Mapping Averaged Pairwise Information (MAPI)

A new exploratory tool to uncover spatial structure

MAPI User Manual

Sylvain Piry[1] and Karine Berthier[2]

[1]*INRA, UMR 1062, Centre de Biologie pour la Gestion des Populations, 755 avenue du campus Agropolis, F-34988 Montferrier sur Lez, France (piry@supagro.inra.fr)*
[2]*INRA, UR 407 Pathologie Végétale, F-84143 Montfavet, France*

Version: 1.0.1 rev. 146 -- Thursday 7th July, 2016

Abstract

Aims: Exploratory and method providing graphical representations summarizing the spatial variation of pairwise metrics (*eg.* distance, similarity coefficient, ...) computed between georeferenced samples.

Datasets: Samples with geographic coordinates, vectorized matrix of the pairwise metric computed between sample pairs. Better the spatial coverage, better the results. Poor-quality datasets (missing coordinates, poor spatial coverage, strongly spatially biased sampling, etc.) are likely to produce poor results.

Software: MAPI is provided as an extension for PostgreSQL ≥9.1 with Postgis ≥2.0. Cartographic representation of MAPI results can be done under GIS software such as QGis ≥2.10. Please, refer to specific documentation for the installation and use of these software. R language (3.2, tested), along with libraries RPostgreSQL (0.4, tested) and data.table (1.9.6, tested) provide a scriptable command-line interface for data transfer and computation submissions. PgAdmin3, provided with PostgreSQL for communicating with the server through a Graphical User Interface, is the dedicated companion tool for databases creation and management.

Principle: Each pair of samples is linked by an ellipse (represented as a geographical polygon), for which the foci are the two sample locations being connected. The eccentricity value for ellipses can be chosen by users (default value 0.975). The attributes of an ellipse are: its area, the geographical distance and the value of the metric computed between the sample being connected. Once computed, the ellipses are overlayed on an hexagonal grid emcompassing the study area. Finally, each cell of the grid receives the weighted mean of the metric values attributed to the ellipses spatially intersecting above the cell ; the metric value being weighted by the invert of the ellipse area. A built-in permutation procedure allows to identify significant areas of high (dis-)continuity. A correction to account for multiple testings is also implemented.

Distribution: MAPI is free, open-source, and distributed as an extension for PostgreSQL which has to be installed. Once installed, databases can access the MAPI functions by enabling the "mapi" extension in each database. The MAPI library is licenced under the Creative Commons 3 licence[1]. It can be downloaded from our website: https://www1.montpellier.inra.fr/CBGP/software/MAPI/.

Reference: If you use MAPI on your datasets, please read and cite:
Piry, S., Chapuis, M.-P., Gauffre, B., Papaïx, J., Cruaud, A., and Berthier, K. (2016). Mapping Averaged Pairwise Information (MAPI): A new exploratory tool to uncover spatial structure. *Methods in Ecology and Evolution (accepted).*

[1]Licence full text available on https://creativecommons.org/licenses/by-nc-sa/3.0/deed.en_GB

Contents

List of Tables

1 Principle

1.1 Why a database?

PostgreSQL & PostGIS

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX, and Windows. PostgreSQL is developed by the PostgreSQL Global Development Group, a diverse group of many companies and individual contributors. It is free and open-source software, released under the terms of the PostgreSQL License, a permissive free-software license.
http://www.postgresql.org/about/

PostGIS adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS). PostGIS follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC).
http://postgis.refractions.net/

First, because it's worth it, and because I like it ☺.

There are good reasons for this choice, such as an excellent memory management and a real efficiency with huge datasets. MAPI computation does "crossings" (ie. spatial joins) between thousands of cells and hundred of thousands network edges defined as polygons, so a "in-memory" implementation would probably overflow your RAM. Indeed, such spatial joins in R are much slower than done by PostGIS...

Within PostgreSQL, optimization balances efficiency *vs.* disk-caching without explicit code. Therefore, you can run big computations, even on a medium-sized computer such as a (good) laptop, without crashing your system (but, anyway, big datasets may need a powerful server). Furthermore, database software are quite resilients to failures and data can be easily backuped for safety.

Then, PostGIS extension provides easy-to-use functions for manipulating spatial data whatever their geometry type (points, lines, polygons). It's a real pleasure to handle geographical objects in such a system. SQL is not so complicated to learn and allows to write complex queries with only a few lines, allowing then a more efficient programming.

Many datasets in our labs are now stored in *ad-hoc* PostgreSQL databases. Data are shared between users, each one contributing to the community.

Fields definition (text, integers, floats, ...) along with relations between tables ensure good data quality. Despite the complexity, such an implementation is much more reliable than any spreadsheet!

Finally, it's very easy to import or export map data from/to GIS or R. This system is very open, allowing collective work and data exchanges.

I guess that when you will get used to PostgreSQL you'll like it and it will become a major tool for you too!

1.2 MAPI framework

The MAPI method relies on spatial join between a hexagonal[2] grid (computed on the fly) and ellipses connecting all pairs of distinct samples between which a pairwise metric of interest has been computed. Each ellipse is built such as its foci are localized on sample coordinates.

In fact, ellipses are polygons with 32 segments approaching an elliptical shape for which the eccentricity value can be set by user (default 0.975).

Furthermore, the location of the sampling points can be "blurred" by considering an error circle with a given radius.

The final shape of the polygon is then the convex hull of two objects: the ellipse *sensu stricto* and the oval joining the two point-centered error circles. Therefore, depending on the values chosen for ellipse's eccentricity and error-circle's radius, the shape of the pairwise polygons can range from a thin oval to a fat ellipse (figure 2 on page 6).

[2] See Sahr (2011) for further informations about hexagonal pixels.

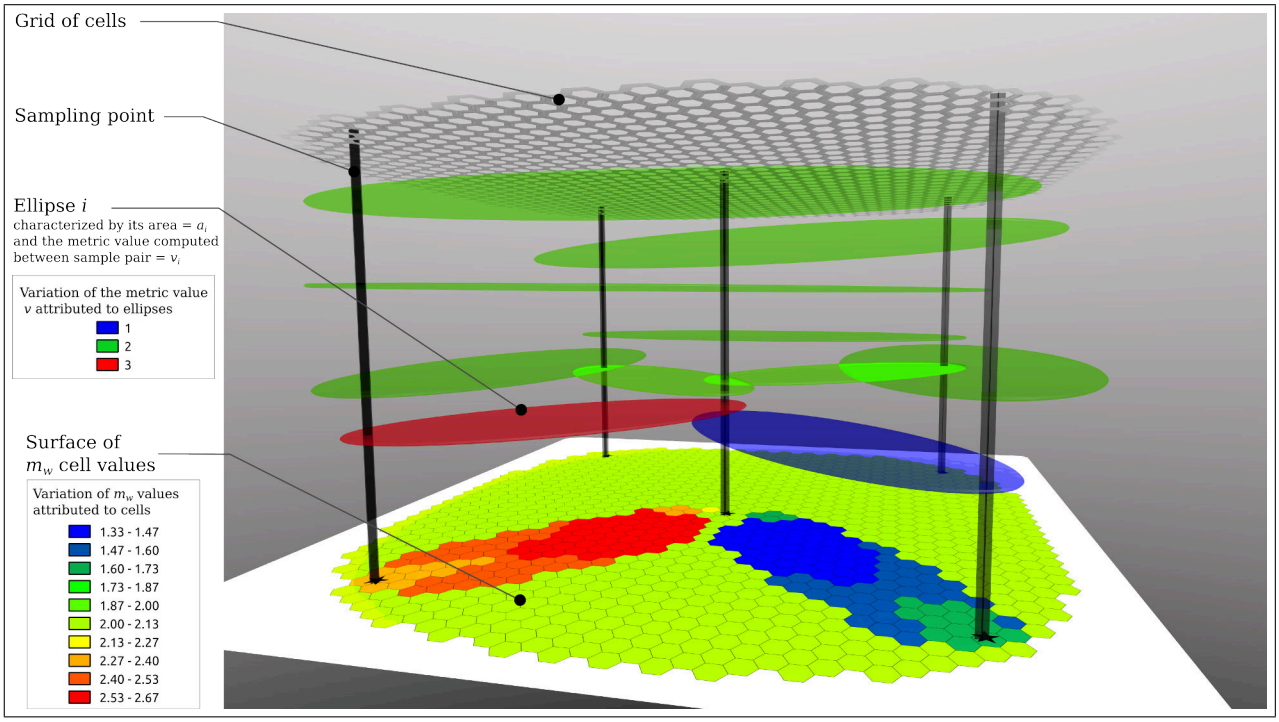


Figure 1: Principle of the MAPI method

The default values for ellipse eccentricity and error circle radius are 0.975 and 10m (GPS error), respectively. Ellipse area depends on the eccentricity value and the distance between the foci (i.e. the sampling points connected).

The weight w_i of an elliptical polygon i (ellipse, buffer, see figure 2 on the next page) is computed as the inverse of its area a_i:

$$w_i = \frac{1}{a_i}$$

For each cell, the sum of weights s_w of the n ellipses geographically intersecting (ie. located above) the cell is computed and stored:

$$s_w = \sum_{i=1}^{n} w_i$$

At the same time, the cells of the grid receive the weighted mean m_w of n ellipse values v_i (figure 1).

$$m_w = \frac{1}{s_w} \sum_{i=1}^{n} w_i \, v_i$$

In a second step, the weighted standard deviation is computed for each cell as:

$$d_w = \sqrt{\frac{1}{s_w} \sum_{i=1}^{n} w_i \, (m_w - v_i)^2}$$

Changing the eccentricty value impacts the relative weights of ellipses in the analysis, in relation with the geographical distance.

Ellipses

In mathematics, an ellipse is a curve on a plane surrounding two focal points (also called foci) such that the sum of the distances to the two focal points is constant for every point on the curve. As such, it is a generalization of a circle, which is a special type of an ellipse that has both focal points at the same location. The shape of an ellipse (how 'flattened' it is) is represented by its eccentricity, which for an ellipse can be any number from 0 (the limiting case of a circle) to arbitrarily close to but less than 1.
http://en.wikipedia.org/wiki/Ellipse

Weighted arithmetic mean

The weighted arithmetic mean is similar to an ordinary arithmetic mean (the most common type of average), except that instead of each of the data points contributing equally to the final average, some data points contribute more than others. The notion of weighted mean plays a role in descriptive statistics and also occurs in a more general form in several other areas of mathematics. If all the weights are equal, then the weighted mean is the same as the arithmetic mean.
https://en.wikipedia.org/wiki/Weighted_arithmetic_mean

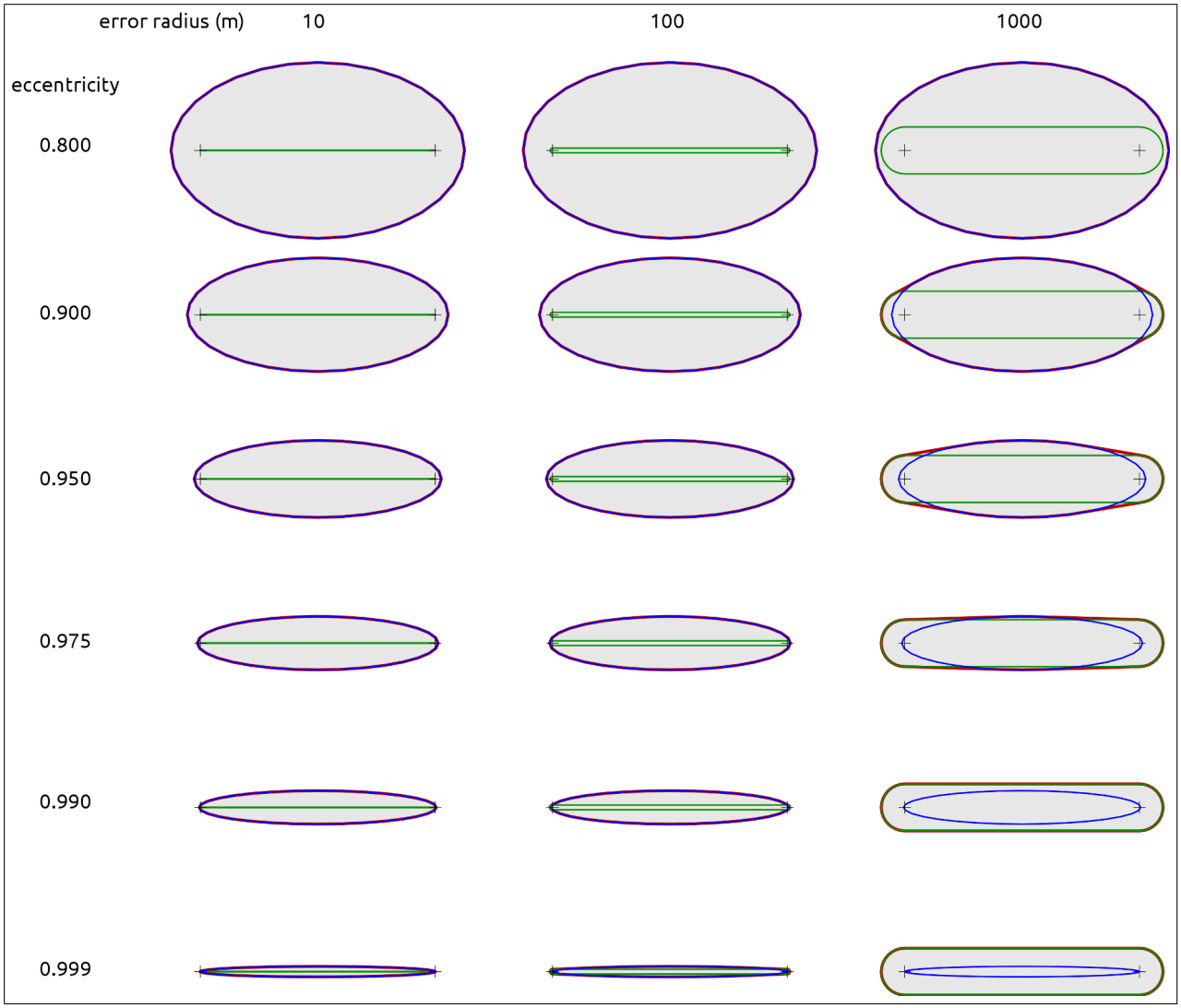


Figure 2: Polygon shape depending on error circle radius, in abscisse, and ellipse's eccentricity, in ordonate. Points (foci) are separated by 10,000 m. Error buffer is drawn as a green line, ellipse in blue, and final shape in gray with red line.

