

# Package: bivrp (via r-universe)

September 17, 2024

**Type** Package

**Title** Bivariate Residual Plots with Simulation Polygons

**Version** 1.2-2

**Date** 2020-04-01

**Author** Rafael de Andrade Moral [aut, cre], John Hinde [aut]

**Maintainer** Rafael de Andrade Moral <rafael.deandrademoral@mu.ie>

**Depends** R (>= 3.0.0), MASS (>= 7.3-35), methods, graphics, stats

**Suggests** mvtnorm (>= 1.0-3), mrfDepth (>= 1.0.10)

**Description** Generates bivariate residual plots with simulation polygons for any diagnostics and bivariate model from which functions to extract the desired diagnostics, simulate new data and refit the models are available.

**License** GPL (>= 2)

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-04-01 13:00:02 UTC

## Contents

bivrp-package . . . . .	2
bivrp . . . . .	3
is_point_inside . . . . .	7
plot.bivrp . . . . .	8
polygon-operations . . . . .	9

<b>Index</b>	<b>12</b>
--------------	-----------

bivrp-package

*Bivariate Residual Plots with Simulation Polygons***Description**

Generates bivariate residual plots with simulation polygons for any diagnostics and bivariate model from which functions to extract the desired diagnostics, simulate new data and refit the models are available.

**Details**

Package: bivrp  
 Type: Package  
 Title: Bivariate Residual Plots with Simulation Polygons  
 Version: 1.2-2  
 Date: 2020-04-01  
 Authors@R: c(person("Rafael", "de Andrade Moral", role = c("aut", "cre"), email = "rafael.deandrademoral@mu.ie"), person("John", "Hinde", role = "aut", email = "john.hinde@mu.ie"))  
 Author: Rafael de Andrade Moral [aut, cre], John Hinde [aut]  
 Maintainer: Rafael de Andrade Moral <rafael.deandrademoral@mu.ie>  
 Depends: R (>= 3.0.0), MASS (>= 7.3-35), methods, graphics, stats  
 Suggests: mvtnorm (>= 1.0-3), mrfDepth (>= 1.0.10)  
 Description: Generates bivariate residual plots with simulation polygons for any diagnostics and bivariate model from which functions to extract the desired diagnostics, simulate new data and refit the models are available.  
 License: GPL (>=2)

Index of help topics:

bivrp	Bivariate Residual Plots with Simulation Polygons
bivrp-package	Bivariate Residual Plots with Simulation Polygons
is_point_inside	Determine if point is inside or outside a simple polygon area
plot.bivrp	Plot Method for bivrp Objects
polygon_area	Polygon operations

**Author(s)**

Rafael de Andrade Moral [aut, cre], John Hinde [aut]

Maintainer: Rafael de Andrade Moral <rafael.deandrademoral@mu.ie>

**Description**

Produces a bivariate residual plot with simulation polygons to assess goodness-of-fit of bivariate statistical models, provided the user supplies three functions: one to obtain model diagnostics, one to simulate data from a fitted model object, and one to refit the model to simulated data.

**Usage**

```
bivrp(obj, sim = 99, conf = .95, diagfun, simfun, fitfun, verb = FALSE,
      sort.res = TRUE, closest.angle = TRUE, angle.ref = - pi,
      counter.clockwise = TRUE, xlab, ylab, main,
      clear.device = FALSE, point.col, point.pch, ...)
```

```
## S3 method for class 'bivrp'
print(x, ...)
```

