INRA
UR341 Mathématiques et informatique appliquées
Domaine de Vilvert
F-78350 Jouy-en-Josas

RCALI

Version 0.2-0; 2011-11-08

R-package to Calculate the Integrated Flow of Particles between Polygons

Annie Bouvier

January 30, 2012

Abstract

RCALI [1] is a R package that makes the interface between the CaliFloPP[2] and R.

`CaliFloPP` is a software that calculates flows of particles between pairs of polygons, when given a so-called individual dispersal function. The individual dispersal function describes the particle dispersion between pairs of points, and `CaliFloPP` deduces the total flows between pairs of polygons. This integration problem is solved by reducing the dimension of the integral and by using algorithms from computational geometry.

In addition, `RCALI` allows to take into account the angle of the current point with the horizontal and so, define anisotropic dispersal functions.

This manual first describes the methods implemented in `CaliFloPP`, then illustrates how to use it through `RCALI`, and last, gives some hints to customize the package.

Résumé

RCALI[1] est un paquetage R qui interface le logiciel CaliFloPP[2] à R.

Le logiciel `CaliFloPP` estime des flux de particules entre paires de polygones: à partir d'une fonction de dispersion dite individuelle, c'est-à-dire décrivant la dispersion des particules de point à point, il calcule les flux totaux émis d'un polygone à un autre. Ce problème d'intégration est résolu en réduisant la dimension de l'intégrale et en utilisant des algorithmes de géométrie algorithmique.

`RCALI` permet en outre de prendre en compte l'angle du point courant avec l'horizontale, et ainsi, de définir des fonctions de dispersion anisotropiques.

Cette notice décrit les méthodes implémentées dans `CaliFloPP`, illustre comment l'utiliser via `RCALI`, et comment adapter le paquetage.

[1] http://w3.jouy.inra.fr/unites/miaj/public/logiciels/RCALI
[2] http://w3.jouy.inra.fr/unites/miaj/public/logiciels/califlopp

Credits

People in Charge

- H. Monod and A. Bouvier, INRA, MIA, Jouy-en-Josas. http://www.inra.fr/miaj.

Authors of CaliFloPP

- K. Adamczyk, A. Bouvier, K. Kiêu et H. Monod, INRA, MIA, Jouy-en-Josas. http://www.inra.fr/miaj
- Ying Fu developped the prototype: see [Y.05].

Other Participants

- Nathalie Colbach, INRA, UMR Biologie et Gestion des Adventices INRA-ENESAD-Université de Bourgogne, Dijon, wrote the seed dispersal function, improved the pollen dispersal function, tested the programme and gave advice for practical adjustments.
- Mathieu Leclaire, computer engineer of the European Project SIGMEA on gene flow modelling, tested the programme and gave advice for practical adjustments.

Author of RCALI

- A. Bouvier, INRA, MIA, Jouy-en-Josas. http://www.inra.fr/miaj

Contents

Chapter 1

Introduction

1.1 An application: Pollen and seed dispersal between fields

The development of genetically modified (GM) plants has triggered much research to study how different types of agriculture can co-exist on a given landscape. In particular, several models have been developed and implemented to quantitatively describe and predict the risks of contamination of non-GM fields by GM fields, such as Genesys for oilseed rape [CCDM01a, CCDM01b].

A key stage in models such as Genesys consists in calculating the pollen and seed flow between two fields A and B. In Genesys, this calculation is performed by integrating a plant-to-plant (or individual) dispersal function ϕ over all emitting plants in A and all receiving plants in B, where ϕ is a function of the distance between the emitting and the receiving plants. The individual dispersal functions ϕ have been previously determined by specifically designed experiments (see *e.g.* [LKV+98]). In practice, because the plant density is high in a cultivated field, the integration is made continuously over A and B.

The calculation of field-to-field pollen and seed dispersals is a key stage not only for biological but also for numerical reasons. First, it is a non-trivial programming task. Relatively simple algorithms can be imagined for integrating the dispersal function over pairs of fields, but they may not be able to cope properly with the large diversity of field sizes and shapes which are met in actual agricultural landscapes. Second, the calculation requires a lot of computing time and so it imposes limits on the size of the landscapes one wants to study. The initial motivation for developing `CaliFloPP` was precisely to make the calculation of pollen and seed flow in Genesys more general and more efficient.

1.2 What CaliFloPP calculates : integrated flow of particles between polygons

Pollen and seed dispersal between fields is just an example of phenomena involving flows of particles between polygonal objects. Other examples include the flow of pathogen spores between fields in plant epidemiology, or the flow of polluting particles between sites in environmental applications.

`CaliFloPP` is a general programme, which makes it possible to calculate such global flows efficiently between pairs of polygons, by integration of an individual dispersal function. When running `CaliFloPP`, the basic entries that one needs to specify are :

- the coordinates of the vertices of each polygon ;

- the individual dispersal function ϕ.

In `CaliFloPP`, the polygons are considered as *continuous* and *homogeneous* sources of emission and continuous and homogeneous reception areas. The individual dispersal function ϕ between two points x and y in $\mathbb{R}^2$ is assumed to depend on $x_2 - x_1$ and $y_2 - y_1$, so that the argument of ϕ is a two-dimensional vector. As a special case, the dispersal may be isotropic so that $\phi(y - x)$ depends on $\sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2}$ only.

For each pair of polygons A and B, `CaliFloPP` calculates the integrated flow from A to B, that is

$$\mathcal{F}(A, B) = \int_A \int_B \phi(y - x)\, dy\, dx. \tag{1.1}$$

1.3 What CaliFloPP can help to calculate : an example

In many applications, the calculations performed by `CaliFloPP` will just represent an initial step, making time-consuming calculations once and for all before simulating a more complex space-and-time model.

Consider for example a landscape constituted of non-GM oilseed rape fields, GM oilseed rape fields, and other fields. Then the expected rate of contamination (due to pollen only, for simplicity) on a given non-GM field B can be defined as the proportion of pollen received by B which has been emitted by neighbouring GM fields. In the present context, this is equal to

$$\mathcal{C} = \frac{\sum_{\mathrm{GM}\, fields A} \mathcal{F}(A, B)}{\sum_{\mathrm{GM}\, fields A} \mathcal{F}(A, B) + \sum_{non-\mathrm{GM}\, fields A} \mathcal{F}(A, B)}. \tag{1.2}$$

Thus, calculating the expected rate of contamination for all non-GM fields requires to calculate $\mathcal{F}(A, B)$ for all pairs of fields with A an oilseed rape field and B a non-GM oilseed rape field. In Genesys, such calculations are performed over several years with different allocations of crops from one year to the next one. Thus, there is an interest in calculating $\mathcal{F}(A, B)$ over all pairs of fields once and for all.

Note that a more general form of equation (1.2) is

$$\mathcal{C} = \frac{\sum_{\mathrm{GM}\, fields A} \alpha_A \mathcal{F}(A, B)}{\sum_{\mathrm{GM}\, fields A} \alpha_A \mathcal{F}(A, B) + \sum_{non-\mathrm{GM}\, fields A} \alpha_A \mathcal{F}(A, B)},$$

where α_A denotes the quantity of pollen per squared meter emitted by field A and $\phi(y-x)$ must be interpreted as the *proportion* of particles emitted at point x that arrives at point y. As this example shows, the `CaliFloPP` calculations are perfectly compatible with models involving different levels of emission or reception between polygons.

1.4 The main steps of the CaliFloPP calculations

According to equation (1.1), the calculation of $\mathcal{A}(A, B)$ requires a four-dimensional integration, since A and B are both two-dimensional. When $\mathcal{A}(A, B)$ must be calculated for many pairs of polygons, this represents a heavy lot of computing time.

In fact, the first step in `CaliFloPP` consists in reducing the dimension of the integral from 4 to 2. This is done by taking profit of the invariance properties of the dispersal function (see Chapter 2).

The price to pay for the dimension reduction is the arrival of geometrical quantities in the reduced integral, whose calculations are not trivial. These calculations, however, can be performed efficiently provided tools from *computational geometry* are used. This is explained in Chapter 3.

In order to apply some of these methods, it is necessary that the polygons be convex. Chapter 3.4 describes a method to partition any polygon into convex sub-polygons.

The second step consists in the integration *per se*, for which several approaches have been considered. Two of them have been implemented in `CaliFloPP`: one based on regular grids over the domain of integration, the other based on cubature integration methods. They are described in Chapter 4.

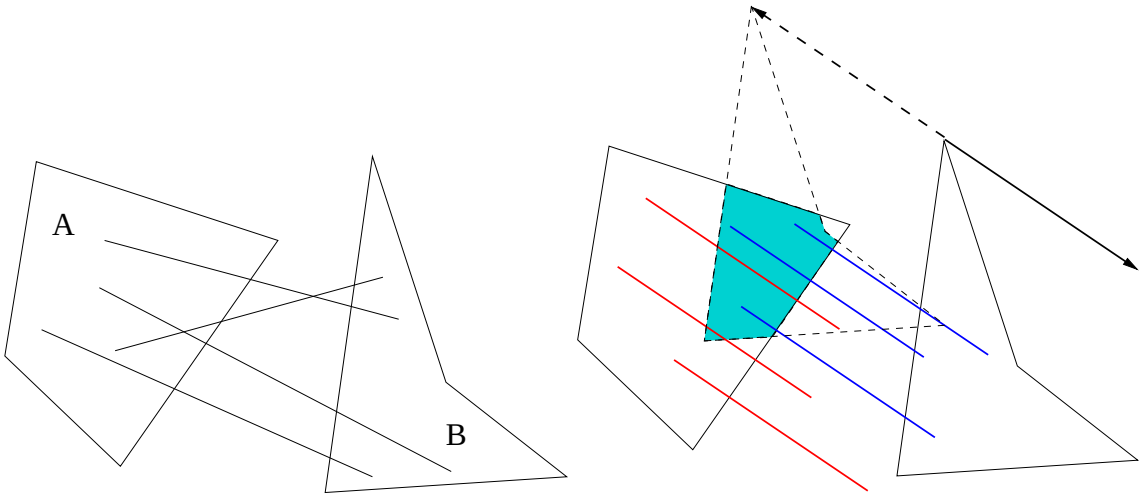
Part I

Methods for Particle Flow Integration

Chapter 2

Integral Reduction

The quantity $\mathcal{A}$ of particles from a polygon A to a polygon B, can be computed by integrating the individual dispersal function, $\phi(y-x)$, over pairs of points (x,y), $x \in A$, $y \in B$, see Section 1 and Fig. 2.1.



(a) The first and second points of each pair can be chosen independently in A and B.

(b) Consider all the pairs of points separated by the same vector t (t is in bold on the figure). For a given t, $x \in A$ and $x+t \in B$, if and only if $x \in A \cap (B-t)$ (such x and $x+t$ are joined here by blue segments; $A \cap (B-t)$ is represented by the blue area).

Figure 2.1: The quantity of particles from a polygon A to a polygon B is computed by integrating the individual dispersal function over all pairs of points $(x,y) \in A \times B$. These pairs are represented here by segments.

Rather than scrolling through the pairs of points by choosing each member independently in A and B, it is worth considering the pairs separated by the same vectors. The dispersal function is constant on such subsets of the pairs of points.

With the variable change $(x,y) \to (x, t = y-x)$, the set

$$\{y-x | x \in A, y \in B\}$$

can be written $\check{A} \oplus B$ where $\check{A} = -A$ and $\oplus$ stands for the Minkowski sum. The Minkowski sum of two sets A and B is simply the set obtained by addition of the points of A and B (see Fig. 2.2). On the other hand, for each vector $t \in \check{A} \oplus B$:

$$x \in A, y = x + t \in B \Leftrightarrow x \in A \cap (B - t)$$

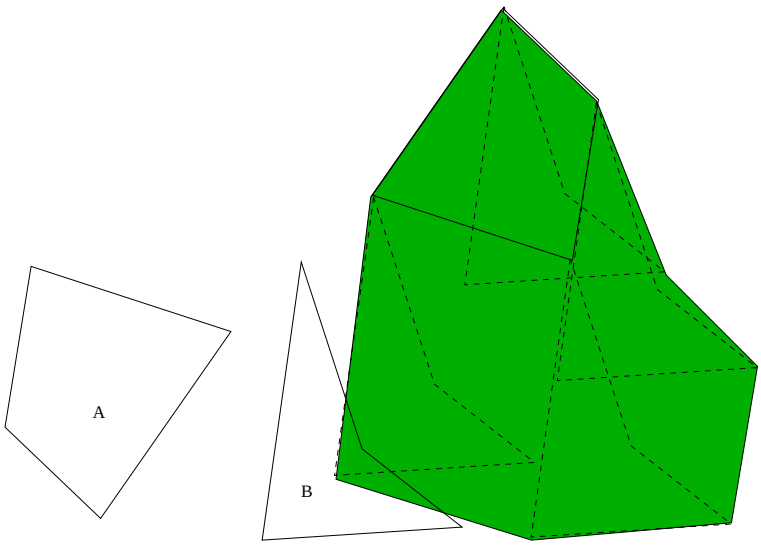


Figure 2.2: The set of points $t = y - x$, where $x \in A$ and $y \in B$, is written $\check{A} \oplus B$. It is the set of points covered by B when the vertex o is moved inside $\check{A}$. Here, to simplify, the origin of the plan o is located on a vertex of B.