Finally, for each cell, a centile (position in a 1 .. 100 ranking with same number in each class) of the sum-of-weights of ellipses is computed. This information can be used to detect and discard poorly-supported cells.

1.3 Detection of significant areas of (dis-)continuity

In order to discriminate a true signal for geographical variation in the pairwise metric from random noise, we implemented a randomisation procedure.

Sample locations are permuted and the MAPI analysis is applied on the permuted dataset. Resulting cell values are stored, and this process can be repeated n_p times (typically $n_p = 1000$).

The observed cell values are ranked against their null distribution obtained from then n_p permutations. This ranking allows to obtain a lower-tailed test p-value p_i for each cell i: *The probability that a MAPI cell value computed from randomly permuted data is lower than the observed MAPI cell value computed from unpermuted data.* An upper-tailed test p-value can be computed as $(1 - p_i)$.

This process is illustrated figure 3 on the next page.

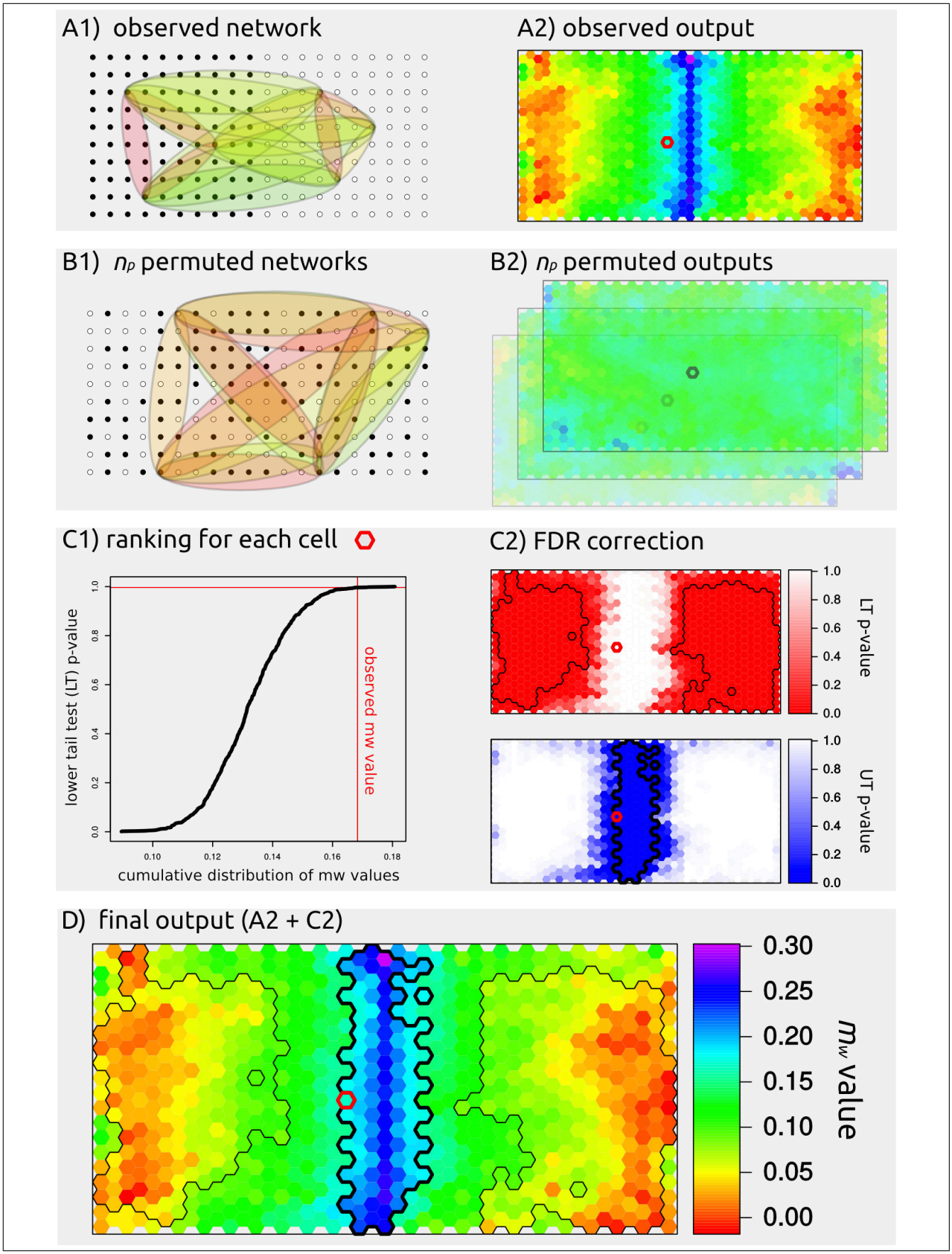


Figure 3: Illustration of MAPI's permutation process, p-values computation and significant areas detection.
A1: Network of a subset of ellipses from observed data. **A2:** MAPI cell values for the observed dataset. **B1:** Network of a subset of ellipses from permuted data. **B2:** Examples of MAPI cell values computed from permuted datasets. **C1:** Example of ranking of an observed cell value against its null distribution and estimation of its lower-tailed test p-value. **C2:** Map of "raw" (uncorrected) lower- (LT) and upper- (UT) tailed test p-values. Significant areas after FDR correction on cell p-values p_i are delineated for each tail by aggregating adjacent significant cells. **D:** Final result (A2 + significant areas from C2).

Finally, the last (independant) step is to aggregate spatially connected cells that exhibit significant lower or, alternatively, higher values. Thresholds for significativity (typically 5%) can be chosen by user.

You may either choose a "raw" thresholding or, which is mostly recommended, apply a "False Discovery Rate" (FDR) algorithm in order to account for multiple testings. By default, the FDR method used in `MAPI_RunAuto` function is the one described in Benjamini and Yekutieli (2001).

1.3.1 Computation time

Computation time varies from a few minutes to several days, depending of your dataset and spatial precision. PostgreSQL is not parallelized yet, so only one core is used for each run. The script provided at the end of this document (200 samples, 855 cells, 1,000 permutations) was executed in less than 20 min on my 2.7 GHz laptop using 1.3 Gio of memory (over 16 Gio). A large dataset (785 samples, 10,130 cells, 1,000 permutations) ran 2 days on a 2.0 GHz server and requested about 10 Gio of memory (over 64 Gio).

The number of ellipses varies as the square of the number of sampling locations, this is the most costful feature. The computation time increases linearily with the number of cells in the grid (but when the number of cells is too high, PostgreSQL "swap" transient memory to the disk and then the computation becomes very slow). On a good server, $\sim$ 100,000 cells appears to be the practical maximum. Note also that wider ellipses (lower eccentricity) need more time to compute as they intersect more cells. For optimization, spatial joins between ellipses and grid are done only once, so each permutation run really faster than the first computation of observed data.

Finally, the complexity of the MAPI process varies roughly in $O(s^2.n_c.n_p)$ with s being the number of distinct sampling locations, n_c the number of cells and n_p the number of permutations.

A special "draft" option allow to move sampling locations to the center of the cell they belong to (see section 6.3 on page 35). Doing this drastically reduce the number of distinct ellipses and then reduce computation time, with, however, a tradeoff on ellipse positions.

1.3.2 Hacking

Feel free to explore the MAPI functions, and - why not? - write your owns for your specific goals. No compilation is needed as plPgSQL is an interpreted language (although a faster weighted-variance computation is optionnaly provided in C language).

For stability reasons, creation of new functions (with your own new names) should be prefered over changes in current names. Don't forget to read the licensing terms before (re-)distribution.

1.4 Mapping results

A MAPI result PostgreSQL table contains for each cell: the geometry (geographical polygon of the cell), the weighted-mean value m_w, the number of ellipses n used for the computation, the sum of weights of those ellipses s_w as well as the centile of this sum-of-weights and, finally, the ranking of the cell value (between 0 and 1) in the permuted distribution (if any). Table 1 on page 10 give details about output tables.

Metadata about the computation (date, time, duration and parameter values) are stored as comments linked to the table. They may be viewed through PgAdmin3 (the database administration tool, provided with PostgreSQL) when clicking on the table name.

Significant areas (cells or group of cells) are stored, also as polygons, in another table ending with "_tails".

A map representing the MAPI results can be drawn using a GIS software, such as QGIS or R (see figure 4 on the facing page), which can directly access the MAPI tables stored in PostgreSQL. A R script along with a step-by-step example using QGis are detailed below.

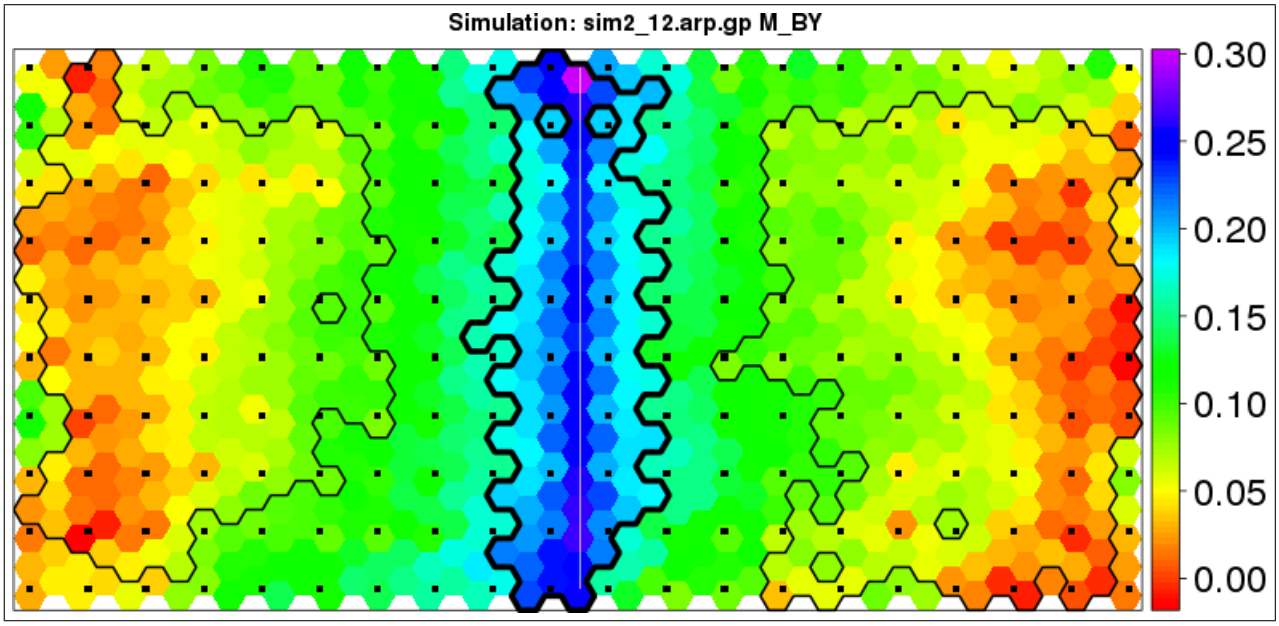


Figure 4: Example of MAPI graphical output produced with R. The simulated genetic barrier is illustrated as a thin white vertical line at the center.

Interpretation of figure 4 : As input data, two panmictic populations ($N_e1 = N_e2 = 100$) were simulated on both sides of a slightly permeable barrier ($N_em = 0.1$) represented here as a thin vertical white line. Individuals were evenly separated by 1,000 (arbitrary) distance units. All genetic distances (a_r in Rousset 2000) were computed between sample pairs using SPAGeDi 1.4 (Hardy and Vekemans, 2002). Sample locations were used to compute both, an hexagonal grid (computed with $\beta = 0.5$ suitable for regular sampling; see below for explainations about the β parameter) and ellipses linking individuals ($eccentricity = 0.975$). Each cell contains the weighted mean of the a_r values attributed to the ellipses intercepting the cell. The MAPI cell values m_w are represented by a color scale.

For the detection of significant areas, the MAPI analysis was applied on 1,000 permutations of samples locations. For each cell, the m_w value from observed data has been ranked against its null distribution obtained from 1,000 permutations. This ranking, between 0 and 1, defines the lower-tailed test p-value.