**Arguments**

<code>obj</code>	fitted model object
<code>sim</code>	number of simulations used to compute envelope. Default is 99
<code>conf</code>	confidence level of the simulated polygons. Default is 0.95
<code>diagfun</code>	user-defined function used to obtain the diagnostic measures from the fitted model object
<code>simfun</code>	user-defined function used to simulate a random sample from the model estimated parameters
<code>fitfun</code>	user-defined function used to re-fit the model to simulated data
<code>verb</code>	logical. If TRUE, prints each step of the simulation procedure
<code>sort.res</code>	logical. If TRUE, points will be sorted using angles formed with the origin (type of ordering can be fine-tuned with arguments <code>closest.angle</code> , <code>angle.ref</code> and <code>counter.clockwise</code> ).
<code>closest.angle</code>	logical. If FALSE, points will be sorted starting from the angle defined in <code>angle.ref</code> , if TRUE, points will be sorted starting from the closest angle to the observed bivariate sample ranked as first
<code>angle.ref</code>	the reference angle from which points will be sorted starting from the closest angle to the input (in radians). Defaults to $-\pi$
<code>counter.clockwise</code>	logical. Should the points be ordered counter-clockwise or clockwise from the reference angle?
<code>xlab</code>	argument passed to <code>par</code>
<code>ylab</code>	argument passed to <code>par</code>

<code>main</code>	argument passed to <code>par</code>
<code>clear.device</code>	logical. If TRUE, clears the plotting device after producing the bivariate residual plot with simulation polygons
<code>point.col</code>	a vector of length 2 with the colors of the points that are inside and outside of the simulated polygons
<code>point.pch</code>	a vector of length 2 with the point characters of the points that are inside and outside of the simulated polygons
<code>...</code>	further arguments passed to <code>plot.bivrp</code>
<code>x</code>	an object of class <code>bivrp</code>

### Details

This approach relies on the same strategy used for producing half-normal plots with simulation envelopes. Given a vector of bivariate model diagnostics, the angle each point makes with the origin is calculated to order them. This can be fine-tuned using the logical arguments `closest.angle`, `angle.ref`, and `counter.clockwise`, see the Arguments section above.

Then, `sim` bivariate response variables are simulated from the fitted model, using the same model matrices, error distribution and fitted parameters, using the function defined as `simfun`. The model is refitted to each simulated sample, using the function defined as `fitfun`. Next, we obtain the same type of model diagnostics, using `diagfun`, again ordered the same way the original bivariate sample was. We have, for each bivariate diagnostic, `sim` simulated bivariate diagnostics forming the whole cloud of simulated diagnostics.

By default, we then obtain the convex hulls of each set of the `sim` sets of points and obtain a reduced polygon whose area is  $(\text{conf} * 100)\%$  of the original convex hull's area, forming the simulated polygon. This is equivalent to passing the argument `reduce.polygon = "proportional"` to `plot.bivrp`. The argument `reduce.polygon = "bag"` can be used to obtain a  $(\text{conf} * 100)\%$  bagplot as the simulated polygon instead of a convex hull. The points are then connected to the centroids of their respective simulated polygons and, if they lie outside the polygons, they are drawn in red. For the final display, the polygons are erased so as to ease visualization.

There is no automatic implementation of a bivariate model in this function, and hence users must provide three functions for `bivrp`. The first function, `diagfun`, must extract the desired model diagnostics from a model fit object. The second function, `simfun`, must return the response variable, simulated using the same error distributions and estimated parameters from the fitted model. The third and final function, `fitfun`, must return a fitted model object. See the Examples section.

This function produces a plot by passing the computed object to `plot.bivrp`. The print method returns a `data.frame` containing all ordered simulated bivariate diagnostics.

### Value

The function returns an object of class "bivrp", which is a list containing the following components:

<code>reslist.ord</code>	list of ordered diagnostics from model refitting to each simulated dataset
<code>res.original.ord</code>	original model diagnostics
<code>res1</code>	diagnostics from variable 1
<code>res2</code>	diagnostics from variable 2

res.original1 original model diagnostics for variable 1  
 res.original2 original model diagnostics for variable 2  
 conf confidence level of the simulated polygons

### Author(s)

Rafael A. Moral <rafael.deandrademoral@mu.ie> and John Hinde

### See Also

[plot.bivrp](#)

