

# Package ‘RCALI’

May 7, 2026

**Title** Calculation of the Integrated Flow of Particles Between Polygons

**Version** 0.3.7

**Date** 2024-09-23

**Encoding** UTF-8

**Author** Annie Bouvier [aut, cph],  
Kien Kieu [aut, cph],  
Kasia Adamczyk [aut, cph],  
Herve Monod [aut, cph],  
Jean-Francois Rey [cre]

**Maintainer** Jean-Francois Rey <jean-francois.rey@inrae.fr>

**Description** Calculate the flow of particles between polygons by two integration methods:  
integration by a cubature method and integration on a grid of points.  
Annie Bouvier, Kien Kieu, Kasia Adamczyk and Herve Monod (2009)  
<doi:10.1016/j.envsoft.2008.11.006>.

**License** GPL-3

**URL** <https://gitlab.paca.inrae.fr/biosp/RCALI>

**Imports** splancs, graphics, grDevices, methods

**NeedsCompilation** yes

**RoxygenNote** 7.3.1

**Repository** CRAN

**Date/Publication** 2024-10-22 10:00:06 UTC

## Contents

RCALI-package . . . . .	2
as.poly . . . . .	3
califlopp . . . . .	3
crlistpoly . . . . .	7
crpoly . . . . .	8
export . . . . .	9
export.default . . . . .	9

export.listpoly . . . . .	10
fpollen . . . . .	10
fseed . . . . .	11
generPoly . . . . .	12
generVois . . . . .	13
getRes . . . . .	13
listpoly-class . . . . .	14
plot.listpoly . . . . .	15
plot.poly . . . . .	16
poly-class . . . . .	16
range.listpoly . . . . .	17
readpoly1 . . . . .	17
readpoly2 . . . . .	18
<b>Index</b>	<b>19</b>

---

RCALI-package

---

*Calculation of the Integrated Flow of Particles Between Polygons*


---

## Description

Calculate the flow of particles between polygons by two integration methods: integration by a cubature method and integration on a grid of points.

## Details

Depends:

splancs

License:

GPL version 2 or later

This package is an interface between R and the programme CaliFloPP:

<http://genome.jouy.inra.fr/logiciels/califlopp/>

News

`file.show(system.file("NEWS", package="RCALI-`

Main function:

`califlopp`

## Author(s)

Annie Bouvier <annie.bouvier@inra.fr>

---

as.poly	<i>Create an object of class 'poly'</i>
---------	---

---

**Description**

Create an object of class "[poly](#)" from two vectors or from a matrix.

**Usage**

```
as.poly(x, y=NULL)
```

**Arguments**

x	vector of x-coordinates or two-columns matrix
y	vector of y-coordinates when 'x' is a vector.

**Value**

An object of class "[poly](#)", i.e a two-columns matrix labelled "xcoord", "ycoord".

**See Also**

[crpoly](#)

**Examples**

```
# A triangle  
a <- as.poly(matrix(c(2,2,2,3,3,3), ncol=2, byrow=TRUE))
```

---

califlopp	<i>Calculation of the Integrated Flow of Particles between Polygons</i>
-----------	---

---

**Description**

Calculation of the flow of particles between polygons by two integration methods: integration by a cubature method and integration on a grid of points.

**Usage**

```
califlopp(file, dispf=c(1,2), param=NULL, resfile = NULL)
```

## Arguments

file	Pathname of the polygons-file. See details.
dispf	The required dispersion functions. Vector of integers or vector of R functions. The maximum length of this vector is 5. <ul style="list-style-type: none"> <li>• If vector of integers, the dispersion functions are then compiled. Five are provided. To modify them, see Details. By default, 1 is for dispersal of oilseed rape pollen, 2 for dispersal of oilseed rape seed (dispersals of oilseed rape are the ones defined in <i>GeneSys</i> - see References), 3 for the constant function, 4 for an anisotropic version of the dispersal of yellow rust of wheat defined in <i>Soubeyrand and all</i>, 5 for a discontinuous function. These functions are viewable in the C file <code>src/functions.cc</code>. To modify them, see Details.</li> <li>• If vector of functions, the dispersion functions are coded in R (more time consuming than compiled version). Two R dispersal functions are provided, <code>fpollen</code> and <code>fseed</code>, the functions used in Genesys. To specify your own function, see details.</li> </ul>
param	Optional list of parameters. Valid components are <code>input</code> , <code>output</code> , <code>verbose</code> , <code>warn.poly</code> , <code>warn.conv</code> , <code>delim</code> , <code>poly</code> , <code>send.and.receive</code> , <code>method</code> , <code>dz</code> , <code>dp</code> . In addition, when method is "cub": <code>maxpts</code> , <code>reler</code> , <code>abser</code> , <code>tz</code> . When method is "grid": <code>seed</code> , <code>step</code> , <code>nr</code> . See details.
resfile	Optional pathname of a result-file. When set, the results are written on it. This file can be read by using function <code>getRes</code> or <code>read.table</code> . See details, as to the content of the file.