The False Discovery Rate (FDR) procedure from Benjamini and Yekutieli (2001) was applied on raw p-values with a cut-off value $\alpha = 0.05$ (5%) for the two tailed tests independently.

Finally, according to lower- and upper- tailed tests, significant spatially connected cells were aggregated to delineate highly continuous and discontinuous genetic areas, respectively. On figure 4, continuous and discontinuous areas are delineated by thin and thick black lines, respectively.

The barrier between the two populations appears as a significant discontinuous area, while the two populations are (more or less) detected as significant continuous areas.

Remark:
If the initial pairwise metric is a genetic distance, the lower tail allows to delineate areas where genetic similarity is the highest (lower genetic distance values between samples) while the upper tail allows to delineate areas where genetic differentiation is maximum (greater genetic distance values between samples).
On the other hand, if the metric represents a similarity measure, the lower tail stands for a greater differenciation while the upper tails reveals highest similarities.

Table 1: Field names and contents of MAPI output tables

Main results table

Field name	Field type	Contents
id	serial integer	auto-increment, primary key
cell_gid	integer	id of the cell in the grid
relation_group	text	name of the group, or the pair of groups in alphabetical order
averaged_value	double precision	weighted mean of the cell (m_w)
weighted_stddev	double precision	weighted standard deviation of the cell (d_w)
sum_of_weights	double precision	sum of the weights of the ellipses intersecting the cell (s_w)
sum_of_weights_centile	smallint	centile of the sum of the weights of the ellipses intersecting the cell [1 .. 100]
number_of_ellipses	integer	number of the ellipses intersecting the cell (n)
permuted_values_str	text	comma-concatenated array of the values computed during permutations process (if any, null otherwise) in permutations order
pvalue	double precision	the rank in [0 .. 1] of the cell against its null permuted distribution (p)
geom	geometry(Polygon)	the geometry of the cell in PostGIS format

Tails table

Field name	Field type	Contents
id	serial integer	auto-increment, primary key
relation_group	text	name of the group, or the pair of groups concatenated in alphabetical order
alpha	double precision	the user-defined α value
method	text	name of the user-defined algorithm used: • 'M_raw' when no correction applied, • 'M_BY' for Benjamini-Yekutieli FDR correction
tail	text	the side of the tail (upper or lower)
geom	geometry(Polygon)	the geometry of the polygon in PostGIS format
area	double precision	the area of the polygon in squared map units

Grid table

Field name	Field type	Contents
gid	serial integer	auto-increment, primary key
x_count	integer	rank of the cell along the x-axis
x	double precision	abscissa of the centroid of the cell in map units
y_count	integer	rank of the cell along the y-axis
y	double precision	ordinate of the centroid of the cell in map units
geom	geometry(Polygon)	the geometry of the cell in PostGIS format

2 Now, try it yourself!

2.1 Installation

The MAPI extension can be downloaded from the MAPI website:
http://www1.montpellier.inra.fr/CBGP/software/MAPI/

2.1.1 Linux installation

Note that, at least for Ubuntu 14.04LTS, the package `postgresql-server-dev-9.3` is needed before running the "make install" command[3]. The `postgresql-9.3-postgis-2.1` package have also to be installed. Change the PostgreSQL version number accordingly to your current installation. Create a directory, copy the zip file in this directory, open a terminal in this directory, unzip the mapi file and as administrator ("sudo"), then compile and install the extension:

```
unzip mapi--1.0.1.zip
sudo make install
```

The extension files are now located in `/usr/share/postgresql/9.3/extension` (this may vary according to your own distribution or version) and are available to the database manager.

Note: If your compilation succeeded you will be able to upgrade to 1.0.1c version which includes a faster function for variance computation. To do so (it's optional), run in your database the sql command:
`ALTER EXTENSION mapi UPDATE TO '1.0.1c';`

2.1.2 Windows installation

Open the `mapi--1.0.1_windows.zip` file, then manually copy the files `mapi.control` and `mapi--1.0.1.sql` in the directory `C:\Programs\PostgreSQL\9.3\share\extension\` (or similar one, depending of your PostgreSQL installation).

You may need administrator rights to write in Programs directory; ask your administrator otherwise.

2.1.3 PostgreSQL configuration (for shared servers only)

This section is intended for servers only, not for personnal computer. Within the same computer, no changes in configuration files are necessary.

In order to access PostgreSQL through network connexions (even within the same computer), network connections should be enabled in postgresql.conf and configured in `pg_hba.conf` files.

In file postgresql.conf, uncomment the line `listen_addresses` in the section "CONNECTIONS AND AUTHENTICATION" by removing the # character:

```
...
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'                  # what IP address(es) to listen on;
```

[3]For softwares installation on old Ubuntu 12.10 LTS you may find help at: http://gishouse.blogspot.fr/2013/01/how-to-install-postgresql-91-postgis-20.html

```
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
...
```

If you wish to allow connections through network (ie. "non-local"), then in file `pg_hba.conf`, add filters for allowing connections, such as:

```
...
# Put your actual configuration here
# ----------------------------------
#
# If you want to allow non-local connections, you need to add more
# "host" records.  In that case you will also need to make PostgreSQL
# listen on a non-local interface via the listen_addresses
# configuration parameter, or via the -i or -h command line switches.

host    all     all             192.168.1.0/24                  md5

...
```

The first column contains "host" to tell PostgreSQL that this line applies to external computers. The second column lists the databases opened for this host (here all databases). The third column lists the users allowed to log in through this connection (here all users). The fourth column is a network IP address and netmask for local network or host range filtering. The fifth column is for the authentification method (md5 stands for encrypted password). Be careful not to create security holes in your database by allowing unidentified connections. If you feel doubtful then ask your administrator for advises. Refer to PostgreSQL documentation for further details.

Default PostgreSQL configuration is not fully optimized for MAPI computations. Some "tuning" may reduce computation times and/or disk loads. It may be wise to read some books for tuning; a comprehensive list of book about PostgreSQL is available at: http://www.postgresql.org/docs/books/. Online documentation and manuals in various languages are also available from this website.

2.1.4 Create roles

For security reasons, it is recommended to create a "role" (i.e. an account in PostgreSQL terminology) by user, including yourself, instead of working under "postgres" username. The following steps should allow you to do so on your own computer; for a server ask your administrator! The name of your "role" and your password will be used later for communications with PostgreSQL.

For Windows users: From your "Start" menu run the program PgAdminIII (a blue elephant). Your PgAdminIII setup should already contains a connection to your local server. Double-click on this connection under the "Servers" node. Right-click on "Login Roles" at bottom, then choose "New Login Role" in the menu. Type the new login to be created in "Role name" in the first tab, the password (twice) in the second tab, and check "Can create databases" in the third tab. If you are the manager of this server also check the "Superuser" box in this third tab. Finally, click on the "Validate" button (bottom-right).

Still in pgAdminIII, now configure a new connection with this newly created role: Click on the "plug" button (top-left) and fill the form with:

- Any name you wish to see such as: "me @my MAPI db" (this is just a name, caps and spaces are allowed);
- type `localhost` in host if you connect to your own computer, the hostname or IP address otherwise;
- leave 5432 as default port number (except if you changed it during installation);

- leave service empty;
- leave `postgres` as default MaintenanceDB;
- type your newly created role name in "Username";
- type your password.

Click "OK".

You should now have a new server "me @my MAPI db" in your "Servers" tree. Connect by double-clicking on it. Use this connection instead of the previous superuser one (except for admin' tasks), it's safer.

For Linux users: The easiest way is to log in a console as "postgres" user himself and create your own role and superuser rights. In a terminal, (provided you are "sudoer" ie. administrator on your computer), become postgres user by typing (with "Enter" in the end): `sudo su - postgres`

When asked, type your own password. You should now be postgres user, ie. the administrator of this database server.

Just type (and send with "Enter" in the end): `psql postgres` to connect to the system "postgres" (ie. master) database through the SQL console.

Create a superuser role (with your role name in *lowercase*) by typing and sending (don't forget the sigle quotes nor the semicolumn in the end):

```
CREATE ROLE role_name WITH PASSWORD 'my_password' LOGIN SUPERUSER;
```

For a normal user able to create new databases, replace `SUPERUSER` by `CREATEDB`.

Terminate the connection and quit psql by sending `\q`

Please, refer to PostgreSQL documentation for advanced user management.

2.1.5 Create your database

From the "Start" button, launch the PgAdmin3 (or PgAdmin*III*) program. In PgAdmin3 open your favourite server, right-click on "Databases" and choose "Create new database" in the menu. Fill the name of the database in the form and click on "Validate" button. You can later come back to this form through right-click on the database, then "Properties...".

2.1.6 Enable database extensions

The extensions have to be "created" within each database in order to use MAPI. You have to enable PostGIS extension prior to MAPI extension.

Within PgAdmin3 you can add an extension to a database by right-clicking on "Extensions" in the database tree then click on "Add an extension". Add "postgis" extension by choosing its name in the scroll field, then click on "Validate", then do the same for the "mapi" extension.

2.1.7 Updates, in the future?

In case of new updates of the MAPI extension: Once the new zip file has been extracted and installed (see above) the MAPI library have to be updated in each database by sending a SQL command:

```
ALTER EXTENSION mapi UPDATE;
```

You may also revert to a previous version (if any), or choose a specific one, eg.:

```
ALTER EXTENSION mapi UPDATE VERSION '1.0.1c';
```

2.2 Load data and run computations from R

Let's consider the installation was succesful, and that you created a database with the MAPI extension (see section 2.1 on page 11). Note : this part may be installed on a server distinct from your own computer. The following examples are written in R language, so you also need R and the libraries RPostgreSQL (dialogs between R and PostgreSQL) and data.table (fast data parser) installed.

As a test, we use the simulated dataset presented figure 4 on page 9. Datafiles are provided as a zip file, with distances between sample genotypes already computed. Download and uncompress this file, which contains a simulated dataset along with R script files from:
http://www1.montpellier.inra.fr/CBGP/software/MAPI/mapi_examples.zip

You may open the `script_sim2_12.r` (or `script_sim2_12_win.r` for Windows) file in a text *editor* (such as Notepad++[4]), but not a text *processor* (like "word") for copy/paste of lines. A more complete script is available in the appendix (section 10.1 on page 44).

Let's start by loading libraries and opening a connection to PostgreSQL service:

```
library(RPostgreSQL)
library(data.table)
con <- dbConnect(PostgreSQL(), host="localhost", port=5432,
        user="YOUR_NAME", password="YOUR_PASSWORD", dbname="YOUR_DATABASE" )
```

Change login, password of your PostgreSQL connection, the name of the database you will work on, and host & port if needed. Ask your administrator if PostgreSQL was installed on a distinct server. This connection will allow R to connect to the database and to send data and SQL commands.

Important!
PostgreSQL object names such as schemas, tables, views and fields should be in lowercase *in order to avoid a syntax implying double quotes around schema/table/view names and protecting these double quotes in caller program (R, Perl, Python, shell, ...).*

Next step is to transfer the data to the database prior to MAPI computations, beginning with sample locations. Minimal data are sample names, sample locations in euclidean coordinates (x, y) and optionally sample groups (could be sex, year, or a constant if there is no group).

The first R line loads the samples csv file in R, the second one erases previous PostgreSQL table (if any) and all dependent views (the "CASCADE" option drops related objects, so be careful!) and the third line transfers the dataframe from R to the PostgreSQL table.

```
s <- as.data.frame(fread("sim2_12_samples.csv", header=TRUE))
dbSendQuery(con, "DROP TABLE IF EXISTS t_sim2_12_samples CASCADE;");
dbWriteTable(con, "t_sim2_12_samples", s, overwrite=TRUE, row.names=FALSE)
```

Now we are going to load the genetic distances computed between each pair of samples.

The file `sim2_12.arp.gp.spa.out` is a SPAGeDi (Hardy and Vekemans, 2002) output, already manually edited for keeping useful lines[5].

[4] Free software available from: https://notepad-plus-plus.org/

[5] SPAGeDi output file contains a huge header before the matrix. Open the file in a simple text editor (not a text processor like MsWord) and remove all lines strictly before this one:
`Name i Name j N°i N°j Spatial dist Pairwise RELATIONSHIP coefficients (...) ALL LOCI`
Also remove an empty line at the end of the file.
Note: A Genepop-formatted (sample genotypes) file is also provided in the training dataset, if you wish to compute other metrics by yourself.

```
d <- as.data.frame(fread("sim2_12.arp.gp.spa.out", header=TRUE))
## renames columns because problem with ˚ character
colnames(d) <- c("name_i", "name_j", "i", "j", "sd", "pw", "all_loci")
dbSendQuery(con, "DROP TABLE IF EXISTS t_sim2_12_distances CASCADE;");
dbWriteTable(con, "t_sim2_12_distances", d, overwrite=TRUE, row.names=FALSE)
```

Data are now fully loaded, but still have to be transformed in order to meet MAPI field names requirements. This operation is realised by creating views. Views are not tables, just on-the-fly queries allowing data reformatting; in this case renaming fields and conversion of samples coordinates in geographical point for Postgis. We first delete ("drop" in SQL) the view if it already exists, then we build ("create") it.

```
dbSendQuery(con, "DROP VIEW IF EXISTS v_sim2_12_samples;")
dbSendQuery(con, "CREATE VIEW v_sim2_12_samples AS
        SELECT ind::text AS point_name,
                sampling_year::text AS point_group,
                ST_SetSRID(ST_MakePoint(x, y), 3857) AS point_geom
        FROM  t_sim2_12_samples;")
```

MAPI works with projected coordinates. The projection to be used depends on your location and the extents of your dataset. If you wish to use UTM, be aware of the number of your band. Local paper maps may provide the name of your local UTM projection. For example, if your coordinates are UTM zone 32 North, the SRID number will be 32632. In Europe, a common geographical system is Lambert93 which SRID number is 2154.