### Examples

```
## simulating a bivariate normal response variable

require(mvtnorm)

n <- 40
beta1 <- c(2, .4)
beta2 <- c(.2, .2)
x <- seq(1, 10, length = n)
X <- model.matrix(~ x)
mu1 <- X%%beta1
mu2 <- X%%beta2
sig1 <- 2
sig2 <- 3
sig12 <- -1.7
Sig1 <- diag(rep(sig1), n)
Sig2 <- diag(rep(sig2), n)
Sig12 <- diag(rep(sig12), n)
V <- rbind(cbind(Sig1, Sig12),
           cbind(Sig12, Sig2))

set.seed(2016)
Y <- as.numeric(rmvnorm(1, c(mu1, mu2), V))

## code for fitting the model estimating covariance or not
bivnormfit <- function(Y, X, covariance) {
  n <- nrow(X)
  p <- ncol(X)
  y <- cbind(Y[1:n], Y[(n+1):(2*n)])
  XtXinv <- solve(crossprod(X, X))
  beta.hat <- XtXinv %% crossprod(X, y)
  mu.hat <- X%%beta.hat
  sigma.hat <- 1/n * t(y - mu.hat) %% (y - mu.hat)
  if(!covariance) sigma.hat <- diag(diag(sigma.hat))
  cov.betas <- sigma.hat %x% XtXinv
  se.s1 <- sqrt(2*sigma.hat[1]^2/(n-p+1))
  se.s2 <- sqrt(2*sigma.hat[4]^2/(n-p+1))
  if(!covariance) se.s12 <- NA else {
    rho <- sigma.hat[2]/sqrt(sigma.hat[1]*sigma.hat[4])
  }
}
```

```

    se.s12 <- sqrt((1+rho^2)*sigma.hat[1]*sigma.hat[4]/(n-p+1))
  }
  se.betas <- sqrt(diag(cov.betas))
  se.sigma <- c(se.s1, se.s2, se.s12)
  coefs <- c(beta.hat, sigma.hat[1], sigma.hat[4], sigma.hat[2])
  names(coefs) <- c("beta1.0", "beta1.1", "beta2.0", "beta2.1", "sig1", "sig2", "sig12")
  fitted <- c(mu.hat)
  resid <- Y - fitted
  Sig1 <- diag(rep(sigma.hat[1]), n)
  Sig2 <- diag(rep(sigma.hat[4]), n)
  Sig12 <- diag(rep(sigma.hat[2]), n)
  V <- rbind(cbind(Sig1, Sig12),
             cbind(Sig12, Sig2))
  llik <- dmvnorm(Y, c(mu.hat), V, log = TRUE)
  ret <- list("coefs" = coefs, "covariance" = covariance, "n" = n,
            "X" = X, "fitted" = fitted, "resid" = resid, "loglik" = llik,
            "Y" = Y, "se" = c(se.betas, se.sigma))
  class(ret) <- "bivnormfit"
  return(ret)
}

## fitting bivariate models with and without estimating covariance
fit0 <- bivnormfit(Y, X, covariance=FALSE)
fit1 <- bivnormfit(Y, X, covariance=TRUE)
## likelihood-ratio test
2*(fit0$loglik - fit1$loglik)
pchisq(54.24, 1, lower=FALSE)

## function for extracting diagnostics (raw residuals)
dfun <- function(obj) {
  r <- obj$resid
  n <- obj$n
  return(list(r[1:n], r[(n+1):(2*n)]))
}

## function for simulating new response variables
sfun <- function(obj) {
  n <- obj$n
  fitted <- obj$fitted
  sig1 <- obj$coefs[5]
  sig2 <- obj$coefs[6]
  if(obj$covariance) sig12 <- obj$coefs[7] else sig12 <- 0
  Sig1 <- diag(rep(sig1), n)
  Sig2 <- diag(rep(sig2), n)
  Sig12 <- diag(rep(sig12), n)
  V <- rbind(cbind(Sig1, Sig12),
             cbind(Sig12, Sig2))
  Y <- as.numeric(rmvnorm(1, c(mu1, mu2), V))
  return(list(Y[1:n], Y[(n+1):(2*n)], "X" = obj$X,
            "covariance" = obj$covariance))
}

## function for refitting the model to simulated data