## Details

### The polygons-file

The coordinates of the polygons should be provided in an ASCII file, denoted here "polygons-file". The unit is the meter. The vertices should be ordered clockwise. The polygons can be closed or not, but without holes. The first line contains the number of polygons. The following lines depend of the input parameter:

`input=1` Two lines per polygon: on the first one, an identifier (a positive integer), followed by the x-coordinates, on the second one, the same identifier followed by the y-coordinates. The function `export.listpoly` generates such a file from R structures

`input=2` Three lines per polygon: on the first one, an identifier (a positive integer), followed by a name for the polygon and by the number of its vertices, on the second one, the x-coordinates, and on the third one, the y-coordinates.

### The individual dispersion functions

The individual dispersion functions can either be compiled or R functions.

**Compiled function:** Five compiled dispersal functions are provided (see argument `dispf`). To replace them by yours, you have to download the source of RCALI, modify and compile it.

Suppose that you have download the tar-archive in the directory `MyDir`. The steps to customize the dispersion functions are:

1/ Replace one or several functions in MyDir/RCALI/src/functions.cc by yours: The dispersion function has one argument, the current point, p, of class Point. You can use p.getX() and p.getY() to get the coordinates of the current point (in meters\*SCALE, where SCALE is the rescaling parameter defined in the file src/calicinfig.h), p.dist0(), the distance of p from the origin (in meters\*SCALE) and p.angle0(), the angle (in degrees, in  $[-\pi, +\pi]$ ) between the line (0,p) and the horizontal line.

2/ Create a directory MyDir/RCALI/libs, place you in MyDir/RCALI/src and type in:

```
R CMD SHLIB -o ../libs/RCALI.so *.cc
```

to create the compiled shared library.

3/ To use in a R-session:

```
source("MyDir/RCALI/R/sourceDir.R")
sourceDir("MyDir/RCALI/R")
dyn.load("MyDir/RCALI/libs/RCALI.so")
```

The help-files are viewable by opening in a browser MyDir/RCALI/inst/doc/html/00Index.html

You can also build the tar.gz file again, after modifications, and install it as a library by using the standard R commands R CMD build and R CMD INSTALL --html.

**R function:** Two R dispersal functions are provided, `fpollen` and `fseed`, the functions used in Genesys.

You can define your own R dispersal function: it should have one vector argument, the localization of the current point, p. The first element of this vector is the distance of p from the origin (in meters) and the second one is the angle (in degrees, in  $[-\pi, +\pi]$ ) between the line (0,p) and the horizontal line (i.e, stating x and y are the coordinates of p, the angle is  $atan2(y, x) * \frac{180}{\pi}$ )

### The parameters

The argument param is a list which valid components are:

input format of the polygons-file. 1 or 2 (see above). Default 2

output output required on the screen: 0 nothing, 1: all results, 2: progression numbers, 3: the integrated flows and their means per squared meter. Default 1

verbose TRUE, if output is required about polygons convexity and landscape translation. Default FALSE

warn.poly TRUE, if output is required about polygons simplification. Default FALSE

warn.conv TRUE, if output is required when cubature convergence is not reached. Default TRUE

delim separator character between values in the polygons-file. Default: tabulate

send.and.receive TRUE, if results are required from sending polygons to target polygons and from target polygons to sending polygons (case of anisotropic functions). Default FALSE

poly required pairs of polygons. List of vectors of length 2, or two-columns matrix. If only one pair is required, it may be a vector of length 2. Default: all pairs of polygons.

method string equal to cub for cubature method, grid for the grid method. Default: cub

dz integer vector, whose length is greater or equal to the number of required dispersion functions. dz[i] is the distance in meters beyond which the ist dispersion function is considered as nul. Default in a standard configuration: 0,21,0,1000,0 for functions number 1 to 5, respectively.