Map projection
Commonly, a map projection is a systematic transformation of the latitudes and longitudes of locations on the surface of a sphere or an ellipsoid into locations on a plane. Map projections are necessary for creating maps, but distort the surface in some fashion. Depending on the purpose of the map, some distortions are acceptable and others are not; therefore, different map projections exist in order to preserve some properties of the sphere-like body at the expense of other properties.
https://en.wikipedia.org/wiki/Map_projection

The x, y coordinates are transformed into a geographical point and the exact geodetic system identifier (SRID) has to be declared. If your original data are in latitude/longitude (*ie.* angular instead of projected coordinates) one more function[6] have to be called in this query in order to project data points, eg:

```
ST_Transform(ST_SetSRID(ST_MakePoint(lon, lat), 4326), 3857)
```

In this example above, "4326" stands for longitude / latitude WGS84 whereas "3857" is a fake Mercator projection, as in our simulated example coordinates have no "terrestrial" meaning. In this simulated case the reference does not matter but for real data sets it must be the exact geodetic system ID corresponding to coordinates values! See "CRS" (Coordinates Reference System) in QGis for gettig the SRID (named EPSG code) for your coordinates system.

Note that the results will be projected in the same geodetic system as the sample locations (or the user-provided grid for advanced use).

The three MAPI mandatory fields are now ready; let's do the same for genetic distances:

```
dbSendQuery(con, "DROP VIEW IF EXISTS v_sim2_12_distances")
dbSendQuery(con, "CREATE VIEW v_sim2_12_distances AS
        SELECT  name_i AS point_1_name,
                name_j AS point_2_name,
                all_loci AS relation_value
        FROM  t_sim2_12_distances;")
```

This is just a renaming of the fields for MAPI mandatory field names. The worst part is now done (good job guys), let's have fun by running computations!

[6] Refer to PostGIS documentation (http://postgis.net/documentation) for further information, and discover the power of simple functions like these!

As usual, the first line of the following script erases table (if any). The second line starts MAPI data treatment. The function `MAPI_RunAuto` accepts 12 parameters, as follows:

1. points_view_name: name of the table or view containing sampling points along with coordinates. The coordinates must be projected (not latitude/longitude). Mandatory fields names and types are:
   - `point_name text`
   - `point_geom geometry(point)`
   - `point_group text`
2. relations_view_name: name of the table or view containing the distance (or similarity) values between pairs of points. Mandatory fields names and types are:
   - `point_1_name text`
   - `point_2_name text`
   - `relation_value float8`
3. beta (β): the parameter controlling the number of cells in the grid. Value of 0.5 is suitable for regular sampling, and 0.25 for random sampling. The smaller the value, the higher the number of cells.
4. results_table_name: the name of the table that will contain the results (in schema "schema_name"). **Warning:** if the table already exists it will be erased.
5. schema_name: the name of the schema containing the tables. Defaults to "public"
6. eccentricity: the eccentricity value for the ellipses. Default value set to 0.975.
7. error_circle_radius: error circle on sample point locations. Default value is 10 projection units (ie. 10m is a typical GPS error radius).
8. nb_permutations: number of permutations. Default value is 0 (no permutations). 1,000 should be correct for an alpha=5% risk. Smaller alpha values may require more permutations (eg. 5,000 for alpha=1%).
9. do_intergroups: if true, analyse relations between different groups. Boolean, default value is false, ie. no inter-groups analysis.
10. min_distance: minimal distance between samples. Float. Default to NULL, ie. no filtering.
11. max_distance: maximal distance between samples. Float. Default to NULL, ie. no filtering.
12. alpha: the threshold for the cut-off in FDR corrections. Defaults to 0.05 (5%).

OK, let's run it (I broke the string in multiple lines for clarity):

```
dbSendQuery(con, "DROP TABLE IF EXISTS t_sim2_12_results;")
n <- dbGetQuery(con, "SELECT MAPI_RunAuto(
        'v_sim2_12_samples',
        'v_sim2_12_distances',
        0.5,
        't_sim2_12_results',
        NULL,
        0.975,
        10.0,
        1000,
        false,
        NULL,
        NULL,
        0.05
);", verbose=true)
```

The two first parameters and the fourth are obvious. Just use understandable names, in *lowercase*! The third one, beta (β), is explained above and detailed below. The fourth one is useful if you work in a dedicated schema (a kind of "folder" in PostgreSQL), which is very useful when you process mutiple projects since one schema per project avoid messing up the tables. Remaining parameters modulate the analysis.

Error radius and eccentricity parameters both shape the ellipsoidal polygons (see figure 2 on page 6). They are "tuning" parameters depending on coordinates precision (GPS, locality, ...) and expected smoothing intensity. This is an exploratory method, so ... explore!

Number of permutations: 1,000 is a suggested value for a 5% risk.

As we have no groups (see section 4.2 on page 30 for more details about groups) whithin this dataset, no need to allow between-groups computations.

The last two parameters allow to filter minimal and maximal between-sample distances considered in the analysis.

Finally we set the threshold in the FDR correction.

Note: In the script, the object n receives in the end the number of cells in results table, this is why we use "dbGetQuery" R function instead of "dbSendQuery".

The option `verbose=true` allows MAPI to log its outputs in the R console. The true number of cells along with their half-size will be displayed here during computation. If this `verbose` option generates an error (uncompatibility), just remove this parameter.

Now, be patient[7], have a coffee break, talk to your colleagues about this wonderful software you are currently testing, read bibliography, or go back home ;-)

2.3 Mapping your results

2.3.1 Building maps within R

The quality of the outputs in R is not as great as in a GIS, but it's easier to process massive datasets such as simulations.

In the following scripts, replace `NAME`, `PASSWORD` by your own login/password of your PostgreSQL role and `DB_NAME` by the name of the database, and change 'localhost' by server's name if the database is hosted by another computer. Be careful to preserve the single quotes!

Unfortunately, the rgdal package in windows systems does not include PostgreSQL driver. Therefore, we propose here two differents scripts, depending on your Operating System[8]. More complete scripts are available in the appendix (sections 10.2 on page 45 and 10.3 on page 46).

Script for Linux:

```
# This code is provided as an example, only for information.
# Refer to specific documentation for further help.
library(sp)
library(rgdal)
library(RColorBrewer)
library(lattice)
library(latticeExtra)

# reads spatial tables; replace host, dbname, name and password...
```

[7] This simple test lasted 20 minutes (1000 permutations) on my quadricore Intel i7 @2.7 GHz laptop under Linux.

[8] Windows version have been adapted thanks to:
https://geospatial.commons.gc.cuny.edu/2014/01/14/load-postgis-geometries-in-r-without-rgdal/.

```
OGRstring<-"PG:host=localhost port=5432 user=NAME password=PASSWORD dbname=DB_NAME"
resu <- readOGR(dsn=OGRstring, layer="t_sim2_12_results")
samp <- readOGR(dsn=OGRstring, layer="v_sim2_12_samples")
# test existence before reading tails
tails.info <- ogrInfo(dsn=OGRstring, layer="t_sim2_12_results_tails")
if (tails.info$nrows > 0) { # read tails table only if data available
        tails <- readOGR(OGRstring, "t_sim2_12_results_tails")
        # filter "BY" method and separates lower and upper tails
        lower.tail <- tails[tails$tail=="lower" & tails$method=="M_BY", ]
        upper.tail <- tails[tails$tail=="upper" & tails$method=="M_BY", ]
} else { # empty tails
        lower.tail <- new("SpatialPolygonsDataFrame")
        upper.tail <- new("SpatialPolygonsDataFrame")
}

# compute colorscale, ...
sumw.thr <- 20 # sum of weights centile threshold
aec <- rainbow(512, start=0, end=0.8)
vals <- sort(resu$averaged_value[resu$sum_of_weights_centile > sumw.thr])
scale.min <- vals[1]
scale.max <- vals[length(vals)]
scale.by <- (scale.max - scale.min) / 20
at.std <- seq(scale.min, scale.max+scale.by, by=scale.by)
at <- sort(at.std)

# prepare plot of spatial layers
pl <- spplot(resu, "averaged_value",
        col="transparent", col.regions=aec,
        colorkey=TRUE, at=at, main=list(label="sim2_12"),
        sp.layout=c("sp.points", samp, col="black", pch=15, cex=0.7))
if (nrow(upper.tail@data) > 0) { # if upper tails detected
        pl <- pl + layer(sp.polygons(upper.tail, fill=c("transparent"), lwd=5 ))
}
if (nrow(lower.tail@data) > 0) { # if lower tails detected
        pl <- pl + layer(sp.polygons(lower.tail, fill=c("transparent"), lwd=2 ))
}

# write the plot in a file
png("sim2_12_mapified.png", width=800, type="cairo-png")
print(pl) # print !!!
dev.off()
```

Same script for Windows / Linux:

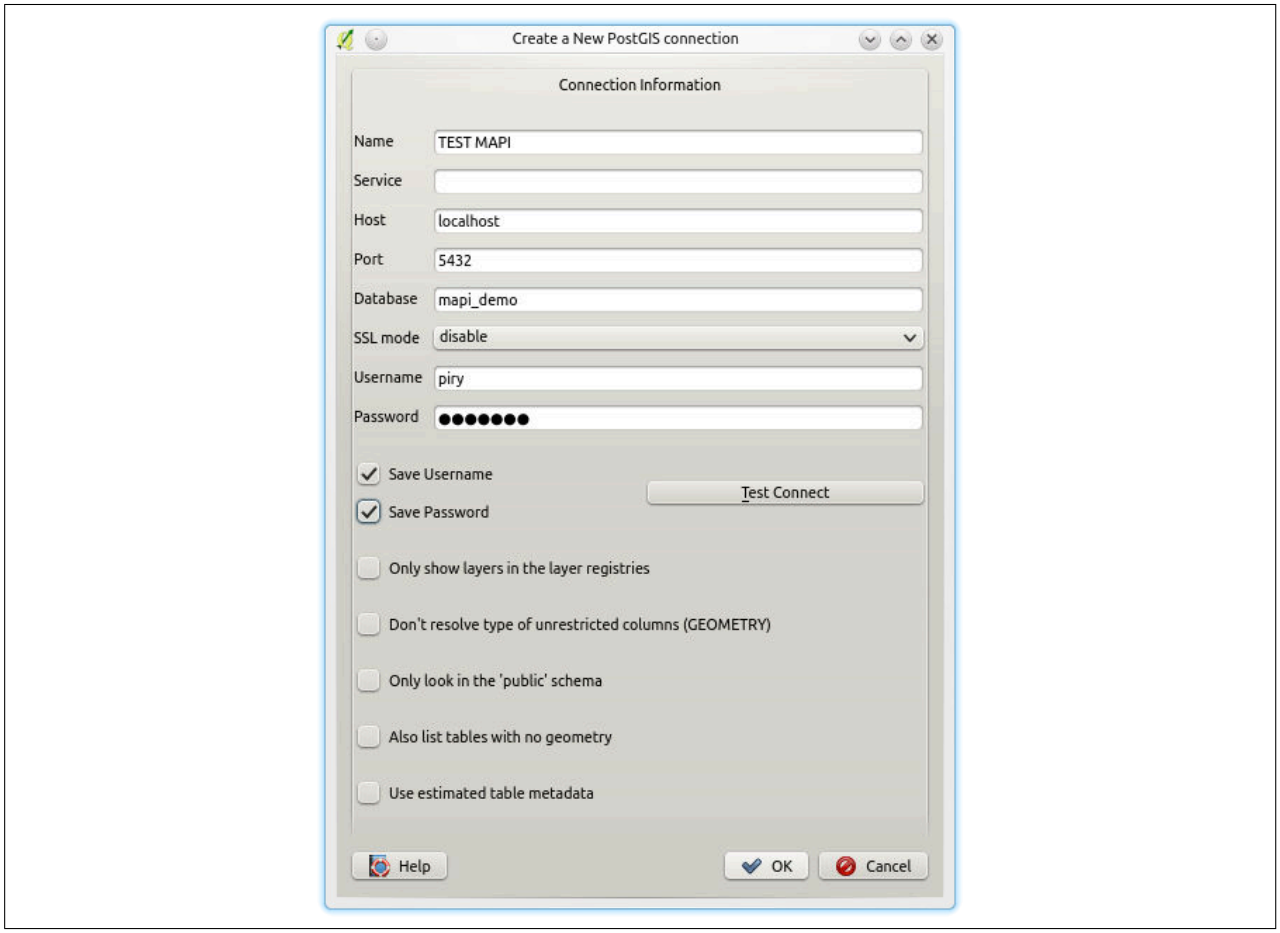
```
## we re-use the "con" connection to the database as used in first MAPI script
library(rgeos)
library(sp)
library(rgdal)
library(RColorBrewer)
library(lattice)
library(latticeExtra)
srid <- as.numeric(unlist(dbGetQuery(con,
    "SELECT DISTINCT ST_SRID(geom) FROM t_sim2_12_results;")))
EPSG <- make_EPSG()
p4s <- EPSG[which(EPSG$code == srid), "prj4"]

resu0 <- dbGetQuery(con, "SELECT id, cell_gid, averaged_value, relation_group,
        ST_AsText(geom) AS tgeom FROM t_sim2_12_results;")
row.names(resu0) <- resu0$id
for (i in seq(nrow(resu0))) {
```

```
  if (i == 1) {
        spResu <- readWKT(resu0$tgeom[i], resu0$id[i], p4s)
  } else {
        spResu <- rbind(spResu, readWKT(resu0$tgeom[i], resu0$id[i], p4s))
  }
}
resu <- SpatialPolygonsDataFrame(spResu, resu0[-2])

tails0 <- dbGetQuery(con, "SELECT id, method, tail, relation_group,
        ST_AsText(geom) AS tgeom
        FROM t_sim2_12_results_tails
        WHERE method LIKE 'M_BY';")
row.names(tails0) <- tails0$id
for (i in seq(nrow(tails0))) {
  if (i == 1) {
        spTails <- readWKT(tails0$tgeom[i], tails0$id[i], p4s)
  } else {
        spTails <- rbind(spTails , readWKT(tails0$tgeom[i], tails0$id[i], p4s))
  }
}
tails <- SpatialPolygonsDataFrame(spTails , tails0[-2])
lower.tail <- tails[tails$tail=="lower", ]
upper.tail <- tails[tails$tail=="upper", ]

samp0 <- dbGetQuery(con, "SELECT point_name, point_group,
        ST_AsText(point_geom) AS tgeom
        FROM v_sim2_12_samples;")
row.names(samp0) <- samp0$point_name
for (i in seq(nrow(samp0))) {
  if (i == 1) {
    spSamp <- readWKT(samp0$tgeom[i], samp0$point_name[i], p4s)
  } else {
    spSamp <- rbind(spSamp, readWKT(samp0$tgeom[i], samp0$point_name[i], p4s))
  }
}
samp <- SpatialPointsDataFrame(spSamp, samp0[-2])


# colorscale
aec <- rainbow(512, start=0, end=0.8)
vals <- sort(resu$averaged_value)
scale.min <- vals[1]
scale.max <- vals[length(vals)]
scale.by <- (scale.max - scale.min) / 20
at.std <- seq(scale.min, scale.max+scale.by, by=scale.by)
at <- sort(at.std)

pl <- spplot(resu, "averaged_value", col="transparent", col.regions=aec,
        colorkey=TRUE, at=at,  main=list(label="sim2_12"),
    sp.layout=c("sp.points", samp, col="black", pch=15, cex=0.7))
if (nrow(upper.tail@data) > 0) {
    pl <- pl + layer(sp.polygons(upper.tail, fill=c("transparent"), lwd=5 ))
}
if (nrow(lower.tail@data) > 0) {
    pl <- pl + layer(sp.polygons(lower.tail, fill=c("transparent"), lwd=2 ))
}

# print plot to png file
png("sim2_12_mapified.png", width=800, type="cairo-png")
print(pl) # print !!!
dev.off()
```