Now, we can rewrite the integral $\mathcal{F}$ defined by the equation (1.1):

$$\mathcal{F} = \int_{\check{A} \oplus B} \int_{A \cap (B-t)} \phi(t)\, dx\, dt \tag{2.1}$$

as:

$$\mathcal{F} = \int_{\check{A} \oplus B} \int_{A \cap (B-t)} dx\, \phi(t)\, dt. \tag{2.2}$$

The integral in x is simply the area of $A \cap (B - t)$, so:

$$\mathcal{F} = \int_{\check{A} \oplus B} \text{area}(A \cap (B - t))\, \phi(t)\, dt \tag{2.3}$$

The computation of one of the two integrals has been replaced by the computation of an area and of an intersection, the intersection of A and a translation of B. The calculation of (2.3) has proven to be faster than the one of (1.1) [1] So, it is the formula implemented in `CaliFloPP`.

[1] Time comparisons have been made by using the integration subroutines of the NAG [NAG] library, D01FCF (adaptive integration, i.e where the evaluation spots depend on the integrand), and D01GCG (Korobov-Conroy method) on real and simulated fields.

Chapter 3

Geometric Computation

In order to compute numerically the two-dimensional integral (2.3), one must compute the area of the intersection area$(A \cap (B-t))$ for different values of $t \in \check{A} \oplus B$ (domain of integration) where A and B are polygons. Note that A and B are not necessarily convex. Computations of polygonal areas, intersections and Minkowski sums are known problems in *computational geometry*. The textbook [O'R98] is a rather nice introduction to that topic. Most algorithms briefly described below are provided there. Note that this chapter is only devoted to the computation of the geometric features involved in the integral (2.3). The problem of numerical integration is treated in Chapter 4.

The intersection $A \cap (B-t)$ is a polygon. The computation of the area of a polygon is straightforward, see Section 3.1 for the case when the polygon is convex.

The computation of the intersection $A \cap (B-t)$ is easy when both A and B are convex, see Section 3.2. When either A or B is not convex, the intersection may be complicated. In particular, it may not be connected.

The computation of the Minkowski sum $\check{A} \oplus B$ is rather easy if A or B are convex, see Section 3.3.

Hence the computation of the integral (2.3) is easy when both A and B are convex. This is why we propose to first decompose A and B as unions of convex polygons:

$$A = \bigcup_{i=1}^{n} A_i, \qquad B = \bigcup_{j=1}^{m} B_j, \tag{3.1}$$

where the areas of $A_i \cap A_{i'}$, respectively $B_j \cap B_{j'}$, are equal to 0 whenever $i \neq i'$, respectively $j \neq j'$. An algorithm for computing such a decomposition is described in Section 3.4. It is easy to check that the integral (2.3) can be written as:

$$\int_{\check{A} \oplus B} \text{area}(A \cap (B-t))\, \phi(t)\, dt = \sum_{i=1}^{n} \sum_{j=1}^{m} \int_{\check{A}_i \oplus B_j} \text{area}(A_i \cap (B_j - t))\, \phi(t)\, dt. \tag{3.2}$$

Based on the algorithm for decomposing polygons into convex ones and tools for computing numerically the integral (2.3) for any pair (A, B) of convex polygons (see Chapter 4), the integral (2.3) for an arbitrary pair of polygons can be computed using Algorithm 1.

Below, convex polygons are represented as sequences of vertices labeled counterclockwise: $(v_0, \ldots, v_{n-1})$. Many computations involve cycling over vertices. Hence it is convenient to use the convention $v_n = v_0$. All edges can be represented as $e_i = (v_i, v_{i+1})$, $i = 0, \ldots, n-1$.

Algorithm 1: Integral computation for an arbitrary pair of polygons

Data: Two polygons A and B, a dispersal function ϕ.
Result: Computation of area$(A \cap (B - t))$ for an arbitrary vector t.

Compute convex polygons $A_1, \ldots, A_n$ such that $A = \bigcup_{i=1}^{n} A_i$ and such that area$(A_i \cap A_{i'}) = 0$ if $i \neq i'$;
Compute convex polygons $B_1, \ldots, B_m$ such that $B = \bigcup_{j=1}^{m} B_j$ and such that area$(B_j \cap B_{j'}) = 0$ if $j \neq j'$;
$\mathcal{F} = 0$;
for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, m$ **do**
        Increment $\mathcal{F}$ by

$$\int_{\check{A}_i \oplus B_j} \text{area}(A_i \cap (B_j - t))\, \phi(t)\, dt;$$

    end
end

3.1 Area of a convex polygon

A convex polygon can be decomposed into triangles, see Figure 3.1. Thus its area is the sum of its triangles areas. Each of them is computed using the following formula

$$\frac{1}{2}(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A), \tag{3.3}$$

where A, B and C are the triangle vertices and the x's and y's are Cartesian coordinates. Equation (3.3) yields a signed area. It is positive if A, B and C are ordered counterclockwise, otherwise it is negative. The final result is exact if the coordinates are integers. The computing time depends on the number of polygon vertices. Algorithm 2 describes the whole procedure. Note that if the polygon vertices are numbered counterclockwise then the triangle vertices are also numbered counterclockwise.

Algorithm 2: Area of a convex polygon. Triangle areas are computed using Equation (3.3).

Data: Convex polygon with vertices (numbered counterclockwise) $v_0, \ldots, v_{n-1}$.
Result: Area of the polygon.

$a = 0$;
for $i = 1, \ldots, n-2$ **do**
    $a = a +$ area(triangle v_0, v_i, v_{i+1});
end

For area computation convexity is not an issue. The area of an arbitrary polygon is also easy to compute. If $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ are the Cartesian coordinates of the vertices (numbered counterclockwise) of a polygon, its area is equal to

$$\frac{1}{2}\sum_{i=1}^{n-1}(x_i + x_{i+1})(y_{i+1} - y_i). \tag{3.4}$$

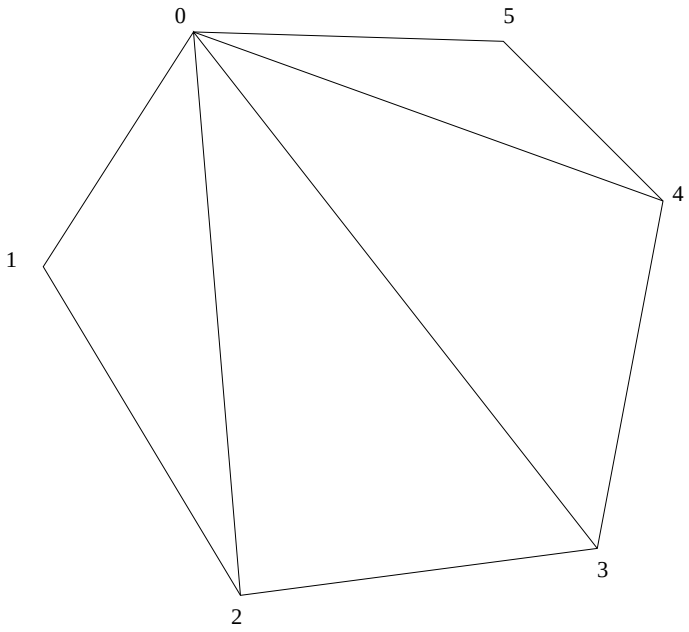


Figure 3.1: Decomposition of a convex polygon into triangles. An arbitrary vertex is connected to all non adjacent vertices.

3.2 Intersection of two convex polygons

Here we focus on the case when the boundaries of the two convex polygons P and Q meet each other: the cases where the intersection is void or where one polygon is included in the other one are not considered.

The intersection $P \cap Q$ is a convex polygon whose vertices are vertices of P or vertices of Q or intersection of edges of P with edges of Q. The intersection can be computed using the algorithm described in [O'R98, Section 7.6]. Two orientated edges $a \subset P$ and $b \subset Q$ are chosen. The edges a and b are advanced so that all the vertices of $P \cap Q$ can be detected and recorded, see Algorithm 3. Note that special cases are not treated: the algorithm is valid if P and Q are in general relative position, i.e. their boundaries cross only at the interior of edges. Also the implementation of this algorithm requires some further low-level functions:

- Test whether two segments meet.
- Compute the intersection of two segments.
- The advancing rule can be reformulated and requires only the computation of signed areas see [O'R98, Section 7.6] for further details.

3.3 Minkowski sum of convex polygons

Let P and Q be two convex polygons. An algorithm for computing the so-called convolution of P and Q is described in [O'R98, Section 8.4]. When both P and Q are convex, the convolution is equivalent to the Minkowski sum. The computation of the convolution is based on the *star diagram*

Algorithm 3: Intersection of two convex polygons

Data: Two convex polygons P and Q
Result: Computation of $P \cap Q$.

```
1  Choose an edge a of P and an edge b of Q;
2  R = ∅;
3  repeat
4      if a meets b then
5          if a ∩ b coincides with the first vertex of R then
6              Terminate;
7          else
8              Add a ∩ b to R;
9          end
10         Advance either a or b;
11     else
12         if One edge points toward the line containing the other then
13             Advance it;
14         else
15             if One edge is on the right-hand side of the other then
16                 Advance it;
17             else
18                 Advance either a or b;
19             end
20         end
21     end
22 until both a and b cycle along P and Q boundaries;
```

of P and Q edges. For any edge e of P Let $\alpha(e) \in [0, 2\pi)$ be the angle of e with an arbitrary axis. The star diagram is a polar representation of the $\alpha's$. The Algorithm 4 shows how to compute the Minkowski sum from the star diagram. Due to the convexity assumption, α is increasing on the set of edges of a polygon if the reference axis is parallel to its first edge.

Algorithm 4: Minkowski sum of two convex polygons.

Data: Two convex polygons P and Q.
Result: The Minkowski sum $P \oplus Q$.
`// Star diagram`
1 Compute α for all edges of P and Q taking as the reference axis a line parallel to the first edge of P;
2 Sort the α's: α_k, $k = 0, \ldots, n + m - 1$ where n is the number of P-vertices and m is the number of Q-vertices;
3 $R = \{$first vertex of $P\}$;
4 **for** $k = 0, \ldots, n + m - 1$ **do**
5 Add to R the latter vertex of R translated by $\vec{e}$ where e is the edge associated with α_k;
6 **end**

3.4 Convex partitioning of a polygon

In order to implement Algorithm 1, one needs to decompose an arbitrary polygon into convex polygons. The decomposition algorithm is three-step:

1. The polygon P is triangulated using an ear removal algorithm, see e.g. [O'R98, Section 1.6].
2. Essential diagonals of the triangulation are identified.
3. The convex subpolygons are created.

An internal (resp. external) diagonal is a segment joining two vertices which is contained in (resp. lies outside) the polygon. An ear is a triangle inside the polygon whose vertices are three consecutive vertices a, b, c of the polygon, i.e. ac is an internal diagonal.

The triangulation algorithm, see Algorithm 5, consists in successive ear removals. In order to test whether a vertex v_i is an ear tip, one tests whether $v_{i-1}v_{i+1}$ is an internal diagonal. The latter test is decomposed into two steps: first test whether $v_{i-1}v_{i+1}$ is a diagonal (internal or external), second test if $v_{i-1}v_{i+1}$ is internal. The first test involves edge intersection tests (loop along the edges). For the second test, one has to consider two cases: v_i is convex or reflex. The second test requires only computation of signed areas.

The triangulation yields internal diagonals (inner edges of the triangulation). A non-essential diagonal divides two adjacent cells whose union is still convex. It is easy to check that a diagonal is non-essential if both its ends are convex.

The last step is to split the polygon according to computed essential diagonals, see Algorithm 6.

Algorithm 5: Triangulation of a polygon.