```

```
ffun <- function(new.obj) {
  Ynew <- c(new.obj[[1]], new.obj[[2]])
  bivnormfit(Ynew, new.obj$X, new.obj$covariance)
}

## Bivariate residual plot for model 1 (without estimating covariance)
plot1 <- bivrp(fit0, diagfun=dfun, simfun=sfun, fitfun=ffun, verb=TRUE)
## without polygon area reduction
plot(plot1, conf=1)
## drawing polygons
plot(plot1, add.polygon=TRUE)
## without ordering
plot(plot1, theta.sort=FALSE, kernel=TRUE, add.dplots=TRUE, superpose=TRUE)

## Bivariate residual plot for model 2 (estimating covariance)
plot2 <- bivrp(fit1, diagfun=dfun, simfun=sfun, fitfun=ffun, verb=TRUE)
## without polygon area reduction
plot(plot2, conf=1)
## drawing polygons
plot(plot2, add.polygon=TRUE, conf=1)
## without ordering
plot(plot2, theta.sort=FALSE, kernel=TRUE, add.dplots=TRUE, superpose=TRUE)
```

---

is\_point\_inside

*Determine if point is inside or outside a simple polygon area*

---

## Description

Returns whether a point is inside or outside the convex polygon formed with the coordinates in a data frame or matrix

## Usage

```
is_point_inside(point, polyg)
```

## Arguments

point	vector of two values for a point in the Cartesian plane
polyg	data frame or matrix with the coordinates forming the convex polygon

## Details

The algorithm used here draws a ray from the point and counts the number of intersections made with the polygon. If the number of intersections is only one, then this means the point is inside the convex polygon.

## Value

This function returns TRUE, if the point is inside and FALSE, otherwise.

**Author(s)**

Rafael A. Moral <rafael.deandrademoral@mu.ie> and John Hinde

**Examples**

```
my_polygon <- data.frame(c(1, 2, 3, 4, 3),
                        c(1, 0, .5, 3, 4))
points_to_test <- list(c(0, 0), c(2.5, 1), c(3.5, 4))

unlist(lapply(points_to_test, is_point_inside, my_polygon))
```

---

plot.bivrp

*Plot Method for bivrp Objects*


---

**Description**

Plots the bivariate residual plot with simulation polygons from a bivrp object

**Usage**

```
## S3 method for class 'bivrp'
plot(x, kernel, superpose.points, chp, add.dplots,
     theta.sort, add.polygon, reduce.polygon, one.dim, pch = 16, cex = 0.8,
     conf, xlab, ylab, main, point.col, point.pch, transparent.colors,
     density.bw, ...)
```

**Arguments**

x	object of class bivrp
kernel	logical. If TRUE, instead of using polygons for each point, computes 2d kernels and plots the contours
superpose.points	only used if kernel or chp is TRUE. Logical argument, if TRUE, plots all simulated bivariate diagnostics
chp	logical. If TRUE, instead of using polygons for each point, performs convex hull peeling over all simulated points
add.dplots	logical. If TRUE, adds the marginal density plots
theta.sort	logical. If TRUE, produces a simulated polygon for each point
add.polygon	logical. If TRUE, plots the simulated polygons as well
reduce.polygon	method used to reduce the polygon area. Defaults to proportional, see <a href="#">get_newpolygon</a> for details. If reduce.polygon = "peel", performs convex hull peeling to reduce the area; if reduce.polygon = "bag", computes a (conf * 100)% bagplot of the points



one.dim	logical. If TRUE, plots only the marginal density plots (only works with theta.sort = FALSE)
pch	argument passed to par
cex	argument passed to par
conf	confidence level of the simulated polygons. Default is 0.95
xlab	argument passed to par
ylab	argument passed to par
main	argument passed to par
point.col	a vector of length 2 with the colors of the points that are inside and outside of the simulated polygons
point.pch	a vector of length 2 with the point characters of the points that are inside and outside of the simulated polygons
transparent.colors	logical. If TRUE, adds transparency to the marginal density plots; if FALSE, only the border lines are drawn
density.bw	the smoothing bandwidth to be used for the marginal densities. Defaults to "SJ" (see <a href="#">density</a> )
...	further arguments passed to par