Now, enjoy your own-made final result which should look like figure 4 on page 9 (contours of significant areas may slightly change between sets of permutations).

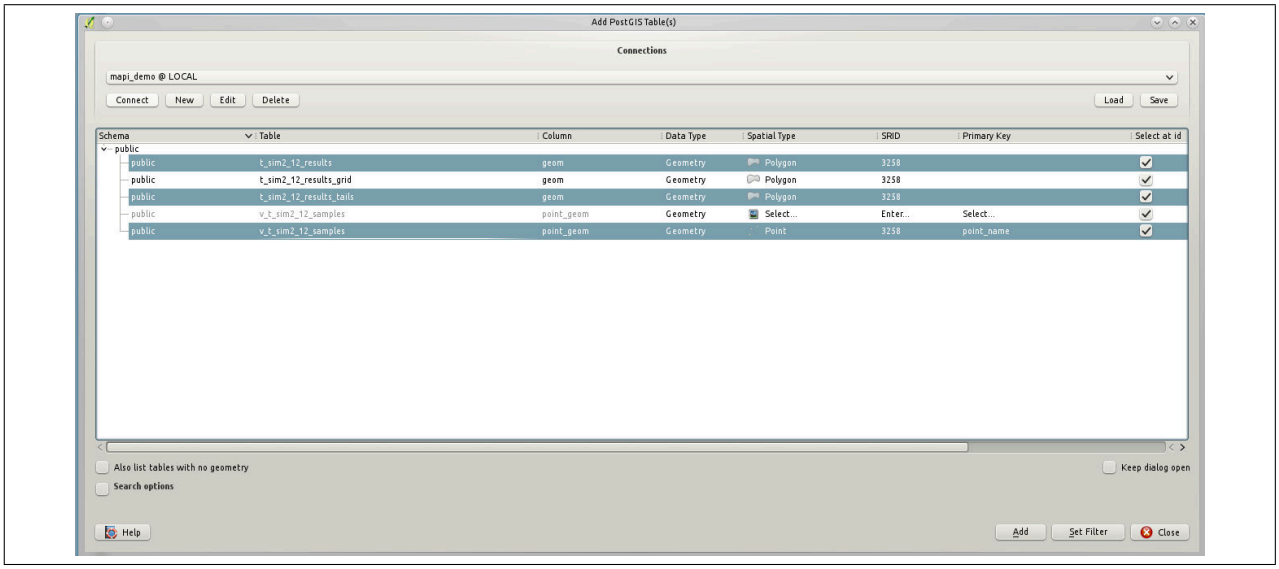
2.3.2 Map your results using QGis

QGis is one of the most popular (and free) GIS software. You may use it for building "manually" nice maps displaying your MAPI results along with landscape layers such as rivers, roads, satellite or aerial images or whatever suits you. Each element open in QGis is named a "layer". Layers can be ordered from bottom to top in to superpose the information. Thematic analysis can be applied on each layer independently. Cherry on the cake, QGis has natively the ability to access PostgreSQL/PostGIS tables as geographical layers. Let's now discover how to connect QGis to the database and map and tune your MAPI layers. From the QGis interface, click on the "blue elephant"  button in order to open the PostgreSQL dialog box.



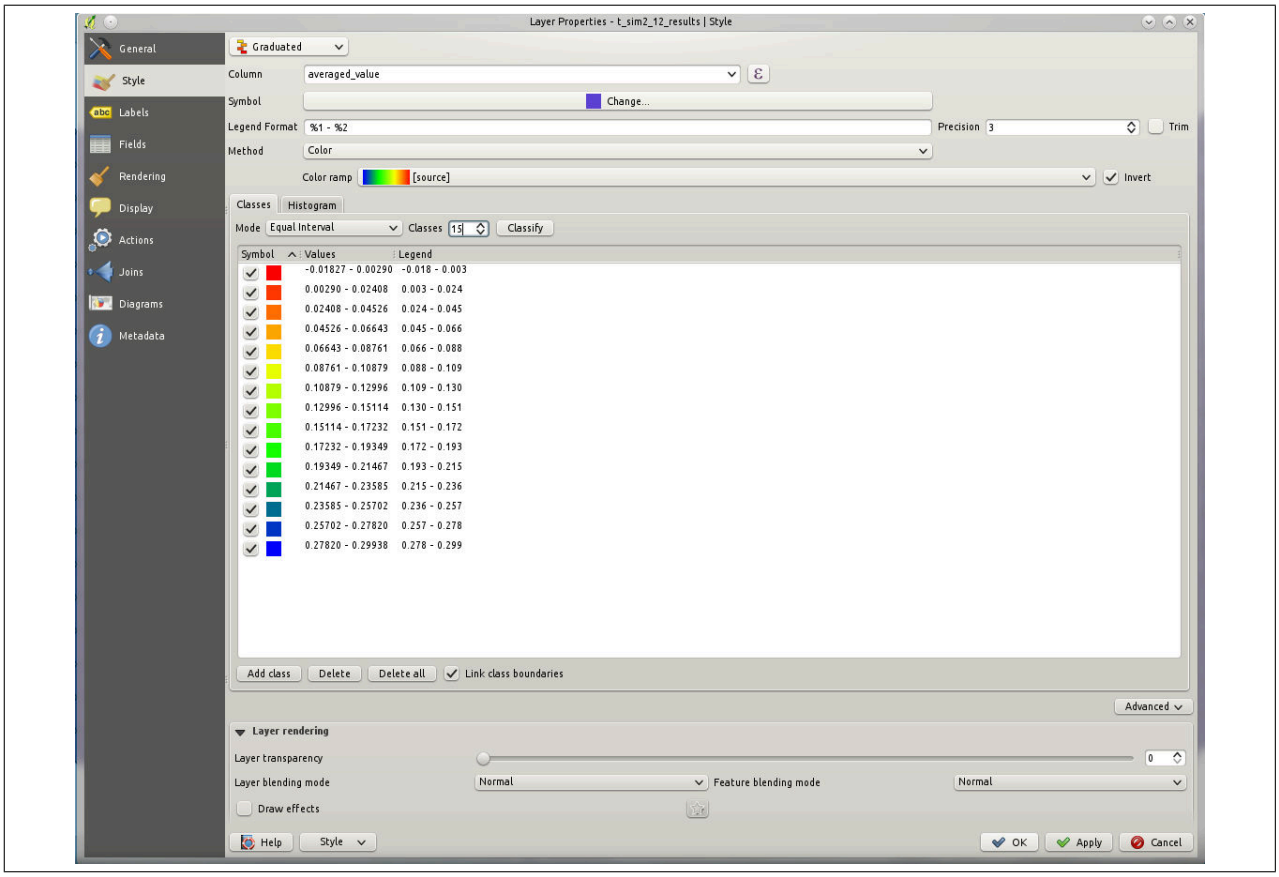
Screen capture 1: QGis /PostgreSQL connexion box.

Enter database name, your login and password for PostgreSQL (and check "save" for both if you are as lazy as I am), give a title to this connexion and finally click on "OK" (screen capture 1). You may be warned that storing login and password is evil...



Screen capture 2: QGis "Add PostGIS Table(s)" dialog.

You may now select the geographical tables containing your precious MAPI results from the "Add PostGIS Table(s)" dialog. Note that for samples, as it is a view, the primary key (unique identifier) is wrongly guessed and must be changed to `point_name`. Click on "Add" button when selection is done.



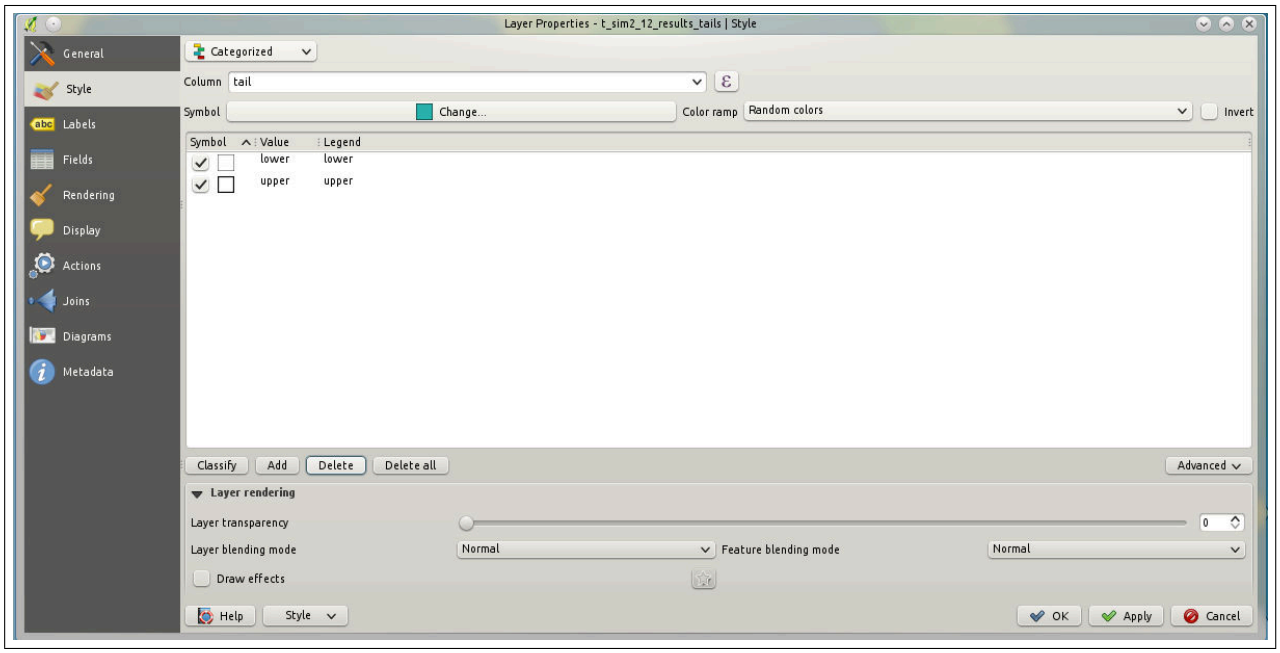
Screen capture 3: QGis styling window with color graduation for results.