`dp` integer vector, whose length is greater or equal to the number of required dispersion functions. `dp[i]` is the distance in meters beyond which the *i*st dispersion function is calculated between centroids only. Default in a standard configuration: 100, 0, 0, 500, 0 for functions number 1 to 5, respectively.

In addition, when method is cub:

`maxpts` maximal number of evaluation points required for each function. Vector of length equal to the number of required functions. Default in a standard configuration: 100000

`reler` relative error required for each function. Vector of length equal to the number of required functions. Should be positive when method is cubature. Default in a standard configuration: 1.0e-3

`abserr` absolute error required for each function. Vector of length equal to the number of required functions. Should be positive when method is cubature. Default in a standard configuration: 1.0e-3

`tz` integer vector, whose length is greater or equal to the number of required dispersion functions. Mode of triangulation for the cubature method. `tz[i]` should be 1, if, for the *i*st dispersion function, triangulation from (0,0) has to be done when (0,0) is included in the integration area and, 0 if not. 1 is recommended when the dispersion function is very "sharp" at the origin. Default in a standard configuration: 0,1,0,0,0 for functions number 1 to 5, respectively.

When method is grid:

`seed` seed of the random generator.

`step` step of the grid on the x-axis and on the y-axis in meter. Vector of length 2.

`nr` maximal number of replications or grids.

### The result-file

When the argument `resfile` is set, a file is created. On this file, the values are separated by tabulates.

Its contains, when the method is cub,

**on the first line:** "npoly:", "input-file:", "nfunc:", "method:", each of these identifiers followed by the actual values.

**on each of the following lines, the results for a couple of polygons:** the identifiers of both polygons; the integrated flow divided by the area of the second polygon, for each dispersal function; the areas of both polygons; then, for each dispersal function, the integrated flow, the lower and upper bounds of the confidence interval, the absolute error, and the number of evaluations.

Its contains, when the method is grid,

**on the first line:** "npoly:", "input-file:", "nfunc:", "method:", "stepx:", "stepy:", each of these identifiers followed by the actual values.

**on each of the following lines, the results for a couple of polygons:** the identifiers of both polygons; the integrated flow divided by the area of the second polygon, for each dispersal function; the areas of both polygons; then, for each dispersal function, the integrated flow, and the standard deviation.

This file can be read in a R-session by using the function `getRes` or `read.table`, with option `skip=1`.

**Value**

Nothing. To store the results, set the argument `resfile`, then use the function `getRes` or `read.table`, with option `skip=1`

**Side effect**

This function creates a temporary file to store the parameters, usually in the directory `tmp` of the user. This file is destroyed at the end of execution.

**Author(s)**

A. Bouvier

**References**

- The CaliFloPP software: <http://genome.jouy.inra.fr/logiciels/califlopp/>
- Main reference paper: A. Bouvier, K. Kieu, K. Adamczyk, and H. Monod. Computation of integrated flow of particles between polygons. *Environmental Modelling & Software*, 24:843–849, 2009.
- N. Colbach, and all. Genesys: a model of the influence of cropping system on gene escape from herbicide tolerant rapeseed crops to rape volunteers. *Agriculture, Ecosystems and Environment*, 83:235–270, 2001.

**See Also**

[getRes](#)

**Examples**

```
# Grid method with compiled constant and seed dispersion functions:
param <- list(method="grid", grid=list(step=c(50,50)))
## Not run: califlopp("MyPolygonsFile",dispf=c(3,1), param=param)

# Cubature method with a R dispersion function:
param <- list(output=1, input=2, dz=0, dp=100, tz=0)
## Not run: califlopp("MyPolygonsFile", dispf=fpollen, param=param)
```

---

crlistpoly

*Create an object of class 'listpoly' from objects of class 'poly'*

---

**Description**

Create an object of class `"listpoly"` from objects of class `"poly"`

**Usage**

```
crlistpoly(...)
```

**Arguments**

... objects of class "[poly](#)"

**Value**

An object of class "[listpoly](#)" : a list where each component is a 'poly' object (see [poly-class](#)).