Data: A polygon P with vertices $v_0, \ldots, v_{n-1}$.
Result: A triangulation of P
`// Find ear tips`
for $i = 0, \ldots, n-1$ **do**
  if $v_{i-1}v_{i+1}$ *is an internal diagonal* **then**
    Set v_i as an ear tip;
  end
end
$T = \emptyset$;
v = first vertex of P;
while P *has more than* 3 *vertices* **do**
  if v *is an ear tip* **then**
    Add to T the triangle uvw where u is the vertex previous to v and w is the vertex next to v;
    `// Update ear tip status of` u `and` w
    if *the vertices before and after* u *form a diagonal* **then**
      Set u as an ear tip;
    end
    if *the vertices before and after* v *form a diagonal* **then**
      Set v as an ear tip;
    end
    Remove v from P;
  end
  Advance v;
end

Algorithm 6: Creation of convex polygons from the essential diagonals

Data: A nonconvex polygon: its *nvertices* vertices and *ndiagonals* diagonals; the essential diagonals are marked. The sides of the polygon are essential diagonals.
Result: The *np* convex subpolygons: their vertices are stored anticlockwise in *polyg*.
$np = 0$
while *there is an essential diagonal* **do**
  (v_a, v_b) = any essential diagonal
  $start = va$
  while $vb! = start$ **do**
    store (v_a, v_b) into $polyg[np]$
    mark (v_a, v_b) non-essential
    `// Determine the following diagonal:`
    (v_b, v_c) is the diagonal starting from v_b and such as the angle (v_a, v_b, v_c) is minimum
    $v_a = v_b; v_b = v_c$
  end
  $np = np + 1$
end

Chapter 4

Two-dimensional Numerical Integration

This chapter is devoted to the numerical computation of integrals of the type

$$\int_{\check{A}\oplus B} \operatorname{area}(A \cap (B - t))\, \phi(t)\, dt, \tag{4.1}$$

where A and B are bounded polygons and ϕ is an arbitrary integrable dispersal function. As seen in Chapter 3, without loss of generality we can focus on the case when both A and B are convex.

We will start by preliminary remarks concerning the smoothness of the integrand. Then we will discuss two methods for numerical integration:

- A simple randomized discretization of the integral;
- An adaptive cubature method.

The first method, described in Section 4.2, combines simple discretization and Monte-Carlo techniques. This method is quite robust (convergence even for non-smooth integrands). It yields unbiased estimates and it is able to assess its precision. However it may be slow compared to other methods.

The second method consists of an adaptive cubature. First, the domain of integration (a convex polygon) is triangulated. Then the integral over each triangle is approximated using the adaptive cubature method proposed by Berntsen and Espelid [BE92]. The convergence of this method depends on the smoothness of the integrand. It may fail to converge if the integrand cannot be approximated locally by a polynomial. Some basic results about the smoothness of the integrand in Equation (4.1) are stated in Section 4.1. Details about the adaptive cubature method are provided in Section 4.3.

4.1 Smoothness of the integrand

The integrand in Equation (4.1) is the product of two functions:

$$\operatorname{area}(A \cap (B - t)) \text{ and } \phi(t).$$

The function $t \mapsto \operatorname{area}(A \cap (B - t))$ is a piecewise linear function. Its support $\check{A} \oplus B$ can be split by line segments obtained by adding edges of $\check{A}$ and edges of B, see Figure 4.1. Inside each cell of

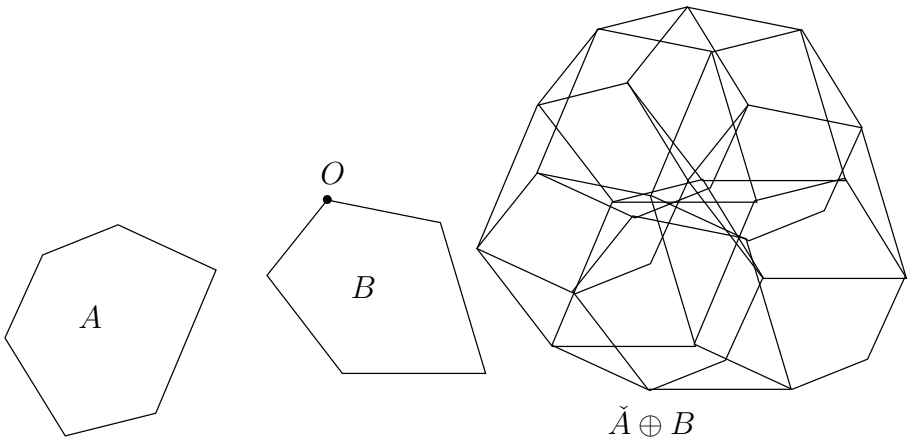


Figure 4.1: Partitionning of the Minkowski sum of two convex polygons A and B according to their edges.

this partition, $t \mapsto \text{area}(A \cap (B - t))$ behaves linearly. The differential $t \mapsto \text{area}(A \cap (B - t))$ is not continuous along the network of cell boundaries.

The smoothness of the whole integrand also depends on the smoothness of the dispersal function ϕ. For instance, the dispersal function given in Equation (10.2) is infinitely continuously differentiable except at the origin where it is not twice differentiable. Then the whole integrand is infinitely continuously differentiable except along the cell boundaries in the partition of the Minkowski sum where it is not differentiable and at the origin where it is not twice differentiable. Another example: the dispersal function given in Equation (10.1) is not differentiable at the origin and it is not twice differentiable along a circle with radius 1.5 meter centered at the origin. Then the whole integrand is not differentiable neither at the origin nor along the cell boundaries and it is not twice differentiable along the circle.

4.2 Method based on grids of points

A simple and intuitive method to perform numerical integration consists of evaluating the integrand over a regular grid of points. The integral is then approximated by summing the integrand over the grid points, and by multiplying the result by the volume of each cell in the grid. Provided the grid position is randomised, this method yields an unbiased estimate of the integral. In addition, it can be repeated several times, with grid positions randomised independently. The integral is then estimated by the mean over the replicated grids. Replications increase precision but also allow the standard error to be estimated.

To simplify, this method will be called **the grid method** in the sequel. In `CaliFloPP`, the distances between the nodes of the grids are the same for all grids and they are chosen by the user. The number of replications is also set by the user.

Let A and B be two polygons. The main steps are :

1. calculation of the Minkowski sum $\check{A} \oplus B$;

2. determination of the smallest rectangle which includes the Minkowski sum and whose sides are parallel to the axes (see Fig. 4.2) ;

3. integration over this rectangle, with the integrand multiplied by a coefficient w equal to 1 if the point is in the Minkowski sum, to 0 otherwise.

Integration is performed as mentioned above, using several grids of points with the following properties :

- all grids have the same x and y lags between adjacent nodes, chosen by the user ;
- each grid is positioned randomly on the integration area, by shifting it randomly relatively to the origin. The shifting is determined by pseudo-random numbers from the uniform distributions over the intervals $[0, p_x]$ and $[0, p_y]$, where p_x and p_y denote the x and y lags between adjacent grid nodes.

The integral is estimated by the mean integral over the replicated grids.

4.2.1 Additional results

- **Standard deviation**

  The `standard deviation` is defined as :

$$\mathrm{sd} = \frac{\sqrt{\sum_{i=1}^{r}(a_i - a_\bullet)^2}}{r-1} \tag{4.2}$$

  where a_i is the integral value calculated by replication i, and r the number of grid replications.

- **Coefficient de variation**

  The `coefficient de variation` is defined as :

$$\mathrm{CV} = \frac{\mathrm{sd}}{a_\bullet} \tag{4.3}$$

- **Precision and confidence intervals**

  Precision and confidence intervals can be calculated by the user from the replications results. For example, when the number of replications is large enough and the distributions are symmetric, they can be calculated from the quantile of the Student distribution. The half-range hr of the confidence interval, or the absolute precision, is then defined as :

$$\mathrm{hr} = \mathrm{sd} \times \frac{\mathrm{qt}(r-1, p)}{\sqrt{r}} \tag{4.4}$$

  where $\mathrm{qt}(n, p)$ is the quantile for probability p of the Student distribution with n degrees of freedom. The relative precision is $\frac{hr}{a_\bullet}$.

4.3 Adaptive cubature method

The cubature method over triangles DCUTRI [BE92] uses an integration rule of degree 27 based on 37 points. Using estimates of approximation errors, triangles are iteratively split into subtriangles until a nominal error is reached. At each iteration the triangle with the largest error is selected for further splitting. The procedure is expected to converge if the integrand is smooth enough inside each triangle. For instance, if the dispersal function is singular at the origin and if the domain of integration $\check{A} \oplus B$ contains the origin, one expects a quicker convergence when the origin is a vertex of the triangulation. Hence (see Figure 4.3):

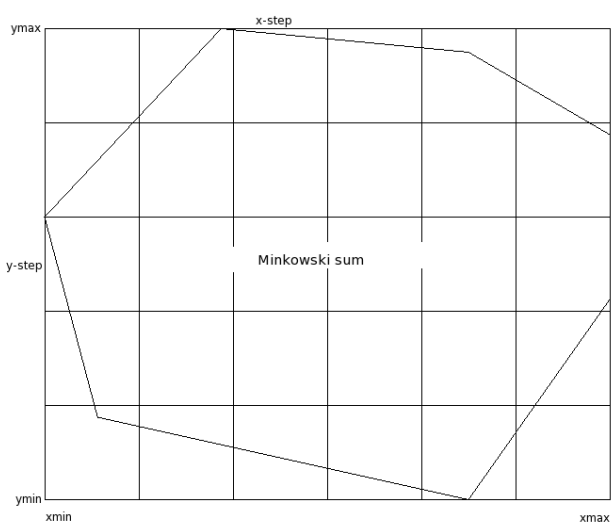


Figure 4.2: Minimal rectangle including the Minkowski sum and grid of points before random shifting

- if $\check{A} \oplus B$ does not contain the origin, it is triangulated from an arbitrary vertex,
- if $\check{A} \oplus B$ contains the origin O, it is triangulated from O.

A further refinement is to avoid to integrate over areas where the dispersal function is considered as negligible that is below a chosen threshold $r_{\max}$. Note that whenever the dispersal function is non-negative and tends to 0 at infinity, it can be considered in practice as null when it is less than the smallest positive number greater than zero for the used type of number. Thus instead of integrating over the whole Minkowski sum $\check{A} \oplus B$, one may integrate only on its intersection with a disc centered at the origin and with radius $r_{\max}$. In practice it is simpler to replace the disc by a regular polygon e.g. an octogon containing it, see Figure 4.4. Hence integration is still performed on a convex polygon.

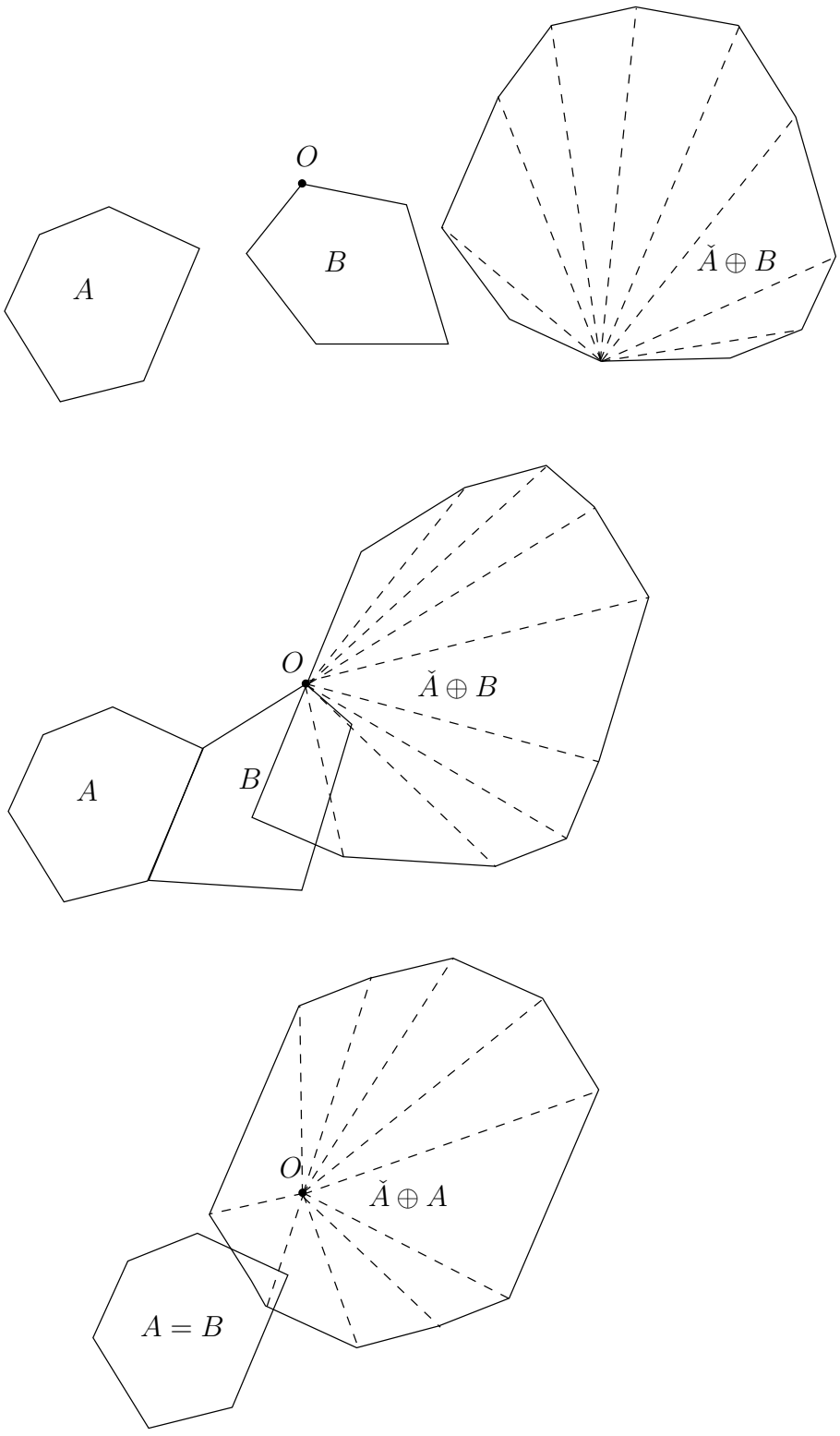


Figure 4.3: Triangulation of the Minkowski sum of two convex polygons A and B. If A and B are disjoint (top), the Minkowski sum $\check{A} \oplus B$ does not contain the origin and is triangulated from an arbitrary vertex. If A and B share a common edge (middle) or if $A = B$ (bottom), the Minkowski sum $\check{A} \oplus B$ contains the origin O and is triangulated from O.