Let's now apply a style to the layers. You can order your layers by moving (drag'n'drop) "results" at the bottom, then "tails" layer and the "samples" on the top.

Right-click on the "results" layer and select "Properties". Choose the "Style" tab on the left.

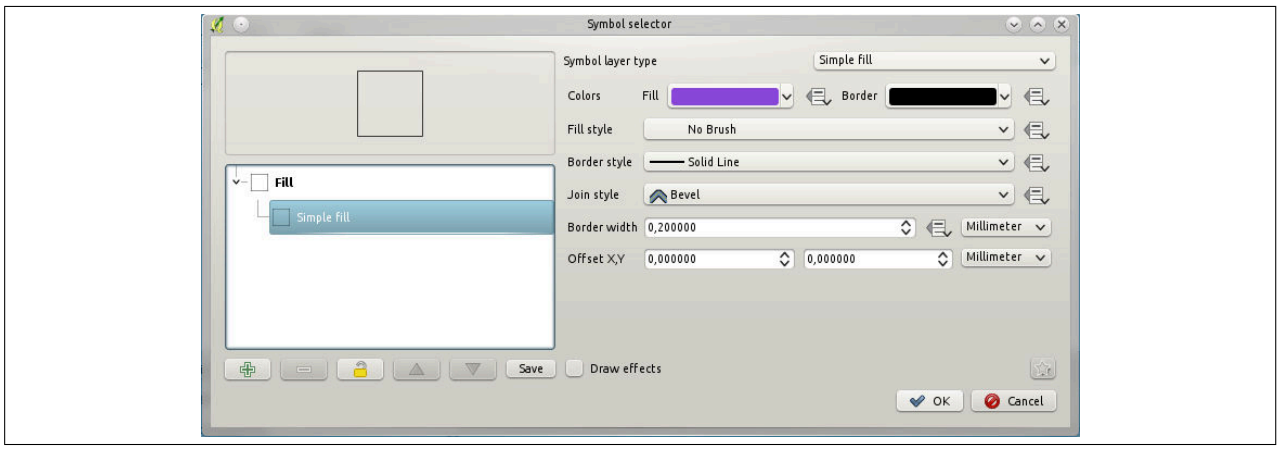
From top to bottom: Select "Graduated", then `averaged_value` in "Column" selector, then click on "Symbol" and in "Simple Fill" set "Border Line" to "No Pen", and "OK". Enter "20" in "Classes" field, select a "Color Ramp" that suits your taste (or build your own otherwise), keep "Equal Interval" in "Mode" selector and finally, click on "classify" in the lower part of the window.

It should more or less[9] look like screen capture 3 on the previous page. Click "OK".



Screen capture 4: QGis styling window with different categories and styles for the upper- and lower tail.

You may now tune the "tails" layer using distinct line width depending of the category (upper or lower tail). Choose "Categorized" then select the column "tail". Click on the "Classify" button.

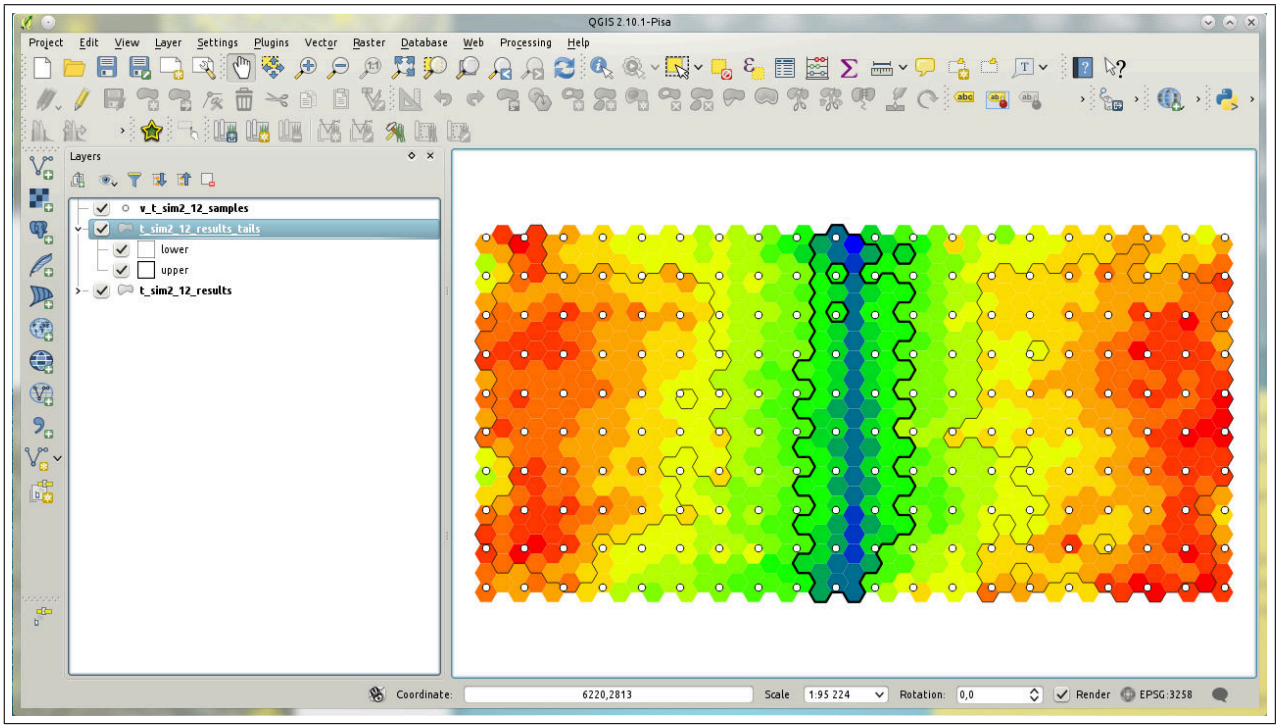


Screen capture 5: QGis styling sub-window with details of style for the lower tail.

Set the line style for each tail (double-click) with "No Brush" and set a width value in "Border width" field.

Finally your screen should look like the screen capture 6 on the facing page.

[9] A bug in my previous QGis version (2.8.1-Wien) leads to a wrong estimation of the range of the values (see http://hub.qgis.org/issues/12306). Version, current versions above 2.9 are now corrected.



Screen capture 6: QGis map from PostgreSQL tables.

Continue to play with QGis to discover more features like building your own colorscales, transparencies, adding backgrounds, etc.

2.4 Save and export results

The MAPI "result" table strore the cells as geometrical shapes (geographical polygons), not rasters (images). Various solutions are available, graphical or command-line style to export your results.

One of the most common file format for exchanging geographical informations is the "ESRI shapefile" format. Note: A field name is limited to 8 characters in shapefiles, so truncations will occur in final shapefile field names. Below, we also give hints to save in other formats.

"Save as" in QGis: Just right-click on the layer, then choose "Save as..." and fill the dialog boxes.

DB Manager in QGis: From the menu Database, open the "DB Manager" assistant. Open the tree until you access the table or view you wish to save. Click on the "Export" arrow and then fill the dialog boxes.

pgsql2shp: Install pgsql2shp and type a command-line such as:
`pgsql2shp -f /tmp/my_mapi_result.shp test_mapi t_sim2_12_results`.
Read man pages for more information.

ogr2ogr: Install ogr2ogr program. It is a bit more complicated to write the command line, but it allows exporting to various formats.

rgdal package in R: As it is based on ogr2ogr it allows the same kind of exports as above but using the R language (linux only!). See package documentation for further information.

PostgreSQL export: From pgAdmin3 you can backup a table (right-click, then "Backup"). Choose "custom" format for binary backups, or "plain" for SQL dump.

There are probably many more solutions than shown in this short list!

2.5 How to build and import my own grid?

2.5.1 Build a grid of hexagons using MAPI

Shapefile
The shapefile format is a popular geospatial vector data format for geographic information system (GIS) software. It is developed and regulated by Esri as a (mostly) open specification for data interoperability among Esri and other GIS software products. The shapefile format can spatially describe vector features: points, lines, and polygons. Each item usually has attributes that describe it, such as name or temperature. The term "shapefile" is quite common, but is misleading since the format consists of a collection of files with a common filename prefix, stored in the same directory. The three mandatory files have filename extensions .shp, .shx, and .dbf. The actual shapefile relates specifically to the .shp file, but alone is incomplete for distribution as the other supporting files are required.
https://en.wikipedia.org/wiki/Shapefile

The function MAPI_GridAuto builds a grid of hexagons, depending on the sampling locations and beta (β) parameter value. This is the function called for building the grid when running MAPI_RunAuto function.

You may wish to set the half-width of the cells parameter by yourself. In this case, just inform the half_width parameter when calling the function MAPI_GridHexagonal, which does the job.

If you prefer to compute the grid with a given a number of cells, the half-width can be estimated using the function MAPI_EstimateHalfwidth.

See functions and parameters in section 9.2 on page 40 for more details and an example.

2.5.2 Build a grid of squared cells using QGis

A dedicated tool is available through the QGis menu: "Vector" → "Resarch Tools" → "Vector grid".

Check "Output grids as polygons", choose the extent you wish either from a layer or from current canvas, along with the size of each cell. Make sure to not create a huge grid with too many cells. Set the shapefile name in your directories ("Browse" button). Check "Add result to canvas" and finish with the "OK" button.

The grid is now displayed on your map. The corresponding shapefile may now be easily loaded in your PostgreSQL database using "Spit" or "DB Manager" plugins in QGis.

2.5.3 Other software

DGGRID (http://discreteglobalgrids.org/software/) is dedicated to the construction of spatial grids. MAPI is unfortunately not able yet (work in progress!) to process an worldwide grid. You may nethertheless use DGGRID to build regional-scale grids in cartesian coordinates.

2.5.4 How to import a grid shapefile in batch?

For batch processing in a shell the "shp2pgsql" command-line tool may be more convenient than point-n-click in QGis. A single line allows to transform the shapefile into SQL code for PostGIS and to import it into the database through a "pipe" eg.:

```
shp2pgsql -s my_srid -d my_grid.shp my_schema.my_grid_table | psql -d my_mapi_db
```

Don't forget to use projected coordinates, or a reprojected the grid (see shp2pgsql manual).

2.5.5 Reusing an existing grid in MAPI

In order to reuse an existing grid you have to call the MAPI_RunOnGrid function, in which the name of the table that stores the grid will play the role of the beta (β) parameter in the MAPI_RunAuto function.

In the same way, you can re-use a grid that has already been automatically generated by calling the MAPI_RunAuto function.

In this advanced mode the detection of significant areas is not automatically run and you have to explicitly add a line. A R-wrapped SQL code should therefore look like this:

```
# computes MAPI using user-provided grid: my_grid_table
n <- dbGetQuery(con, "SELECT MAPI_RunAuto(
        'v_sim2_12_samples',
        'v_sim2_12_distances',
        'my_grid_table',
        't_sim2_12_results',
        NULL,
        0.975,
        10.0,
        1000,
        false,
        NULL,
        NULL,
        0.05
);", verbose=true)
# now computes significant areas detection
n <- dbGetQuery(con, "MAPI_Significance (
        't_sim2_12_results',
        NULL,
        'M_BY',
        0.05,
        false
);", verbose=true)
```

A supplementary option is available to speed up computations (see section 9.1 on page 39). This option simplifies the computation by moving the sampling locations to the centroid of the cell they belong to, and thus simply reduces the number of locations pairs. This option should be used for draft only.

3 How does MAPI handle real-life data?

Let's have a look at some biological datasets processed with MAPI, and how changing parameters impact outputs.

3.1 Watermelon Mosaic Virus dataset (DNA sequences)

The map in figure 5 on the next page illustrates a dataset of DNA sequences from emerging (EM) strains of the watermelon mosaic virus (WMV). Refer to Desbiez et al. (2009); Joannon et al. (2010) for more informations about data and previous analyses.

The pairwise metric is the genetic distance of Tamura and Nei (1993) computed between pairs of sequences. The eccentricity parameter was set to 0.975 and error-circle radius to 1 km.

The significant discontinous area bissecting the area from North to South coincides with the barrier previously identified with the Monmonier algorithm (Joannon et al., 2010).

Figure 9 on page 33 illustrates on the same dataset how changing the eccentricity parameter result in more or less smoothing of the observed patterns.