**See Also**

[poly-class](#)

**Examples**

```
# A triangle:
a <- as.poly(matrix(c(2,2,2,3,3,3), ncol=2, byrow=TRUE))
# A square:
b <- as.poly(matrix(c(2.5,2,2.5,2.5,3,2.5,3,2), ncol=2, byrow=TRUE))
# The both:
z <- crlistpoly(a,b)
```

---

crpoly

*Create an object of class 'poly' by clicking on points*

---

**Description**

Create an object of class "[poly](#)" by clicking on a graphic: the locations of the clicks will be the vertices of the polygon.

**Usage**

```
crpoly()
```

**Details**

The system prompts the user to enter points on the current graphic using the mouse or other pointing device. The points are joined on the screen with the current line symbol. A polygon of the points entered is drawn on the current graphics device and returned as a 'poly' object.

**Value**

An object of class "[poly](#)" i.e a two-columns matrix with the coordinates.

**Warning**

A plot should be drawn on the current graphics device before.

**See Also**

[as.poly](#), [getpoly](#) of the package [splancs](#)

**Examples**

```
## Not run: plot(x=c(1,10), y=c(1,10), type='n')
## Not run: a<-crpoly()
#   Enter points with button 1
#   Finish with button 2
```

---

export	<i>Generic for 'export'</i>
--------	-----------------------------

---

**Description**

Generic function for 'export'

**Usage**

```
export(x, filename)
```

**Arguments**

x	object to export
filename	target filename

**Value**

None.

**See Also**

[export.listpoly](#)

---

export.default	<i>export.default method</i>
----------------	------------------------------

---

**Description**

Default method export

**Usage**

```
## Default S3 method:
export(x, filename)
```

**Arguments**

x	object to export
filename	target filename

**Value**

None.

**See Also**

[export.listpoly](#)

---

<code>export.listpoly</code>	<i>Create a polygons file in format 1.</i>
------------------------------	--

---

**Description**

From an object of class "[listpoly](#)", create a polygons file for input to the software CaliFloPP or to the function [califlopp](#). The format of this file corresponds to the format 1 of them, i.e you should set the param input to 1 when used as input to [califlopp](#).

**Usage**

```
## S3 method for class 'listpoly'
export(x, filename)
```

**Arguments**

<code>x</code>	Object of class " <a href="#">listpoly</a> "
<code>filename</code>	Name of the file to be created.

**Value**

None.

---

<code>fpollen</code>	<i>Individual pollen dispersion function</i>
----------------------	--

---

**Description**

From a vector of distances, calculate the individual pollen dispersion function used in *Genesys*

**Usage**

```
fpollen(point)
```

**Arguments**

<code>point</code>	scalar or vector of length 2, whose first element is a distance expressed in meter.
--------------------	---

**Value**

The calculated dispersions.

**References**

Colbach, N. and Clermont-Dauphin, C. and Meynard, J.M. *Genesys: a model of the influence of cropping system on gene escape from herbicide tolerant rapeseed crops to rape volunteers. i. temporal evolution of a population of rapeseed volunteers in a field.* Agriculture, Ecosystems and Environnement, 83:235-253, 2001.

**See Also**

[fseed](#)

**Examples**

```
distance = seq(1,1.5, by=0.05)
a=matrix(distance, ncol=1)
b= apply(a,1,fpollen)
par(pty="s")
plot(x=distance, y =b)
lines(x=distance, y = apply(a,1,fseed))
```

---

fseed

*Individual seed dispersion function*

---

**Description**

From a vector of distances, calculate the individual seed dispersion function used in *Genesys*

**Usage**

```
fseed(point)
```

**Arguments**

point                    scalar or vector of length 2, whose first element is a distance expressed in meter.

**Value**

The calculated dispersions.

**References**

Colbach, N. and Clermont-Dauphin, C. and Meynard, J.M. *Genesys: a model of the influence of cropping system on gene escape from herbicide tolerant rapeseed crops to rape volunteers. i. temporal evolution of a population of rapeseed volunteers in a field.* Agriculture, Ecosystems and Environnement, 83:235-253, 2001.