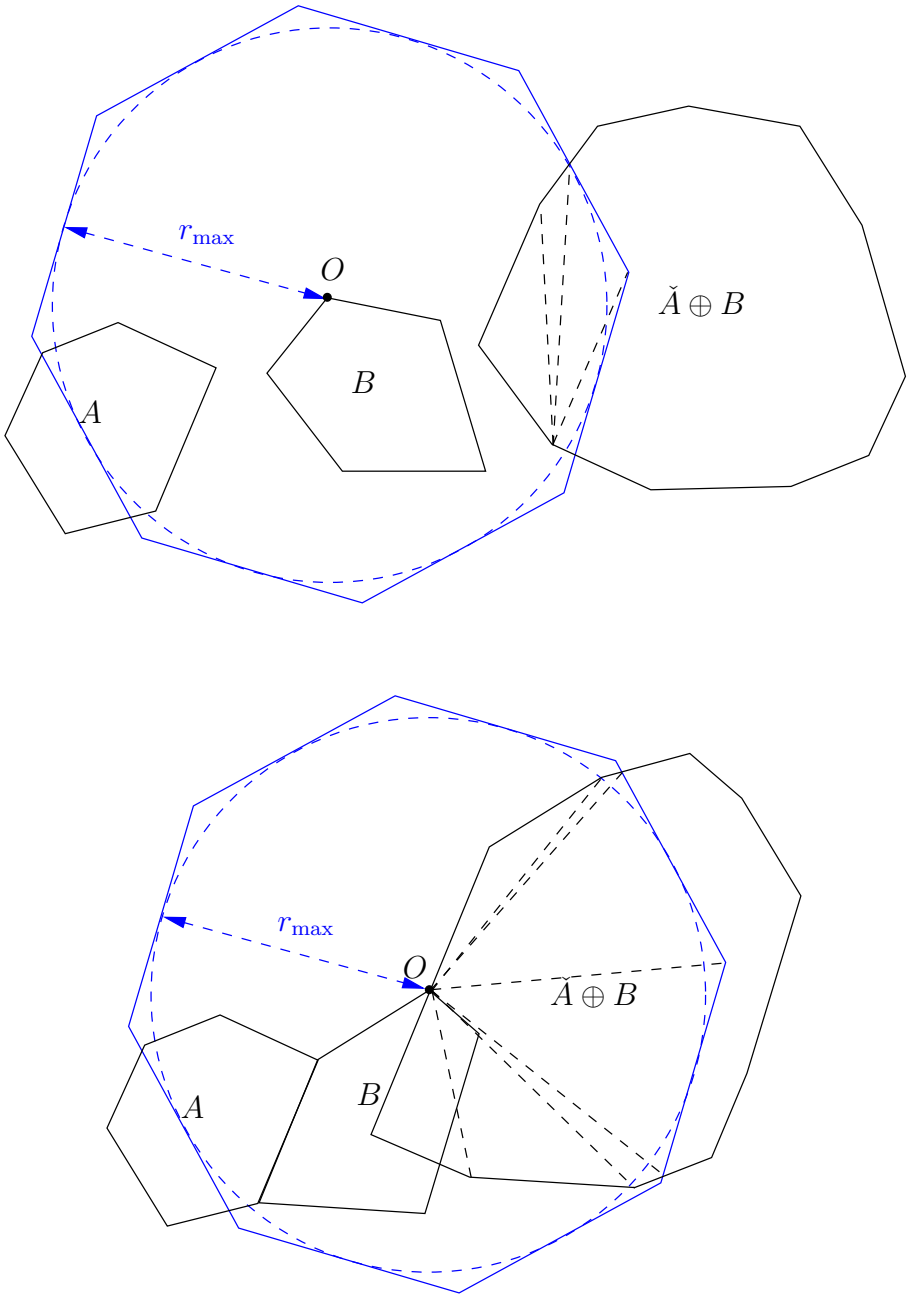


Figure 4.4: Reduction of the domain of integration. Beyond $r_{\max}$ the dispersal function is considered as negligible. The integral is computed only over the intersection of $\check{A} \oplus B$ and an octogon (blue solid line) containing the disc centered at the origin with radius $r_{\max}$ (blue dash line). Top: the Minkowsi sum $\check{A} \oplus B$ does not contain the origin . Bottom: the Minkowski sum contains the origin.

Part II

Example

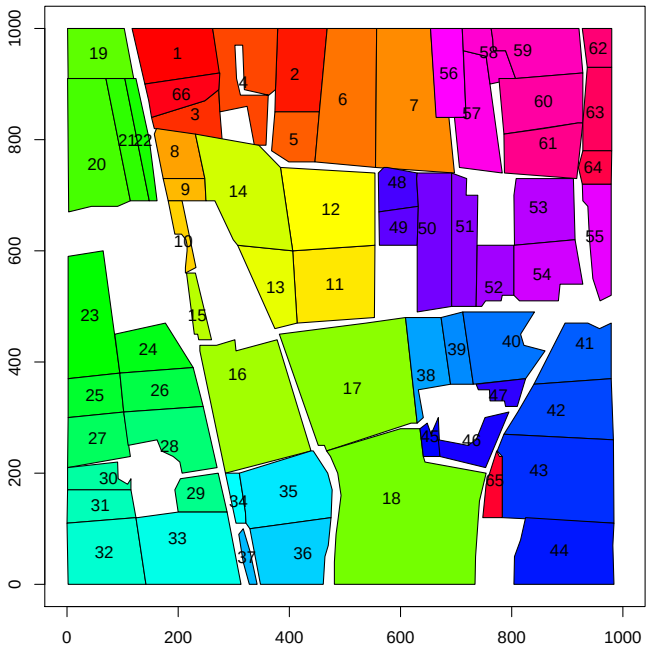


Figure 4.5: Example of 66 polygons extracted from the ORTHO IGN data base. Unit is meter.

4.4 Introduction

In this Section, we analyze the behavior of the grid and cubature methods on a real landscape. First, we compare the results calculated with different parameterisations of each method; then, the two methods are compared.

4.4.1 Data

The data are 66 polygons extracted from the ORTHO demo, a IGN[1] data base , see Fig. 4.5. To illustrate contrasted situations, five pairs of polygons are treated:

- **(1,1)**: two identical convex polygons,
- **(14,14)**: two identical non-convex polygons,
- **(11,12)**: two convex polygons next to each other,
- **(56,57)**: two polygons next to each other, one convex, the other non-convex,
- **(4,4)**: two identical very irregular polygons.

[1]Institut Géographique National: http://www.ign.fr/rubrique.asp?rbr_id=1619

Table 4.1: Grid results with different numbers of replications (r) and grid step equal to 1 m.

Polygons	r	$\widehat{\mathcal{F}}$	$\widehat{\sigma}$	$\widehat{\sigma}/\widehat{\mathcal{F}}\times 100$	*Times*
1↔1	5	12222.2	234.6	1.88	3.5
	10	12282.5	217.7	1.77	7.0
	15	12238.8	204.1	1.66	10.5
	20	12247.7	220.9	1.80	14.1
14↔14	5	23542.1	461.6	1.96	31.9
	10	23359.0	391.1	1.67	63.7
	15	23411	344.4	1.47	95.3
	20	23413.3	332.4	1.41	126.7
11↔12	5	164.5	2.4	1.48	7.0
	10	166.1	3.1	1.89	14.0
	15	167.3	3.1	1.85	21.1
	20	167.2	3.1	1.84	27.9
56↔57	5	132.5	2.0	1.51	5.9
	10	133.0	2.2	1.70	11.7
	15	132.5	2.1	1.62	17.6
	20	132.8	1.9	1.49	23.4
4↔4	5	14610.1	72.3	0.5	45.7
	10	14570.9	91.5	0.62	90.7
	15	14617.7	155.2	1.06	136.7
	20	14631.5	151.5	1.03	181.2

The individual dispersal function is the oilseed rape pollen dispersal function defined in Section 10.8.1.

4.5 Influence of the parameterisation in the grid method

In the grid method, two parameters must be chosen according to the user's needs: the number of replications and the grid step.

4.5.1 Influence of the number of replications

To compare the results calculated with different numbers of replications, four values were successively applied: $r = 5$, 10, 15, 20. The grid step was constant and equal to 1 m for both axes.

The results[2] are given in Table 4.1. They are: the evaluated mean flow ($\widehat{\mathcal{F}}$), the standard deviation ($\widehat{\sigma}$), the coefficient of variation ($\widehat{\sigma}/\widehat{\mathcal{F}}$) and the execution time[3]. We can notice that the execution times are the longer as the polygons are the more irregular: calculation is faster on convex polygons (1↔1, 11↔12) than on non convex ones (14↔14, 56↔57) and very much longer on irregular polygons (4↔4).

4.5.2 Influence of the grid step

To compare the results calculated with different grid steps, four values were tried successively: *step* = 0.25 m, 0.5 m, 0.75 m and 1 m. The number of replications, r, is set to 10.

The results are displayed in Table 4.2. As expected, the times increase as the steps become smaller.

[2]Results are dependent on the random numbers generator; so, actual values may be slightly different.

[3] Execution times depend on the material context. They have been observed here on a Dell Biprocessor, in shared mode and 3,2GhZ.

Table 4.2: Grid results with different steps ($r = 10$);

Polygons	*step*	$\widehat{\mathcal{F}}$	$\widehat{\sigma}$	$\widehat{\sigma}/\widehat{\mathcal{F}}\times 100$	*Times*
1↔1	1	12282.5	217.7	1.77	7.0
	0.75	12196.9	27.3	0.22	12.6
	0.5	12277	32.8	0.26	28.5
	0.25	12273.8	4.08	0.03	111.5
14↔14	1	23359.0	391.1	1.67	63.7
	0.75	23362.7	183.2	0.78	114.5
	0.5	23376.7	63.4	0.27	258.5
	0.25	23380.6	7.7	0.03	1016.6
11↔12	1	166.1	3.1	1.89	14.0
	0.75	167.2	1.4	0.85	25.3
	0.5	167.1	0.5	0.34	57.1
	0.25	167.2	0.1	0.05	223.9
56↔57	1	133.0	2.26	1.70	11.7
	0.75	132.5	0.87	0.66	21.1
	0.5	132.5	0.32	0.24	47.8
	0.25	132.5	0.02	0.02	187.5
4↔4	1	14570.9	91.5	0.62	90.7
	0.75	14585.4	46.9	0.32	164.5
	0.5	14607.7	14.8	0.10	370.9
	0.25	14612.8	1.85	0.01	1452.8

4.6 Influence of the parametrization in the cubature method

In the cubature method, two parameters must be chosen: the maximum number of function evaluations and the precision.

4.6.1 Influence of the number of evaluations

We compare the results calculated with different numbers of evaluations: $Neval = 10^6,\ 10^5,\ 75 \times 10^3,\ 5 \times 10^4$.

