rRMSE   COV     RB   RSE rRMSE   COV
## 1  0.011 0.173 0.173 0.948 -0.005 0.184 0.173 0.962
## 2 -0.042 0.170 0.184 0.904 -0.009 0.186 0.168 0.970
## 3 -0.098 0.166 0.205 0.846  0.000 0.186 0.165 0.968
## 4 -0.136 0.162 0.239 0.754  0.018 0.182 0.171 0.970
##
## $Oblique
##      RB   RSE rRMSE   COV     RB   RSE rRMSE   COV
## 1 0.011 0.173 0.173 0.948 -0.005 0.184 0.173 0.962
## 2 0.118 0.170 0.229 0.884  0.020 0.196 0.191 0.960
## 3 0.222 0.166 0.337 0.728  0.041 0.200 0.221 0.936
## 4 0.281 0.163 0.402 0.636  0.058 0.197 0.228 0.938
##
## $Random
##      RB   RSE rRMSE   COV    RB   RSE rRMSE   COV
## 1 0.021 0.172 0.182 0.938 0.002 0.183 0.184 0.956
## 2 0.054 0.170 0.200 0.924 0.031 0.182 0.198 0.934
## 3 0.081 0.166 0.241 0.854 0.053 0.178 0.234 0.878
## 4 0.085 0.163 0.254 0.854 0.061 0.175 0.248 0.880
```