**See Also**[fpollen](#)**Examples**

```
distance = seq(1,1.5, by=0.05)
a=matrix(distance, ncol=1)
b= apply(a,1,fseed)
par(pty="s")
plot(x=distance, y =b)
```

---

`generPoly`*Generate a regular grid of polygons*

---

**Description**

Generate a regular grid of polygons and optionally write them on a file for input to [califlopp](#)

**Usage**

```
generPoly(step = 5, np = 10, file = "data", plot = TRUE)
```

**Arguments**

<code>step</code>	Step of grid, in meter.
<code>np</code>	Number of polygons on each axis.
<code>file</code>	Pathname of the polygons file to be created. If NULL, no file created.
<code>plot</code>	If TRUE, plot of the polygons.

**Details**

When `file` is not NULL, a polygons file is created. It is in the format 2 of [califlopp](#) and the values separator is tabulate.

**Value**

An object of class "[listpoly](#)"

**Note**

The polygons are numbered from 1, from the left to the right, then from bottom to top.

**See Also**

["listpoly-class"](#), [plot.listpoly](#)

**Examples**

```
a <- generPoly(np=3, file=NULL)
```

---

generVois	<i>Generate the neighbors of each polygon of a regular grid</i>
-----------	---

---

**Description**

For each polygon of a regular grid, generate the indexes of its neighbour polygons

**Usage**

```
generVois(np = 10, nvois = 1)
```

**Arguments**

np	Number of polygons on each axis of the grid.
nvois	Number of polygons required in the neighbourhood, on each direction, for each polygon

**Details**

For each polygon P of a regular grid made of np polygons along each axis, the indexes of its neighbours are generated: the neighbours are the polygons included in the square made of nvois polygons below, above, at the right and left sides of P, in the limit of the grid bounds.

**Value**

A two-columns matrix, of all pairs of neighbours.

**See Also**

[generPoly](#), [califlopp](#)

**Examples**

```
generVois(np=3)
```

---

getRes	<i>Read a result-file of 'califlopp'</i>
--------	--

---

**Description**

Return the results stored on a file created by califlopp.

**Usage**

```
getRes(ficres)
```

**Arguments**

`ficres` Pathname of a result-file created by [califlopp](#).

**Value**

A data.frame with as many rows as pairs of polygons on the file. The columns are:

<code>poly1, poly2</code>	Identifiers of the polygons.
<code>mean.flow/area</code>	Integrated flow divided by the area of the second polygon. If several dispersion functions have been studied, as many columns as functions. The columns labels are then <code>mean.flow.f1/area</code> , <code>mean.flow.f2/area</code> , etc...
<code>area1, area2</code>	Areas of the polygons in squared meters.
<code>mean.flow, conf.int.lower, conf.int.upper, abs.err, n.eval</code>	when the method is "cubature", only: integrated flow, lower and upper bounds of the confidence interval, absolute error, number of evaluations. If several dispersion functions have been studied, the columns labels are suffixed with ".f1", "f2", etc...
<code>mean.flow, std</code>	when the method is "grid", only: mean of the integrated flow, standard deviation. If several dispersion functions have been studied, the columns labels are suffixed with ".f1", "f2", etc...

**Note**

This function works when RCALI has been configured with `OUTPUT_FILE_FORMAT = LIGHT` only (this is the default; see the file `src/caliconfig.h` to be sure your configuration is compatible).

Details about the returned values can be found in the Reference Manual.

**See Also**

[califlopp](#)

---

<code>listpoly-class</code>	<i>Class 'listpoly'</i>
-----------------------------	-------------------------

---

**Description**

A class to store a suite of polygons and its methods.

**Objects from the Class**

Objects can be created by calls of the function [crlistpoly](#) and [generPoly](#)

**Slots**

**list** List where each component is an object of [poly-class](#)

**Methods**

[plot.listpoly](#), [range.listpoly](#), [export.listpoly](#)

**See Also**

[poly-class](#)

---

plot.listpoly	<i>Plot of an object of class 'listpoly'</i>
---------------	--

---

**Description**

Plot all the polygons of an object of class "[listpoly](#)", optionnaly with colors.

**Usage**

```
## S3 method for class 'listpoly'  
plot(x, add = F, color = T, ...)
```

**Arguments**

x	Object of class " <a href="#">listpoly</a> ".
add	TRUE, to add to a previous plot, for example after a zoom.
color	TRUE, to fill each polygon in a different color.
...	List of arguments passed as-is to <a href="#">plot.poly</a>

**Value**

None.