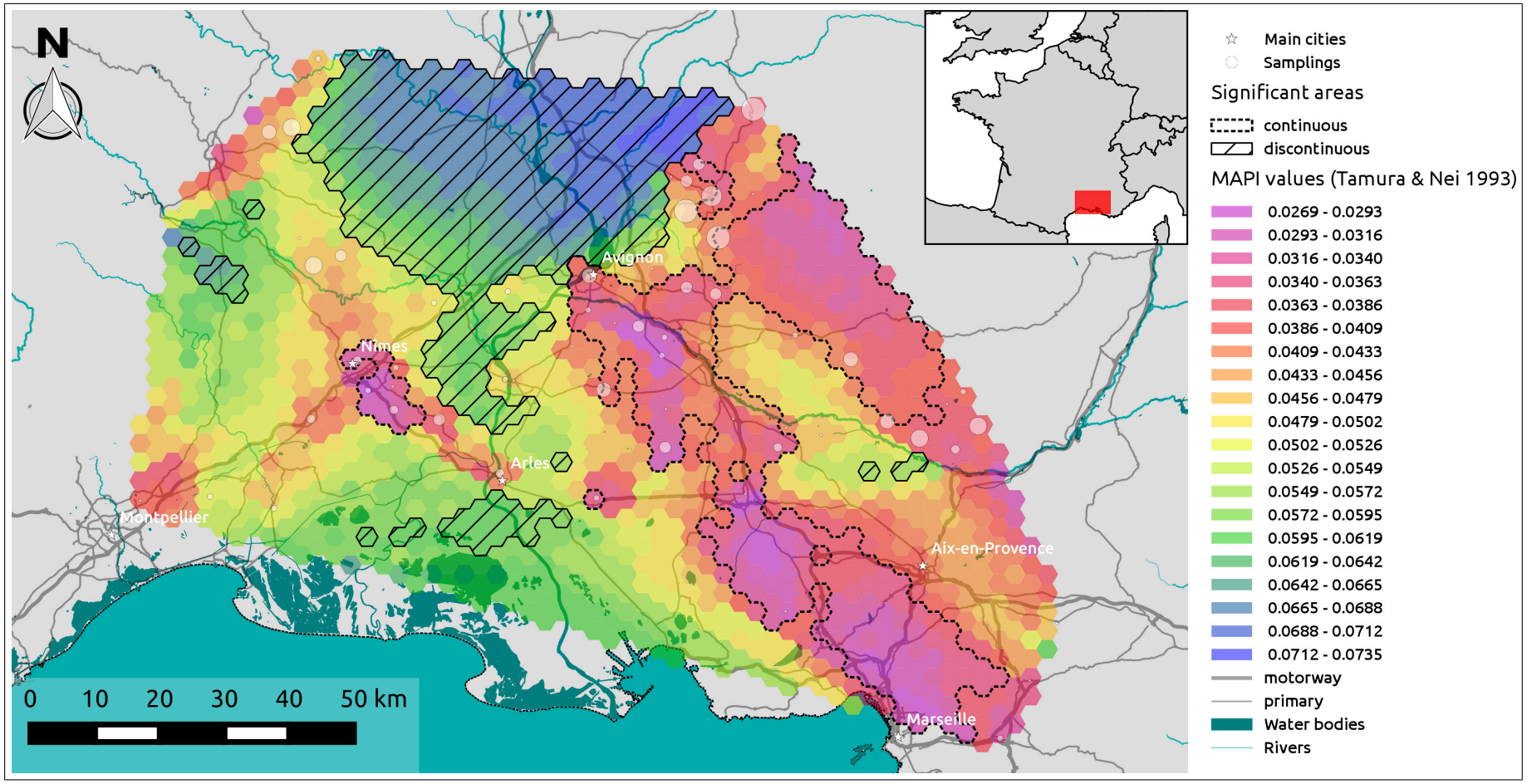


Figure 5: Illustration of the MAPI analysis of plant virus sequences (Watermelon Mosaic Virus, emerging strain) in southern France in 2006-2008.

3.2 Common vole dataset (microsatellite markers) and distance filtering effects

Genetic data are individual microsatellite genotypes and the metric used in the MAPI analysis is the genetic distance a_r (Rousset, 2000). See Gauffre et al. (2008, 2014) for more information on data and previous analyses. The role of the highway as a barrier was a central question in this study.

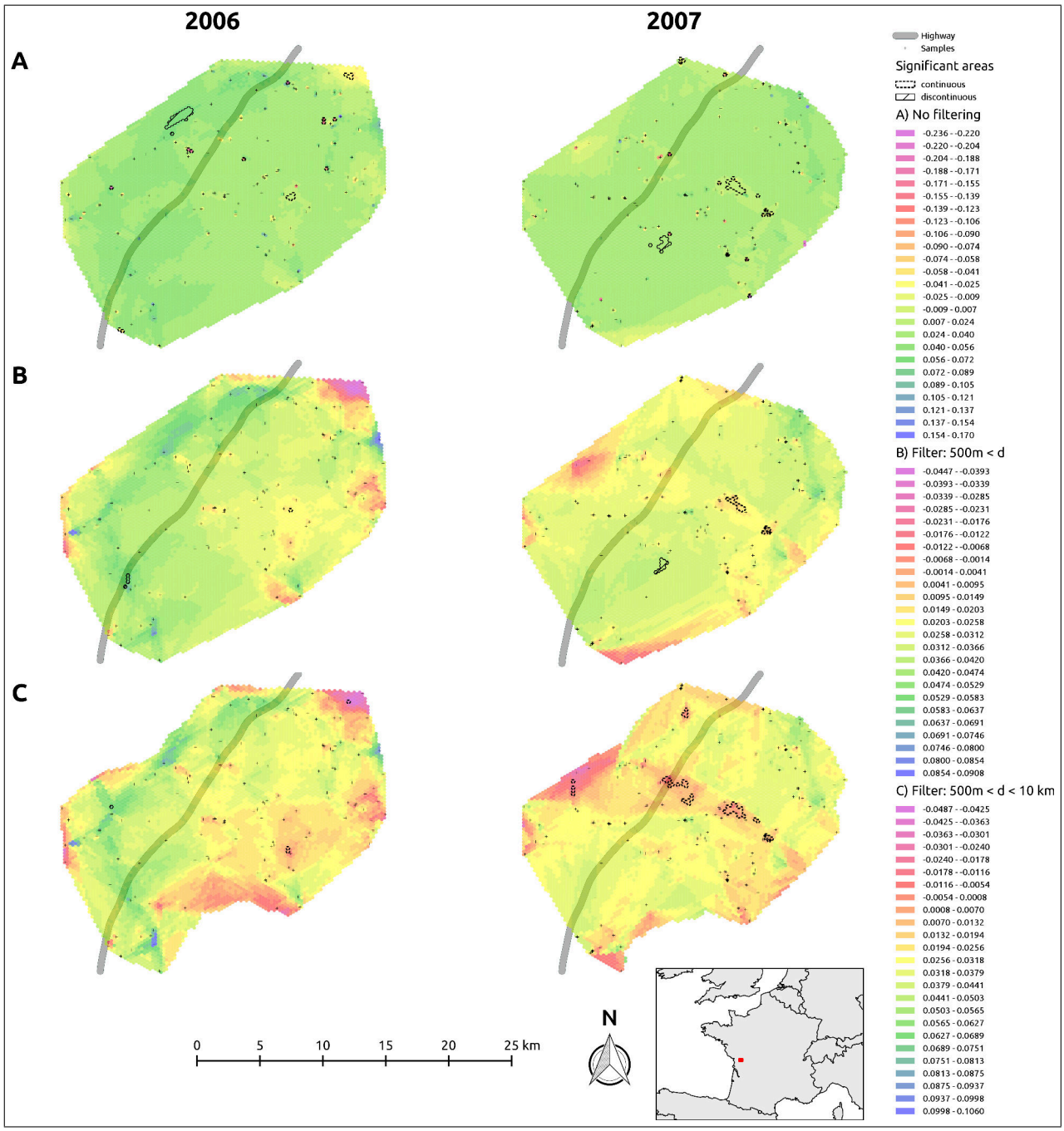


Figure 6: MAPI analysis of common vole *Microtus arvalis* microsatellite data sampled in 2006 and 2007 in western France. The highway is the line running from south-west to north-east. Maps A, B & C were done using different values to filter between-samples geographical distances.

Figure 6 shows the effects of the distance filtering on the MAPI results. MAPI parameters values were: $eccentricity = 0.975$, sampling $error\_circle\_radius = 10m$ (GPS points), $\beta = 0.25$, $nb\_permutations = 1000$. When mapping the results in QGis, cells with the 5% lowest sum-of-weights were discarded in order to avoid local side-effects ($sum\_of\_weights\_centile > 5$ filter).

Maps **A**: no filtering on distance. Extreme m_w cell values occur very locally (isolated cells) due to the sampling of highly related individuals within colonies (Gauffre et al., 2008, 2014). This effect, althrough it may reveal intra-deme structure, should be removed to investigate global patterns.

Maps **B**: low-pass filter (parameter $min\_distance = 500$). Local effects due to both social system and sampling scheme are not visible anymore and a more global pattern is emerging.

Maps **C**: pass-band filter (parameters $min\_distance = 500$, $max_distance = 10000$). This setting enhances intermediate-scale information.

In all cases, clearly the highway does not appear as a likely barrier!

3.3 Forest ground beetles dataset (microsatellite markers) and landscape analysis

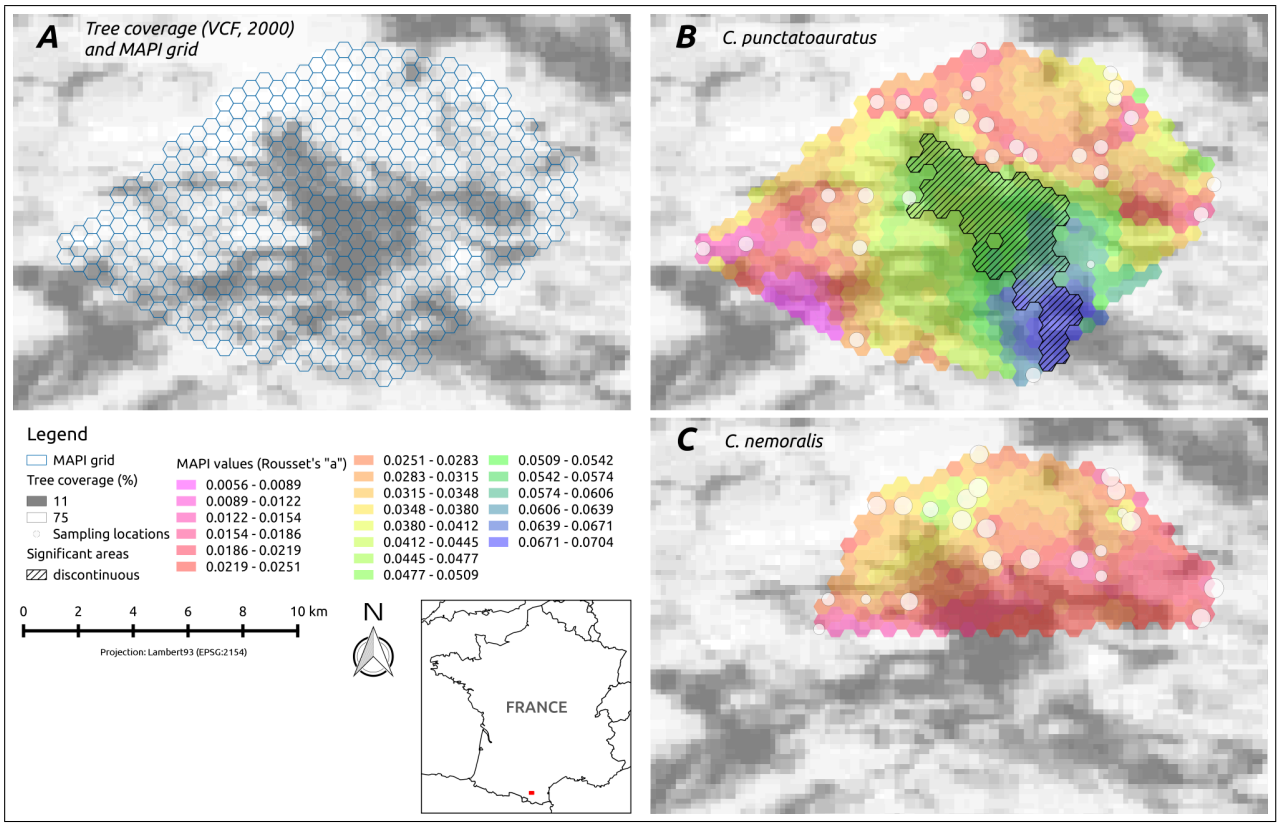


Figure 7: MAPI results on two ground beetle species. The upper left map (A) illustrates the tree coverage (source: Vegetation Continuous Fields, year 2000, DiMiceli et al. (2011)) along with the cells of the MAPI grid. Right maps show the tree coverage layer under MAPI results for *Carabus punctatoauratus* (B) and *Carabus nemoralis* (C). A significant discontinuous area (lined area from the upper tailed test) is observable only for *C. punctatoauratus*.

Data are individual microsatellite genotypes from two forest ground-beetles with contrasted level of habitat specialization: a forest specialist (*Carabus punctatoauratus*) and a generalist (*C. nemoralis*). The MAPI analysis (figure 7) were performed using the genetic distance a_r (Rousset, 2000), and an eccentricity value of 0.975. See Brouat et al. (2003) for more information on data and previous analyses. This dataset is available in the "examples.zip" file on the MAPI website: http://www1.montpellier.inra.fr/CBGP/software/MAPI/.

MAPI analyses supported the conclusions of Brouat et al. (2003) by identifying a significant area of genetic discontinuity corresponding to a large open field for the specialist, whereas no significant pattern was found for the generalist.

See also figure 10 on page 36 for another funny output of the same dataset.