As the integration process stops as soon as either the maximal number of evaluations or the required absolute or relative precisions are reached, these precisions should be small enough to ensure that all the evaluations are run (here, the required precisions are set to 1.0e-30).

The results are given in Table 4.3[4] and the confidence intervals are represented in Fig. 4.6. The results obtained by the grid method with a rather great number of replications ($r = 10$) and a small step ($step = 0.25$ m.) are given as references.

4.6.2 Influence of the required precision

To evaluate the impact on the results of the required relative precision, several values were successively tried: $req.rel.er = 0.0001,\ 0.001,\ 0.01,\ 0.1$. The number of evaluations was set to its maximum.

The results are summarized in Table 4.4 and the confidence intervals are represented in Fig. 4.7.

As previously, the results calculated by the grid method with $r = 10$ and $step = 0.25$ m., are given as reference values.

[4] The actual number of evaluations may be slightly less than the required number, because it is a multiple of the number of triangles built by the method on each integration regions.

Table 4.3: Cubature results with different numbers of evaluations. The columns 2 and 3 are grid results ($r = 10, step = 0.25$). The subsequent ones are cubature results with different numbers of evaluations.

Polygons	$\widehat{\mathcal{F}}$	*Times*	$\widehat{\mathcal{F}}$	*rel.er*	*abs.er*	*Neval*	*Times*
1↔1	12273.8	111.5	12273.2	2.7e-6	0.033	999962	13.3
			12273.2	3.3e-5	0.4	99974	1.3
			12273.2	4.6e-5	0.57	74962	1.02
			12273.2	5.3e-5	0.65	49950	0.65
14↔14	23380.6	1016.6	23381.1	1.8e-5	0.42	999999	15.1
			23381.4	0.0003	7.7	99863	1.47
			23381.5	0.0006	15.07	74999	1.14
			23379	0.002	54.09	49987	0.72
11↔12	167.2	223.9	167.3	1.3e-6	0.0002	999962	16.8
			167.3	3e-5	0.005	99974	1.6
			166.3	4.2e-5	0.007	74962	1.3
			167.3	8e-5	0.013	49950	0.83
56↔57	132.5	187.5	132.4	5.2e-6	0.0007	999888	17.2
			132.5	0.00010	0.013	99900	1.7
			132.5	0.00016	0.022	74888	1.37
			132.5	0.0005	0.062	49876	0.87
4↔4	14612.8	1452.8	14613.5	5.7e-5	0.8	999888	14.3
			14611	0.0066	96.7	99900	1.4
			14625.6	0.017	249.5	74888	1.15
			14623.9	0.058	849.6	49876	0.73

Table 4.4: Cubature results with different relative precisions required. The columns 2 and 3 are grid results ($r = 10, step = 0.25$). The subsequent ones are cubature results with different relative precisions required.

Polygons	$\widehat{\mathcal{F}}$	*Times*	$\widehat{\mathcal{F}}$	*req.rel.er*	*rel.er*	*abs.er*	*Neval*	*Times*
1↔1	12273.8	111.5	12273.3	0.0001	9.8e-5	1.21	23754	0.3
			12273	0.001	0.00093	11.48	8362	0.1
			12283.4	0.01	0.0098	121.28	5550	0.08
			12279.7	0.1	0.066	820.0	4070	0.06
14↔14	23380.6	1016.6	23381.2	0.0001	9.9e-5	2.3	218707	3.24
			23381.8	0.001	0.00099	23.2	62715	0.9
			23370.4	0.01	0.0099	233.3	30007	0.45
			22762.8	0.1	0.099	2275.2	12395	0.21
11↔12	167.2	223.9	167.3	0.0001	9.9e-5	0.016	42402	0.7
			167.28	0.001	0.00097	0.16	10286	0.2
			166.8	0.01	0.009	1.5	2738	0.05
			166.7	0.1	0.07	12.5	2442	0.05
56↔57	132.5	187.5	132.4	0.0001	9.9e-5	0.013	100344	1.76
			132.3	0.001	0.0009	0.13	16872	0.99
			132.3	0.01	0.0098	1.3	8140	0.19
			132.5	0.1	0.096	12.78	4292	0.08
4↔4	14612.8	1452.8	14613.5	0.0001	9.9e-5	1.46	587708	8.6
			14613.9	0.001	0.0009	14.6	176860	5.16
			14622	0.01	0.0099	145.4	86136	2.17
			14432.3	0.1	0.099	1430.26	38332	1

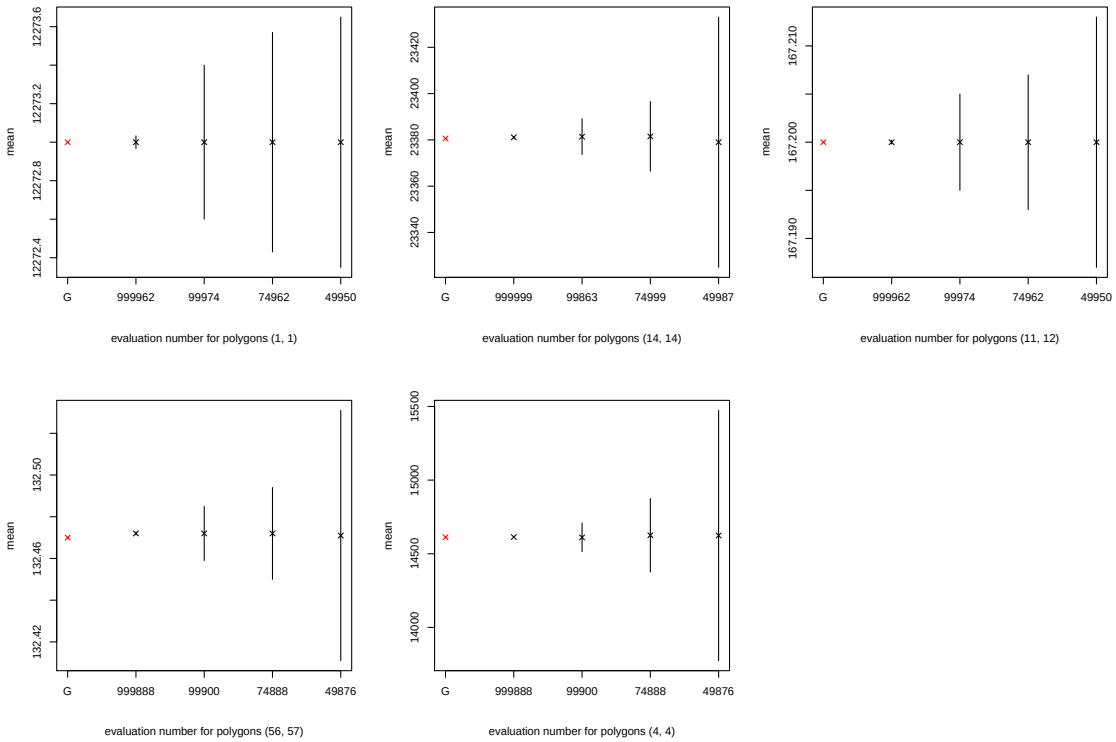


Figure 4.6: Cubature method: Mean and confidence interval against number of evaluations. The first value (abscissa G) is the grid reference value.

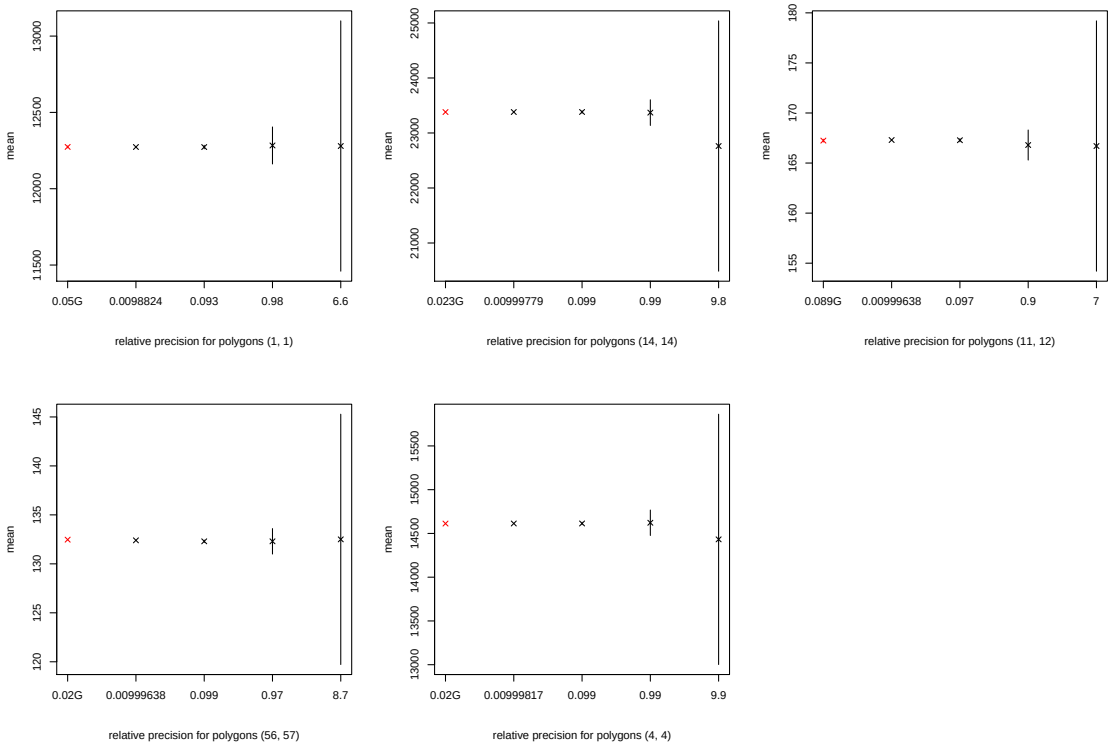


Figure 4.7: Cubature method: Mean and confidence interval against relative precision. The first value (abscissa suffixed by G) is the grid reference value.

4.7 Global comparison

4.7.1 Global summary

From these computation experiments, some general characteristics can be noticed:

- When the required precision is small enough, the results calculated by the cubature and grid methods are very similar.
- The polygon shape is influent on execution times, in both methods: calculation is faster on convex polygons than on nonconvex ones and very much longer on irregular polygons.
- The cubature method is faster than the grid method.
- In the cubature method, the maximum number of evaluations should be great enough for the precision to be reached. This is all the more so since the polygons are more irregularily shaped. For very irregular polygons, convergence may not be reached, whatever the number of evaluations is.

4.8 Influence of the dispersal function

Section 4.1 has pointed out possible problems when the dispersal function is not smooth (its derivative is not continuous[5]).

To bring into light this behavior, the following non differentiable individual dispersal function is proposed:

$$\phi(t) = \begin{cases} a - b \times t^2 & \text{when } t \leq \sqrt{a/b} \\ 0 & \text{otherwise,} \end{cases}$$

with $a = 10, b = 20$. See Fig. 4.8

All the results calculated by the cubature method on the chosen pairs of polygons are then null except for the pair 4↔4, whatever the required precision is. The grid method gives coherent values.

Comments: Before applying the cubature method on a new dispersal function, it is strongly recommended to compare some results with the ones calculated by the grid method.

4.9 Conclusion

Comparison of the results calculated by the grid and cubature methods on different types of polygons extracted from a real landscape has shown the coherence of these methods. The shapes of the polygons and the required precision of the results have great impact on the execution times with a great advantage for the cubature method. However, the grid method is convenient to provide reference values in case of non convergence or when testing non smooth individual dispersal functions.

[5] When the dispersal function becomes very suddenly null, the triangles built by the cubature method may intersect the support of the function without any evaluation points being in this support.

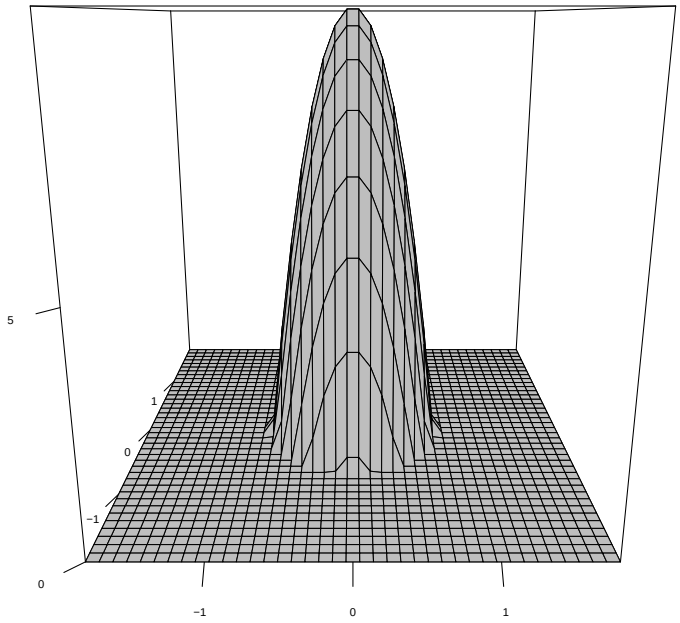


Figure 4.8: Non derivable dispersal function.