**Note**

To make a zoom, type in: `zoom()` (see `'splancs'`) then use this function with argument `"add=TRUE"`.

---

<code>plot.poly</code>	<i>Plot of an object of class 'poly'</i>
------------------------	--

---

**Description**

Plot an object of class "[poly](#)" via the function [polymap](#) of the library `splancs`.

**Usage**

```
## S3 method for class 'poly'
plot(x, ...)
```

**Arguments**

<code>x</code>	Object of class " <a href="#">poly</a> ".
<code>...</code>	Arguments passed as-it to ' <a href="#">polymap</a> '.

**Value**

None.

---

<code>poly-class</code>	<i>Class 'poly'</i>
-------------------------	---------------------

---

**Description**

A class to store the coordinates of a polygons and its methods.

**Objects from the Class**

Objects can be created by calls of the functions [as.poly](#) and [crpoly](#)

**Slots**

**m** Matrix with two columns `xcoord` and `ycoord`

**Methods**

[plot.poly](#)

**See Also**

[listpoly-class](#)

---

range.listpoly	<i>Range of the coordinates of an object of class 'listpoly'</i>
----------------	--

---

**Description**

Return the minimum and maximum coordinates over all the polygons of an object of class "[listpoly](#)".

**Usage**

```
## S3 method for class 'listpoly'  
range(x, ...)
```

**Arguments**

x	an object of class " <a href="#">listpoly</a> "
...	arguments passed as-it to the R function range

**Value**

Matrix ; its two columns are "xrange" and "yrange" and its two lines are "lower" and "upper".

---

readpoly1	<i>Read a polygons file in format 1</i>
-----------	---

---

**Description**

Create an object of class "[listpoly](#)" from the values read on a polygons file in format 1.

**Usage**

```
readpoly1(filename, delim=" ")
```

**Arguments**

filename	Name of the polygons file
delim	Separator character on the file.

**Value**

An object of class "[listpoly](#)", i.e a list, each component of which is an object of class "[poly](#)"

**See Also**

[readpoly2](#)

---

`readpoly2`*Read a polygons file in format 2*

---

**Description**

Create an object of class "[listpoly](#)" from the values read on a polygons file in format 2.

**Usage**

```
readpoly2(filename, delim="\t")
```

**Arguments**

<code>filename</code>	Name of the polygons file
<code>delim</code>	Separator character on the file.

**Value**

An object of class "[listpoly](#)", i.e a list, each component of which is an object of class "[poly](#)"

**See Also**

[readpoly1](#)

# Index

- \* **IO**
    - export, 9
    - export.default, 9
    - export.listpoly, 10
    - readpoly1, 17
    - readpoly2, 18
  - \* **classes**
    - listpoly-class, 14
    - poly-class, 16
  - \* **data**
    - crlistpoly, 7
    - crpoly, 8
    - generPoly, 12
    - generVois, 13
    - getRes, 13
  - \* **hplot**
    - crpoly, 8
    - generPoly, 12
    - plot.listpoly, 15
    - plot.poly, 16
  - \* **manip**
    - as.poly, 3
    - fpollen, 10
    - fseed, 11
    - range.listpoly, 17
  - \* **optimize**
    - califlopp, 3
  - \* **package**
    - RCALI-package, 2
- as.poly, 3, 8, 16
- califlopp, 2, 3, 10, 12–14
- crlistpoly, 7, 14
- crpoly, 3, 8, 16
- export, 9
- export.default, 9
- export.listpoly, 4, 9, 10, 10, 15
- fpollen, 4, 5, 10, 12
- fseed, 4, 5, 11, 11
- generPoly, 12, 13, 14
- generVois, 13
- getRes, 4, 6, 7, 13
- listpoly, 7, 8, 10, 12, 15, 17, 18
- listpoly-class, 14
- plot.listpoly, 12, 15, 15
- plot.poly, 15, 16, 16
- poly, 3, 7, 8, 16–18
- poly-class, 16
- polymap, 16
- range.listpoly, 15, 17
- RCALI (RCALI-package), 2
- RCALI-package, 2
- read.table, 4, 6, 7
- readpoly1, 17, 18
- readpoly2, 17, 18