**Author(s)**

Rafael A. Moral <rafael\_moral@yahoo.com.br> and John Hinde

**See Also**

[bivrp](#)

---

`polygon-operations`      *Polygon operations*

---

**Description**

Convex polygon operations - determination of area, centre of mass, and area reduction

**Usage**

```
polygon_area(P)
get_k(P, conf)
get_newpolygon(conf, P, method)
get_reduced_bag(x, y, conf)
compute_bagplot(x, y, conf)
```

**Arguments**

P	2-column matrix or data.frame with the coordinates of the vertices of the convex polygon
conf	proportion of the area of polygon P
method	method used to reduce the area of the polygon. Use method = "proportional" to scale the distances between the centroid and the vertices by $\sqrt{\text{conf}}$ ; use method = "get_k" to subtract the same distance k from the centroid to each vertex.
x	x coordinates (of raw data) used to obtain the reduced bag
y	y coordinates (of raw data) used to obtain the reduced bag

**Details**

The function `compute_bagplot` uses an adapted version of the code written by P. Segaert to obtain the bagplot, that uses the Fortran subroutine written by P.J. Rousseeuw, I. Ruts and A. Struyf.

**Author(s)**

Rafael A. Moral <rafael.deandrademoral@mu.ie> and John Hinde

**References**

Rousseeuw P.J., Ruts I., Tukey J.W. (1999). The bagplot: A bivariate boxplot. *The American Statistician*, 53, 382–387.

**See Also**

[is\\_point\\_inside polygon](#)

**Examples**

```
oldPolygon <- data.frame(x=c(2,1,3,4.5,5), y=c(1,3,5,4.5,2))

# area
polygon_area(oldPolygon)$area
# centre of mass
polygon_area(oldPolygon)$centre

# get a new polygon with 50% of the area of the old one
newPolygon <- get_newpolygon(conf=.5, P=oldPolygon, method="proportional")
polygon_area(newPolygon)$area/polygon_area(oldPolygon)$area

# second method
newPolygon2 <- get_newpolygon(conf=.5, P=oldPolygon, method="get.k")
polygon_area(newPolygon2)$area/polygon_area(oldPolygon)$area

# illustration
plot(oldPolygon, xlim=c(0,6), ylim=c(0,6), main="(a)", pch=16)
polygon(oldPolygon, lwd=2, col="#00000033")
```

```
text(oldPolygon, c(expression(P[1]), expression(P[2]),
                    expression(P[3]), expression(P[4]),
                    expression(P[5])), pos=c(1,2,3,4,4), cex=2)
polygon(newPolygon, border=4, lwd=2, col="#52A3E199")
points(newPolygon, pch=16, col=4)
text(newPolygon, c(expression(paste(P[1],minute)), expression(paste(P[2],minute)),
                    expression(paste(P[3],minute)), expression(paste(P[4],minute)),
                    expression(paste(P[5],minute))), pos=c(1,3,2,4,4), col=4, cex=2)

C <- polygon_area(oldPolygon)$centre
text(C[1], C[2], "C", pos=4, cex=2)
for(i in 1:5) lines(c(C[1], oldPolygon[i,1]),
                   c(C[2], oldPolygon[i,2]), lty=2, lwd=2, type="b")
```

# Index

\* **package**

bivrp-package, 2

\* **polygon**

polygon-operations, 9

bivrp, 3, 9

bivrp-package, 2

compute\_bagplot (polygon-operations), 9

density, 9

get\_k (polygon-operations), 9

get\_newpolygon, 8

get\_newpolygon (polygon-operations), 9

get\_reduced\_bag (polygon-operations), 9

is\_point\_inside, 7, 10

plot.bivrp, 4, 5, 8

polygon, 10

polygon-operations, 9

polygon\_area (polygon-operations), 9

print.bivrp (bivrp), 3