Part III

Customization Guide

Chapter 5

Configuration variables

After downloading and un-taring the tar-file of the package, the file `src/caliconfig.h` contains configuration variables that you should customize according to your needs. After modification, recompilation is required: see 6 If you modify the value of any of them, reflect this change in the help file of the R function **califlopp** (`man/califlopp.Rd`).

The following tables describe the configuration variables.

In the first column, in addition to the variable names, information is given about the possibility for the user to modify the default value: an asterisk means that it is the case, through the argument **param** of the R function **califlopp** (in an **R** session, see the on-line help of **califlopp**). The name of the corresponding component of **param** is given in parenthesis in the following table.

5.1 Input

Name	Meaning	Comments
DEFAULT_INPUT_FORMAT (input) (*)	Format of the polygons-file	- should be 1 if each polygon is coded on two lines: 1/ an identification number, followed by the x-coordinates 2/ the same number, followed by the y-coordinates, - should be 2 if each polygon is coded on three lines: 1/ an identification number, a name, the number of vertices (followed possibly by other data that are ignored) 2/ the x-coordinates 3/ the y-coordinates.
DEFAULT_DELIM (delim) (*)	Separator character in the polygons-file	should be between double-quotes
MAX_LINE_POLY	Maximal number of characters on each line of the polygon file	
MAX_NAME	Maximal length of the polygon names	
PATH_MAX	Maximal number of characters for pathnames	usually, PATH_MAX is defined in stdio.h

5.2 Output

Name	Meaning	Comments
OUTPUT_FILE_FORMAT	Content of the result-file	- should be ALL to output all the results - should be FLOW to output the polygon identifiers and the flow by square meter, - should be LIGHT to output all the results except for the time.
OUTPUT_WARNING	Warnings output on the error unit	- should be ALL to print all warnings, - should be NOTHING for minimum warnings.
DEFAULT_OUTPUT (output)	Output on the standard output unit	- should be ALL to print all the results, - should be FLOW to print the integrated flows, the flows by m^2, - should be LIGHT to print the integrated flows, only, (one line per pair of polygons) - should be NOTHING for no print.
DEFAULT_VERBOSE (verbose) (*)	verbose mode	should be 1 to get output about the decomposition into convex polygons and landscape relocation, and 0 otherwise

5.3 Error treatment

Name	Meaning	Comments
ERR_POLY	treatment of erroneous polygons	- should be 0 if an error on a polygon should be a warning: the erroneous polygon is then ignored - should be 1 if an error on a polygon should be fatal

5.4 Landscape features

Name	Meaning	Comments
MAX_VERTICES	Maximal number of vertices per polygon	
MAX_TRIANGLES	Maximal number of convex subpolygons per polygon	This number depends on the polygons shapes: more they have obtuse angles, more the number of convex subpolygons should be great. But, be careful: If values of MAX_VERTICES and MAX_TRIANGLES are too big, execution errors may occur ("Segmentation fault" or "Out of memory")
TRANSLATE	Landscape relocation.	Should be 1 if the landscape should be systematically relocated, so that the left-bottom corner of the landscape is (1,1). (Recommanded value)
SCALE	The polygon-coordinates are multiplied by SCALE.	Should be 1 or a multiple of 10. For example, to take into account centimeters, set SCALE to 100
SAFE	Maximal range of the coordinates	It is the maximal range of the coordinates after they have been multiplied by SCALE. SAFE should be less than INT_MAX (which is usually= 2147483647)
DISTP	When the distance between two successive vertices is less than or equal to DISTP, the second vertex is suppressed.	Expressed in meters.
ANGLEPREC	Precision of the angle between 3 successive vertices.	When the arccosinus of the angle between three successive vertices is inside [π-ANGLEPREC, π+ANGLEPREC], the vertices are considered as aligned, and the second one is suppressed. When it is inside [-ANGLEPREC, +ANGLEPREC], it is supposed that the sharp spike they form is an artefact, and the second one is suppressed.

5.5 Individual dispersal functions

Name	Meaning
DZ1, DZ2, DZ3, DZ4, DZ5	Thresholds for dispersal distances: When the minimal distance between two polygons is greater than or equal to these values, the corresponding dispersal function (f1 for DZ1, ... f5 for DZ5) is supposed to be null; distances are in meter. Negative or null values mean that there is no limit in the dispersal.
DP1, DP2, DP3, DP4, DP5	Thresholds for dispersal distances: When the minimal distance between two polygons is greater than or equal to these values, the dispersal is calculated between polygons centroids; distances are in meter.

5.6 Methods features

5.6.1 Cubature method

Name	Meaning
DEFAULT_ABS_ERR (abser) (*)	Default absolute precision
DEFAULT_REL_ERR (reler) (*)	Default relative precision
DEFAULT_MAX_PTS	Maximal number of evaluation points per integration region.
DEFAULT_NB_PTS (maxpts) (*)	Default maximal number of evaluations per triangle (should be in [37,DEFAULT_MAX_PTS])
MAX_SREGIONS	Maximal number of subregions per integration region.
TZ1, TZ2, TZ3, TZ4, TZ5 (tz) (*)	Method of triangulation for the cubature method. Should be True, if triangulation from (0,0) has to be done when (0,0) is included in the integration area (recommended value when the dispersal function is very "sharp" at the origin).

5.6.2 Grid method

Name	Meaning
MAX_EST	Maximal number of estimations
DEFAULT_EST (nr) (*)	Default number of estimations ($\leq$ MAX_EST)
DEFAULT_STEPX ($step_0$) (*)	Default step on the x-axis grid of points; in meters.
DEFAULT_STEPY ($step_1$) (*)	Default step on the y-axis grid of points; in meters.
DEFAULT_SEED (seed) (*)	Default value of the seed for the random number generator.

5.7 Numerical parameters

Name	Meaning	Comments
REAL_PREC	Precision for real comparisons in geometrical computations	Recommended: (REAL_MIN*1.0e+4)

Chapter 6

How to make changes

1. Download the tar-archive file of `RCALI` package and untar it: a directory named `RCALI` is created.
2. Possibly, reflect the changes you make in the source code in the help-file `man/califlopp.Rd` of the function `califlopp` (default values of the parameters).
3. After alteration of the source code, recompilation is required. Typically use the standard R command: `R CMD build RCALI` on top of your `RCALI` directory.

Part IV

User Guide

Chapter 7

The polygons-file

In accordance with the gene flow application and the dispersal functions defined in Section 10.8, we assume in the following that 1 unit in the polygon coordinates corresponds to 1 m.

7.1 Constraints on the polygons

- Polygons should be without holes.
- Small and narrow polygons (approximately less than 1 m^2) should be avoided because of possible numerical problems, as well as polygons with many obtuse angles because their decomposition into convex subpolygons may not be possible.
- Shape and size restrictions are set in the file `src/caliconfig.h`. They are:

  MAX_VERTICES, the maximal number of vertices per polygon,
  MAX_TRIANGLES, the maximal number of convex subpolygons per polygon,
  SAFE, the maximal extent of the landscape.

Notes: There is no limit in the number of polygons (except possible memory limitation) and the polygons may intersect.

7.2 Syntax of the polygons file

The polygons file contains the coordinates of the polygons. It should respect the following rules:

- It should be an ASCII-file.
- Vertices should be ordered clockwise.
- The polygons may be closed or not.
- The coordinates may be negative or null. The number of decimal digits taken into account depends on the constant `SCALE`[1]. (See Section 10.4).

[1] Constant set in the file `src/caliconfig.h`

- The values separator is the character DEFAULT_DELIM[1] set in the file `src/caliconfig.h`. It can be changed by the component `delim` of the argument `param` of the main function **califlopp**, of `RCALI`.

  The separator character can be repeated any number of times between successive values.

- Two formats for the polygons file are catered for:

  The default format is defined by the constant DEFAULT_INPUT_FORMAT[1]. It can be changed by the component `input` of the argument `param` of the main function **califlopp**, of `RCALI`.

- In format 1, there are 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 R `export.listpoly` generates such a file from R structures

- In format 2, there are 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 polygon names may consist of several words, as long as these words are not separated by the values separator character.

Note:
If the number of polygons set on the first line, `npoly`, is less than the effective number, only the first `npoly` polygons will be treated.

Chapter 8

Input

The argument `dispf` of the main function of `RCALI`, **califlopp**, describes the dispersal functions. The argument `param` describes the other entries.

8.1 The dispersal functions

The required dispersion functions can be described by two different ways in the vector argument `dispf`:

1. By vector of integers, when the dispersion functions are programmed in C (their source is in the file `RCALI/src/functions.cc`) and compiled. 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 GeneSys - see [CCDM01a] and [CCDM01b]), 3 for the constant function, 4 for an anisotropic version of the dispersal of yellow rust of wheat defined in Soubeyrand and all - see [Pap11], 5 for a discontinuous function. Details and instructions to modify them are given in the online help of **califlopp**.

2. By R functions. The user defines the dispersal functions in R functions.

8.2 The parameters

Parameters can be described in the list argument `param`. Default values are provided. Details and default values can be found in the online help of **califlopp**. Its components are:

- **input** The format of the polygons-file: 1 or 2 (see 7.2).
- **delim** Character separator between values in the polygons-file.
- **method** String equal to `cub` for cubature method, `grid` for the grid method.
- **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.
- **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 ist dispersion function is calculated between centroids only.

- **poly** Required pairs of polygons.
- **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).
- **output** The required output on the screen (see 9.1)
- **verbose** TRUE, if output is required about polygons convexity and landscape translation.
- **warn.poly** TRUE, if output is required about polygons simplification.
- **warn.conv** TRUE, if output is required when cubature convergence is not reached.

When the method is `cub` (cubature), additional components may be given. They are all vectors of length equal to the number of required functions.

- **maxpts** Maximal number of evaluation points required for each function.
- **reler** Relative error required for each function.
- **abser** Absolute error required for each function.
- **tz** Mode of triangulation for the cubature method for each function.

When the method is `grid` (evaluation on a grid), additional components may be given:

- **seed** Seed of the random generator.
- **step** Steps of the grid
- **nr** Maximal number of replications or grids.

Chapter 9

Output

9.1 Screen output depend on the parameter "output"

Screen output depend on the component `output` of the argument `param` of the main function of `RCALI`, **califlopp**. Its default value is the constant `DEFAULT_OUTPUT` set in the file `src/caliconfig.h`. For each pair of polygons:

- When `output`= 1 or `DEFAULT_OUTPUT`=ALL, output are:
  - With the grid method:
    - the integrated flow calculated at each replication,
    - the final value of the integrated flow, its mean per m^2 of both polygons, the standard deviation and the variation coefficient.
  - With the cubature method:
    - the integrated flow, its mean per m^2 of both polygons, the absolute error, the confidence interval and the number of evaluations.
      An asterisk before the absolute error means that the convergence has not been reached with the required precision.
  - With both methods, the areas of the polygons.
- When `output`= 2 or `DEFAULT_OUTPUT`=LIGHT, an iteration number (starting from 1) is only the output.
- When `output`= 3 or `DEFAULT_OUTPUT`=FLOW, output consist of: the integrated flow and its mean per m^2 of both polygons.
- When `output`= 0 or `DEFAULT_OUTPUT`=NOTHING: nothing is written.

Note: When all the pairs of polygons are treated, only $(npoly*(npoly+1))/2$ results are displayed, where *npoly* is the number of polygons, unless the component `send.and.receive` of the argument `param` of the R-function **califlopp** is TRUE.

9.2 The result file

When the argument `resfile` of the RCALI-function **califlopp** is set, a file is created. It contains part or all of the results.

On the result-file:

- The values are separated by tabulates.
- The first line contains: `"npoly:"`, `"input-file:"`, `"nfunc:"`, `"method:"`, each of these identifiers followed by the actual values (`"method:"` is followed by either `"cubature"` or `"grid"`).
  When the method is grid, the remaining of the line is `"stepx:"`, followed by the x-axis step and `"stepy:"`, followed by the y-axis step.
- On each of the following lines, the results for a couple of polygons are written:
  - the identifiers of both polygons;
  - for each dispersal function, the integrated flow divided by the area of the second polygon;
  - the areas of both polygons.

  When the constant OUTPUT_FILE_FORMAT[1]=LIGHT, the remaining of the line is:
  - When the method is cubature: For each dispersal function,
    1. the integrated flow,
    2. the lower and upper bounds of the confidence interval,
    3. the absolute error,
    4. the number of evaluations.
  - When the method is grid, for each dispersal function,
    1. the integrated flow,
    2. the standard deviation,

Examples can be found in files suffixed by `.res`, in the subdirectories of the `examples` directory.

9.3 Error treatment

When an error is encountered, an explicit message is issued on the standard error unit (the screen, by default). Some types of errors are treated specifically:

- **Error in a polygon:**
  A polygon is considered as not valid when it cannot be split into convex subpolygons, i.e when it has too many obtuse angles. The treatment depends on the constant `ERR_POLY`[2]. When `ERR_POLY` is null, an error message is issued, and the polygon is ignored. Otherwise, the error is fatal.
- **Memory allocation problem, Overflow and Range Error:** An error message is issued and execution stops and returns a negative value.

[1] Constant set in the file `src/caliconfig.h`.
[2] Constant set in the file `src/caliconfig.h`

Chapter 10

Example

10.1 The polygons-file

The polygons file is named `poly.txt`. It is in format 1 (see 7.2) and the separator character is the blank character. Its first lines[1] are:

```
66
1 540139  540116 540261    540274
1 1794900   1795000   1795000  1794920
2  540378 540467 540453 540373 540374
2  1795000 1795000 1794850 1794850 1794890
```

10.2 Calculation by the cubature method

Only one result is calculated here: the integrated flow from the polygon 66 to itself, by the cubature method. Thresholds are required for the relative errors: 1.0e-4 for function 1 (pollen flow) and 1.0e-3 for function 2 (seed flow). The other parameters are let to their default values. The argument `param` of the R function **califlopp** is:

```
> library("RCALI")
> param <- list(input=1, delim=' ',
+          reler=c(1.0e-4, 1.0e-3),
+          poly=c(66,66))
> file <- paste(system.file("extdata", package = "RCALI"),
+           "poly.txt", sep='/')
>  califlopp(file=file, param=param)


CaliFloPP -  Copyright (c) 2007 - INRA

Number of polygons: 66
--------------------

Parameters:
-----------
```

[1] The file `poly.txt` can be found in the directory `extdata` of the package.

```
verbose: 0
output: 1
scale: 10
maximal dispersion distances for each function: 0 21
minimal dispersion distances for each function: 100 0
(the dispersion is calculated between centroids,
 for distances beyond these values)
method:cubature
function 1: relative precision = 0.0001, absolute precision = 0.001
            maximal number of evaluations points fixed to 100000
function 2: relative precision = 0.001, absolute precision = 0.001
            maximal number of evaluations points fixed to 100000
mode of triangulation: 0 1

Polygons  66, 66
-------------------

Elapsed real time in integration: 0 seconds

Integrated flow for function 1:
 mean: 6117.53 mean/area1: 0.942609 mean/area2: 0.942609
 absolute error: 0.61155 relative error: 9.99668e-05
 confidence interval: [6116.92, 6118.15]
 nb. evaluations: 70448

Integrated flow for function 2:
 mean: 6403.84 mean/area1: 0.986724 mean/area2: 0.986724
 absolute error: 5.91296 relative error: 0.000923345
 confidence interval: [6397.93, 6409.75]
 nb. evaluations: 14726

area1: 6490 area2: 6490


Total elapsed real time in integration: 0 seconds (0.000000 minutes)
```

10.3 Calculation by the grid method

The parameter-file, `exg2.param`, is:

```
> param <- list(input=1, delim=' ',
+          method="grid",
+          output=3,
+           nr=20,
+           step=c(0.25, 0.25),
+          poly=c(66,66))
>  califlopp(file=file, param=param)
```

```
CaliFloPP -  Copyright (c) 2007 - INRA

Number of polygons: 66
--------------------

Parameters:
-----------
verbose: 0
output: 3
scale: 10
maximal dispersion distances for each function: 0 21
minimal dispersion distances for each function: 100 0
(the dispersion is calculated between centroids,
 for distances beyond these values)
method:grid
seed: 1
x-axis step: 0.25 m.
y-axis step: 0.25 m.
number of estimations: 20

Polygons  66, 66
--------------------

Integrated flow for function 1:
 mean: 6117.35 mean/area1: 0.942581 mean/area2: 0.942581

Integrated flow for function 2:
 mean: 6403.98 mean/area1: 0.986745 mean/area2: 0.986745



Total elapsed real time in integration: 41 seconds (0.000683 minutes)
```

Part V

Main steps of the programme

The main steps of the programme are shortly sketched here. The user is invited to read this Section for a better understanding of the input, output, warning and error messages.

10.4 Preprocessing on the polygons

1. The coordinates are multiplied by `SCALE`[2], a multiple of ten, and then truncated to integers. For example, 2.986 is considered as 2 m if `SCALE`= 1, and as 298 cm if `SCALE`= 100.

2. The landscape is relocated so that the minimal x-coordinate (y-coordinate respect.) is one,

   - when a x-coordinate (y-coordinate respect.), after multiplication by `SCALE`, is greater than `SAFE`[2]
   - when it is null or less than zero,
   - systematically when `TRANSLATE`[2] = 1.

3. Simplification of the polygons: the aligned[3] or too close vertices[4], as well as the sharp spikes[5] are removed from the polygons.

4. The areas and the centroids of the polygons are calculated.

5. Convex subpolygons are created.

10.5 Steps for each pair of convex polygons

For each pair of convex polygons (P_1, P_2), the steps are:

- When the minimal distance between the polygons P_1 and P_2 is greater than a given threshold, DP[6], the dispersal function is calculated between the centroids of these polygons, and the result is multiplied by the product of their areas.

- When this distance is greater than the threshold DZ[6], the dispersal is automatically set to zero.

- Otherwise, for each pair of convex subpolygons in P_1 and P_2, the Minkowski sum is calculated and the flow is estimated by the result of an integration on all the Minkowski sums. The integrand is the product of the individual dispersal function by the area of the intersection between the first subpolygon and a translation of the second one in the pair.

[2] `SCALE`, `SAFE`,`DISTP` and `TRANSLATE` are constants set in the file `src/caliconfig.h`.

[3] When the arccosinus of the angle built by three successive vertices is near to π, the vertices are considered as aligned and the middle one is suppressed.

[4] When the distance between two sucessive vertices is less than or equal to `DISTP`, the second vertex is suppressed.

[5] When the arccosinus of the angle built by three successive vertices is near to zero, it is supposed that the three vertices draw a sharp spike, and the second one is suppressed.

[6] Thresholds are set in the file `src/caliconfig.h`; they are different for each dispersal function. DP constants are the distances beyond which calculation is made between centroids. DZ constants are the distances beyond which dispersal is considered as null.

10.6 Integration methods

Two integration methods are implemented (See details in Section 4):

- The **grid method** :
  Integration is made by discretisation of the Minkowski sum on regular grids of points. Several grids of regularly spaced points are generated, each one randomly shifted from the origin. The successive results can be considered as replications.

  The iterative process stops when the number of replications is reached.

- The **cubature method**:
  This method is a numerical adaptive cubature method over triangles.
  The absolute and relative precisions can be controlled, as well as the maximal number of evaluations.

10.7 Final results

With the grid method, in addition to the mean of the integrated flow over the replications, the coefficient of variation and the standard deviation are calculated.
With the cubature method, in addition to the integrated flow, the absolute precision and a confidence interval are calculated.

10.8 The individual dispersal functions

In the deliverable, some individual dispersal functions are defined. We described here the first two ones. See the comments in the file `src/functions.cpp` for the subsequent ones. These functions may be customized according to the needs. See paragraph 10.8.3.

The user has also the possibility of coding the individual dispersal functions in R, but timing performance can be affected: see the help-file of the `RCALI` function `califlopp`.

10.8.1 Individual dispersal function of oilseed rape pollen

The first function is the oilseed rape pollen individual dispersal function described by Étienne Klein [KLP+06] until 50 m and by Céline Devaux for distances beyond 50 m [DLAK06]:

$$\phi(t) = \begin{cases} d + er + fr^2 & \text{when } 0 \leq r \leq 1,5 \\ \frac{b}{1+r^c/a} & \text{when } 50 \leq r < 1,5 \\ [\frac{b}{1+h^c/a}/(1+h)^g] * (1+r)^g & \text{when } r > 50 \end{cases} , r = \| t \| . \tag{10.1}$$

with $a = 3.80$, $b = 0.03985$, $c = 3.12$, $d = 0.340$, $e = -0.405$, $f = 0.128$, $g = -2.29$[7], $h = 50$ and with t the distance between the source and target points. Distances are in meters. See Fig. 10.1.

For this function, the threshold beyond which the flow is calculated between centroids only, is 100 m.

[7] According to N. Colbach ([CCDM01a] and [CCDM01b]), g can vary between $-2,14$ and $-2,56$.

10.8.2 Individual dispersal function of oilseed rape seeds

The second function is the oilseed rape seed individual dispersal function proposed by Nathalie Colbach [CCDM01b]:

$$\phi(t) = \begin{cases} \dfrac{b \times c \times r^(c-2) \times \exp(-b \times r^c)}{2.0 * \pi} \end{cases}, \quad r = \|t\|. \tag{10.2}$$

with $b = 1.38930$, $c = 2.08686$ and with t the distance between the source and the target points. Distances are in meter. See Fig. 10.1.

For this function, the threshold beyond which the flow is supposed to be null is 21 m.

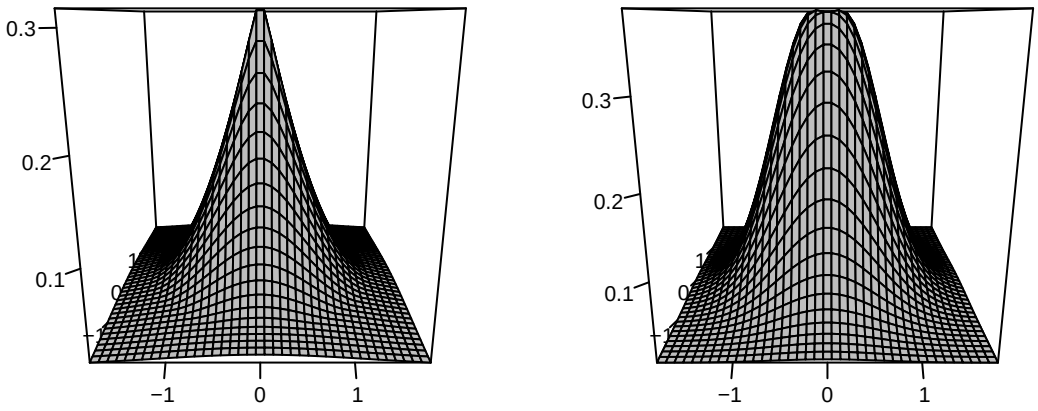


Figure 10.1: Individual Dispersal Function of Pollen (on the left) and Oilseed (on the right) Rape Seeds.

10.8.3 How to modify or add individual dispersal functions

The individual dispersal functions are coded in the C file `src/functions.cpp`. To modify them, change the formulae expressions in the source code. Don't forget they must be smooth functions. Change also the DP* and DZ* constants in the file `src/caliconfig.h`, i.e the thresholds for calculating dispersal between centroids only and for considering that dispersal is zero, respectively.

In the delivered package, up to five dispersal functions can be defined. By default, the DEFAULT_NFUNCTIONS[8] first functions are considered only.

After alteration of the source code, recompilation is required, typically by using the standard R command: <tt>R CMD build RCALI</tt> on top of your <tt>RCALI</tt> directory (see 6).

[8] DEFAULT_NFUNCTIONS is defined the file `src/caliconfig.h`.

Part VI

A Small Glossary

- $\widehat{\mathcal{F}}$ **: integrated flow in the cubature method:**
  For each pair of convex subpolygons of a given pair of polygons, the Minkowski sum is calculated and triangulated. $\widehat{\mathcal{F}}$ is the result of an adaptive cubature integration method on all the triangles.
  Special cases: when threshold distances have been set, $\widehat{\mathcal{F}}$ is automatically set to zero, or calculated between centroids only, beyond these distances (see 10.5).

- $\widehat{\mathcal{F}}$ **: mean of the integrated flows in the grid method:**
  $\widehat{\mathcal{F}}$ is the sum over all the pairs of convex subpolygons, of the mean of the calculated values over the replications, i.e: $\widehat{\mathcal{F}} = \sum_{i=1}^{i=k} \sum_{j=1}^{j=r} \widehat{f}_{i,j}/r$, where k is the number of pairs of convex subpolygons and $\widehat{f}_{i,j}$ is the calculated integrated flow between one such pair of subpolygons, at the j^{th} grid generation.
  Special cases: when threshold distances have been set, $\widehat{\mathcal{F}}$ is automatically set to zero, or calculated between centroids only, beyond these distances (see 10.5).

- **Coefficient of variation in the grid method:**
  $CV =$*standard deviation*$/$*mean*.

- **Confidence Interval in the cubature method:**
  $IC = [\widehat{\mathcal{F}} - absolute\ error, \widehat{\mathcal{F}} + absolute\ error]$

- **Minkowski sum of two polygons A and B.**
  This sum is a polygon, denoted by $A \oplus B$. It is the set of points p, such that: $p = y - x, x \in A, y \in B$, i.e the set of points covered by B when a vertex of B is moved inside A.

- **Relative error in the cubature method:**
  $relative\ error = absolute\ error/result$

- **Standard deviation in the grid method:**
  $standard\ deviation = \sqrt{\sum_{j=1}^{j=r}(\hat{f}_{.j} - \hat{f}_{..})^2/(r-1)}$, where r is the number of replications, i.e the number of grid generations, $\hat{f}_{.j}$ the result of the grid integration at replication j, and $\hat{f}_{..}$ the mean result over the j replications.

Part VII

Developer Guide

Chapter 11

Implementation

11.1 Programme steps

The main steps have been described from the user's point of view in Section V. Here, they are sketched at the programming level.

11.1.1 Preprocessing

Function `ReadPoly`:

1. Read the polygons file and convert the coordinates into integers (->[1]`ReadCoord`);
2. Relocate the landscape (-> `TranslatedParcel`);
3. For each polygon:
   (a) remove aligned vertices and vertices forming a sharp spike;
   (b) create convex subpolygons: first, determine the essential diagonals, i.e the diagonals which split the polygon into convex parts (-> `Triangulate`), and then determine and store the convex subpolygons ((-> `HMAlgor`);
   (c) compute areas (-> `Area2`).

11.1.2 Calculation steps

The function `suite` pilots the calculations process. It realizes the following tasks:

1. Create an object, `methode`, of class `methodGrid` or `methodAdapt`.
2. Call its method `VerifArgu` to verify its attributes.
3. Pilot the loop on the required pairs of polygons: for each of them, call the function `go` which invokes the methods `CalcR` on `methode` to perform the calculations, `Print` to output the results on the screen, and `PrintFic` to output them on a result-file.

11.1.3 Calculation by the grid method

The method `CalcR` of the class `methodGrid` is summarized by the following algorithm:
(The names of the devoted functions are between square brackets.)

[1] Arrow is for a call to a subroutine.

Algorithm 7: methodGrid::CalcR

Data: A pair of polygons A and B, nf individual dispersal functions, user-defined parameters.
Result: Dispersal estimations from A to B by the grid method.

```
1  Calculation of mindist, the minimal distance between the polygons.
2  foreach dispersal function φ do
3      if mindist ≥ threshold for function annulment then
4          result = 0
5      end
6      if mindist ≥ threshold for calculation between centroids then
7          t = distance between centroids; result = φ(t) × area(A) × area(B)
8      end
9  end
   // In the other cases:  one function at least should be integrated
10 foreach pair of convex subpolygons A_i and B_j in A and B do
11     Compute and store their Minkowski sum: sommeM_ij [ SommeMinkowski];
12 end
13 foreach replication do
14     foreach pair of convex subpolygons A_i and B_j do
15         Grid generation [ Integration]:
16         foreach point t of the grid do
17             if t ∈ sommeM_ij [ InPolyConvex] then
18                 ar = area(A_i ∩ (B_j − t)); [ ConvexIntersect]
19                 foreach dispersal function φ to be integrated do
20                     result+ = φ(t) × ar;
21                 end
22             end
23         end
24     end
25 end
26 Compute the final results: means over the replications, standard deviation.
```

11.1.4 Calculation by the cubature method

The algorithm 8 summarizes the tasks realized by the method `CalcR` of the `methodAdapt` class (the names of the devoted functions are between square brackets).

$octo_\phi$ stands for the smallest octogon centered in (0,0) which includes the circle of radius equal to the distance beyond which the dispersal function ϕ becomes nul; $octo_\phi$ is an attribute of the `methodAdapt` class and is created by the class constructor.

Algorithm 8: methodAdapt::CalcR

Data: A pair of polygons A and B, nf individual dispersal functions, user-defined parameters.
Result: Dispersal estimations from A to B by the cubature method.

```
1  Calculation of mindist, the minimal distance between the polygons.
2  foreach dispersal function φ do
3      if mindist ≥ threshold for function annulment then
4          result = 0
5      end
6      if mindist ≥ threshold for calculation between centroids then
7          t = distance between centroids; result = φ(t) × area(A) × area(B)
8      end
       // In the other cases:  integration should be done
9      stvertce = ∅; result = 0;
10     foreach pair of convex subpolygons A_i and B_j of A and B do
11         Compute and store the Minkowski sum: sommeM_ij; [ SommeMinkowski]
12         if dispersal has no limit then
13             S = sommeM_ij;
14         else
               // no dispersal beyond a given distance
15             S = sommeM_ij ∩ octo_φ [ TConvexIntersect];
16         end
17         if S ≠ ∅ then
18             if (0,0) ∈ S [ InPolyConvex] then
19                 stvertce+ = triangulation of S from (0,0) [ Triangulate0];
20             else
21                 stvertce+ = triangulation of S from any vertex [ Triangulate] ;
22             end
23         end
24     end
       // End of the loop over A_i, B_j
25     Integration on the stvertce triangles for function φ [ Adapt::Integration];
26 end
   // End of the loop over the functions
```

11.2 Data structures

The main data structures are described in the following array:

Name	Dimension	Type	Content
Poly	npoly, nspoly, nvert, DIM	tPolygoni (integer)	Polygons coordinates, counterclockwise sorted, non-closed polygons
ni	npoly, nspoly	integer	$ni_{i,j}$: number of vertices in the subpolygon j of the polygon i
a	npoly	integer	a_i: number of convex subpolygons of the polygon i
vertices	structure	tVertex (integer)	Linked list of the ordered vertices of a polygon. Each element contains:
	V[DIM]	integer	- coordinates of a vertex,
	next		- pointer to the next vertex or to the head of the list if none,
	prev		- pointer to the preceding vertex or to the head of the list if none,
	vnum	integer	- vertex indices
sommeM	nvert, DIM	tPolygoni (integer)	Minkowski Sum
intersection	id. as vertices	tdVertex (real)	Intersection of polygons

npoly: number of polygons
nspoly: number of convex subpolygons in a polygon
nvert: number of vertices in a polygon
DIM: space dimension (here =2).

11.3 Functions list

Some important functions are listed here.
(Words in *italic* refer to data structures.)

Function name	File name	Fonction
ecrmess	util.cpp	Error message output and return to the calling programme
libMem	util.cpp	Memory de-allocation
califlopp_sd	fluxsd.cpp	Pilot
suite	go.cpp	Pilot the loop over the pairs of polygons
go	go.cpp	Pilot the treatment of one pair of polygons
read1Poly, read2Poly	read1Poly.cpp	Read the coordinates in format 1 and 2, resp.
ReadCoord	readPoly.cpp	Read the polygons-file; verify the coordinates; scale multiplication of the coordinates
ReadVertices	readPoly.cpp	Create *vertices*
ReadPoly	readPoly.cpp	Create *Poly* with: - aligned vertices removal - non-convex polygons splitting - areas computation
TranslateParcel	readPoly.cpp	Relocation of the polygons
ConvexIntersect	intersection.cpp	Convexity test and creation of *intersection*
Triangulate, HMAlgor, Area2	geom.cpp	Programmes of geometric computation
SommeMinkowski	zoneintegration.cpp	Compute *sommeM*
genrand_real2	mt19937ar.cpp	Random numbers generation[1]
f0,f1,fX	functions.cpp	Individual dispersal functions
f_	methodAdapt.cpp	The integrand for cubature method
Integration	methodGrid.cpp, methodAdapt.cpp	Integration
CalcR	methodGrid.cpp, methodAdapt.cpp	Compute one result
Print, PrintFic	methodGrid.cpp, methodAdapt.cpp	Output results on screen and on file, resp.

See reference [MN98].

Error codes: The list and meaning of the errors codes can be found in the file `calierror.h`.

Chapter 12

How to Modify

It may be useful to know how to modify some parts. Among them:

- The individual dispersal functions: see Section 10.8.3.
- The format of the polygons-file: see the files `read1Poly.cpp` and `read2Poly.cpp`.
- In interactive mode, the dialogue with the user: see the file `fluxad.cpp`, and the `ReadArgu` method in the files `methodGrid.cpp` and `methodAdapt.cpp`.
- Screen output: see the file `go.cpp`, and the `Print` method in the files `methodGrid.cpp` and `methodAdapt.cpp`.
- File output: output on the result-file depend on the constant OUTPUT_FORMAT set in the file `caliconfig.h`. They are carried on by the `suite` function and by the `PrintFic` methods in the files `methodGrid.cpp` and `methodAdapt.cpp`.
- Erroneous polygons treatment: the identifiers of the erroneous polygons are made negative by the `ReadPoly` function. Their treatment depends on the constant ERR_POLY set in the file `caliconfig.h`.
- Memory allocation: memory allocation is made via the macros CREER and NEW and deallocation via the macros DETRU and FREE; they are coded in the file `calimacros.h`.
- Random number generator: see the variable COPTIONS2 in the file `obj/subdir.mk` and the corresponding code in `methodGrid.cpp`.
- Computing time measurement: see the file `timing.cpp`.
- Estimation of several integrals by the cubature method: this possibility can be added when several integrals have enough similarity. Estimation of all of them can be made in one call. It is less time consuming. For that, modify:
  - the function `f_` in the file `methodAdapt.cpp`: the vector `funvls` should contain as many results as integrals on output.
  - the function `CalcR` in the file `methodAdapt.cpp`: the first argument of the `Adapt` constructor should be equal to the number of integrals.
  - the output operator in the file `adapt/Adapt.h`: it should print as many results and absolute errors as integrals.

After alteration of the source code, recompilation is required: see paragaph 6.

Part VIII

Bibliography

Bibliography

[BE92] Jarle Berntsen and Terje O. Espelid. Algorithm 706, DCUTRI: An algorithm for adaptative cubature over a collection of triangles. *ACM Transactions on Mathematical Software*, 18(3):329–342, 1992. 23, 25

[CCDM01a] N. Colbach, C. Clermont-Dauphin, and J.M. Meynard. 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. 9, 53, 64

[CCDM01b] N. Colbach, C. Clermont-Dauphin, and J.M. Meynard. Genesys: a model of the influence of cropping system on gene escape from herbicide tolerant rapeseed crops to rape volunteers. ii. genetic exchanges among volunteer and cropped populations in a small region. *Agriculture, Ecosystems and Environnement*, 83:255–270, 2001. 9, 53, 64, 65

[DLAK06] C. Devaux, C. Lavigne, F. Austerlitz, and E.K. Klein. Modelling and estimating pollen movement in oilseed rape (*Brassica napus*) at the landscape scale using genetic markers. *Molecular Ecology*, 2006. 64

[KLP+06] E.K. Klein, C. Lavigne, H. Picault, M. Renard, and P.H. Gouyon. Pollen dispersal of oilseed rape: estimation of the dispersal function and effects of field dimensions. *Applied Ecology*, 43:1141–1151, 2006. 64

[LKV+98] C. Lavigne, E.K. Klein, P. Vallée, J. Pierre, B. Godelle, and M. Renard. A pollen-dispersal experiment with transgenic oilseed rape. estimation of the average pollen dispersal of an individual plant within a field. *Theoretical and Applied Genetics*, 96:886–896, 1998. 9

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998. 77

[NAG] *The NAG Fortran Library.*
URL: `http://nag.co.uk/numeric/fl/FLdescription.asp`. 16

[O'R98] Joseph O'Rourke. *Computational Geometry in C.* Cambridge University Press, second edition edition, 1998. Supplementary material `http://maven.smith.edu/~orourke/books/compgeom.html`. 17, 19, 21

[Pap11] J. Papaix. Structure du paysage agricole et risque épidémique, une approche démo-génétique. Master's thesis, ABIES, 2011. 53

[Y.05] Fu Y. Méthodes d'intégration pour le calcul des flux de gènes entre parcelles dans un paysage agricole. Master's thesis, École Centrale de Nantes, 2005. Mémoire de stage, Unité INRA MIA, Jouy-en-Josas. 3

The red figures at the end of each reference refers to the pages in the document.