Further analyses of landscape information at the scale of the MAPI cells can be performed in R. To do so, MAPI results and the landscape raster are loaded in R using `readGDAL`. Values from the landscape raster within each MAPI-cell can be summarized (eg. mean) using the R `extract` command. Note that the raster containing tree coverage from VCF (Vegetation Continuous Fields) was first clipped and reprojected in the same projection (Lambert93, EPSG=2154) as the MAPI results and saved as `trees2000_2154.tif`. Here is a piece of R code[10] for such treatment:

```
library(rgdal)
library(raster)
# read the raster
trees <- raster(readGDAL("trees2000_2154.tif"))
OGRstring <- paste("PG:dbname=",db," host=",host," user=",user," password=",pwd," ", sep="")
# get MAPI results table
results <- readOGR(OGRstring, "my.schema.my_results_table")
# extract mean of raster values covered by each cell
results.trees <- extract(x=trees, y=results, method="simple", small=TRUE, fun=mean, sp=TRUE)
# rename the extracted column
names(results.trees@data)[names(results.trees@data) == 'band1'] <- "trees"
# get center coordinates of cells
results.centroids <- getSpPPolygonsLabptSlots(results.trees)
colnames(results.centroids) <- c("x", "y")
# keep only useful columns
results.data <- cbind(as.data.frame(results.trees@data[ ,c("cell_gid", "relation_group",
        "averaged_value", "weighted_stddev", "sum_of_weights", "sum_of_weights_centile",
        "number_of_ellipses", "pvalue", "trees")]), results.centroids)
# write all this stuff in a file
write.table(results.data, file="my_mapi_landscape.csv", row.names=FALSE, sep="\t")
```

Now, the dataframe `results.data` contains for all cells the MAPI value m_w, the centroid coordinates, and the tree coverage. These data can be used for further statistical treatments, but remember that cells are spatially autocorrelated.

4 Exploring the parameter space

Even if MAPI is (almost) free of assumptions, some parameters allow to filter inter-sample distances, to change grid resolution, or to modify ellipse shape[11].

4.1 Cell size and grids

Based on the Nyquist-Shannon theory, the side-length p of a pixel can be computed using the area of sampling sampling zone (A) and the number of sampled points (N) as follow:

$$p = \beta \cdot \sqrt{\frac{A}{N}}$$

where β is the value depending on sampling regularity: $\beta = 0.5$ for regular sampling, $\beta = 0.25$ for random sampling (Hengl, 2006).

The area of a squared pixel is: $a = p^2$ The halfwidth hw for an hexagonal cell of this same area a can be computed as $hw = \sqrt{a/2.598}$.

[10] R is not my cup of tea, so if you know a better/shorter/faster way to do the job, please share your knowledge with me in order to improve this manual!

[11] Be careful that "In the parameters space, everyone can hear you scream".

Nyquist–Shannon sampling theorem

In the field of digital signal processing, the sampling theorem is a fundamental bridge between continuous-time signals (often called "analog signals") and discrete-time signals (often called "digital signals"). It establishes a sufficient condition for a sample rate that permits a discrete sequence of samples to capture all the information from a continuous-time signal of finite bandwidth. The application of the sampling theorem to images should be made with care. For example, the sampling process in any standard image sensor (CCD or CMOS camera) is relatively far from the ideal sampling which would measure the image intensity at a single point. Instead these devices have a relatively large sensor area at each sample point in order to obtain sufficient amount of light. Despite images having these problems in relation to the sampling theorem, the theorem can be used to describe the basics of down and up sampling of images.
https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem

Regular hexagons

In geometry, a hexagon is a polygon with six edges and six vertices. The common length of the sides equals the radius of the circumscribed circle. The longest diagonals of a regular hexagon, connecting diametrically opposite vertices, are twice the length of one side. From this it can be seen that a triangle with a vertex at the center of the regular hexagon and sharing one side with the hexagon is equilateral, and that the regular hexagon can be partitioned into six equilateral triangles. The area A of a regular hexagon of half-width or side length t is given by: $A = \frac{3\sqrt{3}}{2}t^2 \approx 2.598t^2$.
https://en.wikipedia.org/wiki/Hexagon

Then, cell's halfwidth can actually be estimated as:

$$hw = \sqrt{\frac{\beta^2}{2.598} \cdot \frac{A}{N}}$$

In the "automatic" MAPI function, you just set the beta parameter and MAPI computes automatically a grid of heaxagonal cells. Despite optimizations for improving performances, very high-resolution grids ($> 100,000$ cells) may probably overflow ... your patience.

If the default grid doesn't meet your requirements, you can build your own using MAPI the functions MAPI_GridAuto and MAPI_GridHexagonal. Refer to the technical part 9.2 on page 40 of this manual for a detailed description.

You also can build your own grid, or use an existing land-cover (provided that cell sizes are quite similar) as input for the MAPI_RunOnGrid function.

4.2 Groups

If the field `point_group` includes more than one value, each group is processed separately but share a same common grid. Results will be stored group by group in the same table, and then will have to be separated using the `relation_group` field when mapping. By default, no inter-groups computations are done (parameter #9 set to false). Note that all results are in the same geographic table, so in order to map distinct group results you MUST filter the group you wish to map. Otherwise the map would be unpredictable depending on the z-order of the cells.

A classical example is to use the sex as group. You can produce different maps for each sex, but use the full MAPI output to apply a common color scale on all maps. The same result table has therefore to be loaded three times, one (hidden) for all groups in order to define the color scale, another time filterd for male relationships, and another time for female relationships. The color scale can be copied from the full result and pasted on the filtered results (in QGis, right-click, style, copy style from hidden layer with complete table, then on each "group" layer right-click, style, paste style).

Different sampling seasons may be processed in the same way. You can therefore analyze different maps sharing the same colorscale.

If the parameter #9 is set to true, and if inter-group metrics were computed and loaded, all possible relationships will be processed.

4.3 Filtering on distances

Depending of the matter at end, it may be wise (or not) to work with a subset of between-samples geographical distances. The range of distances used in the analysis can be restricted using two parameters of the main MAPI functions.

Figure 8 on the next page illustrates how distance filtering determine the number of connections in the network.

The parameter #10 controls the minimal distance between samples and acts as a low-pass filter. The parameter #11 controls the maximal distance between samples and acts as a high-pass filter. No filtering

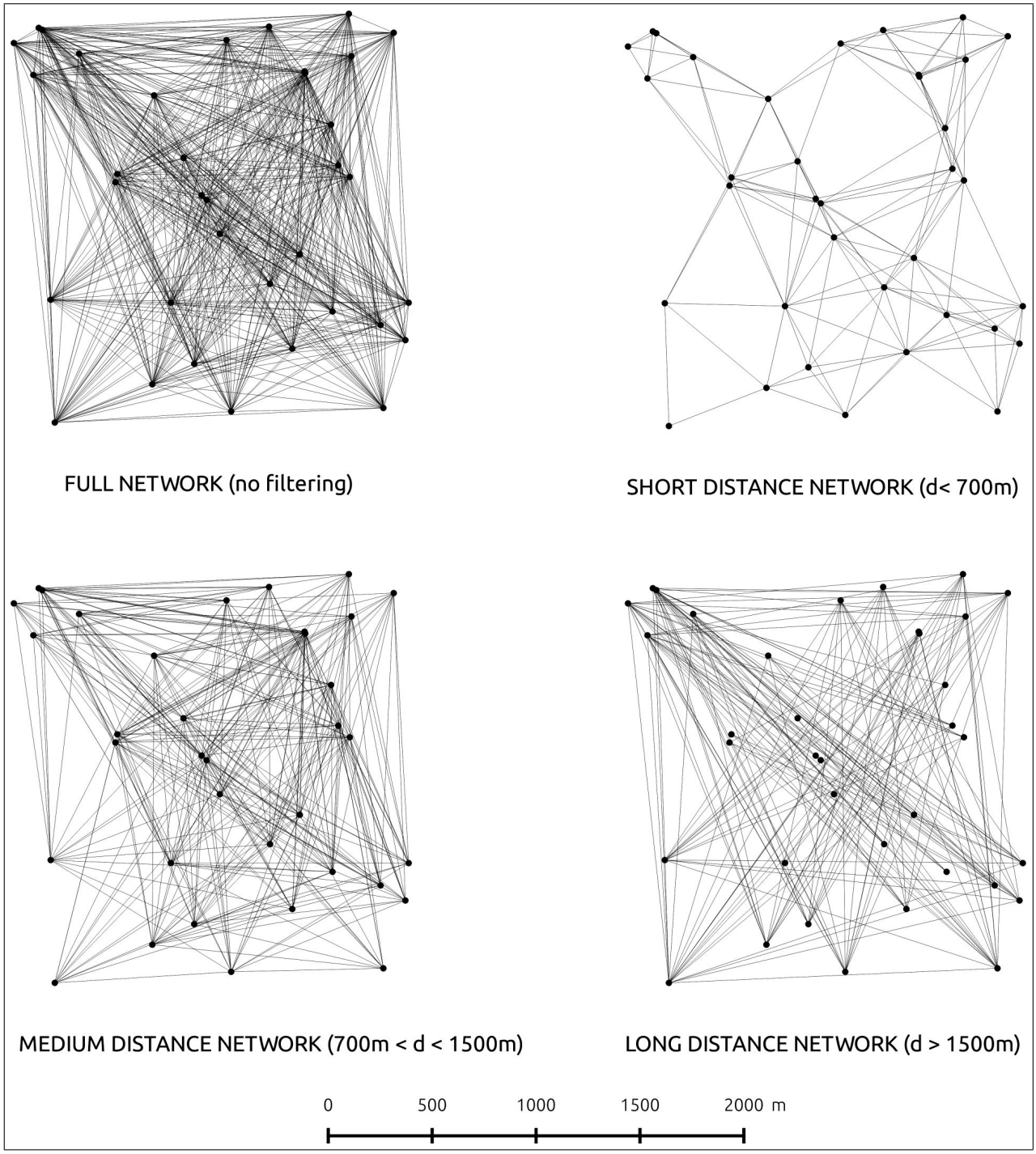


Figure 8: Illustration of distance filtering on network connectivity (random spatial points interconnected).

is applied when these parameters are set to NULL. Both values combined allow to build a band-pass filter which discards short and long distances.

Distances are expressed in units depending on the geodetic system of the sample points coordinates. If a grid is imported its geodetic system prevails.

See figure 6 on page 27, for an example of filtering on geographic distance.

4.4 Ellipse shape

The construction of the ellipsoidal polygons according to the eccentricity and error circle radius parameters is described in the section 1.2 on page 4 (see also figure 2 on page 6).

Setting of the error circle radius (*err_radius* parameter) depends on how accurate are sample coordinates and should be made according to sampling protocols (GPS coordinates, approximated positions...).

For the *eccentricity* parameter, as a start to explore a new dataset, we suggest to use the default value (0.975), which is a good comprimise in term of smoothing intensity.

The effect of this parameter is illustrated figure 9 on the next page on a plant virus data set (DNA sequence). Six independent analyses were performed using the genetic distance of Tamura and Nei (1993) as the metric of interest, a minimal distance between samples of 500m and values of eccentricity ranging from 0.800 to 0.999. A lower- and an upper-tailed test, based on 1000 permutations, were perfomed to detect significant continuous and discontinuous areas. The color scale on figure 9 is such that blue stands for high genetic distances, red to purple for small genetic distances. The sampling sites are represented with white circles with a size proportional to the number of samples.

The difference between the six analyses mainly result in the level of pattern fragmentation. High eccentricity values (i.e. 0.990, 0.999) lead to over-detailled patterns while low values (e.g. 0.800) have a quite strong flatening effect. A important point, however, is that the main triangle-shaped area of high genetic discontinuity that bisect the study area from North to South is uncovered from the six analyses, whatever the eccentricity value.

Performing several independent analyses using different eccentricity values can allow to assess the robustness and then the relevance of the observed patterns.

This intends to illustrates effects of eccentricity on MAPI results: From the different maps, we can see that the triangle-shaped discontinuous areas in the northern part of the study area and the central hole between continuous areas in the eastern part are very consistent. This show that well-marked spatial patterns, and then relevant information, can be uncovered whatever the values used for ellipse eccentricity, number of grid cells and moving-window radius. In other words the method is quite robust about main features.

5 Maping and interpeting MAPI outputs

Keep in mind that MAPI is essentially a smoothing procedure using the overlap between ellipses as a way to share information between spatial connections. As the bandwidth for kernel density estimators, eccentricity is a tuning parameter that controls the level of smoothing and then impacts the resulting surface. This surface is a grid in which the cells provide information on the average intensity of the network's relationships crossing at the cell locations. The pattern of variation observed over the surface allows to localise highly continuous and discontinuous areas where the majority of the crossing connections correspond to very low or very high pairwise values (when using dissimilarity measures). The significance of these areas can be assessed using a nonparametric randomisation procedure.

Also, analysis of MAPI cell values along with landscape features must be considered as an exploratory approach that can help to identify candidate variables but not as a test of hypothesis of causal relationships. The landscape genetic analysis presented in Piry et al. (2016) is some kind of proof-of-concep, not a "magic solution".

Hereafter, we try to give some hints to map and interpret MAPI results.

5.1 Color ramp, transparency and background

In this manual, we illustrated similarity (continuous areas) with "warm" colors (eg. yellow to red) while dissimilarity was illustrated with "cold" colors (eg. blue to dark pink). Of course this choice is yours;

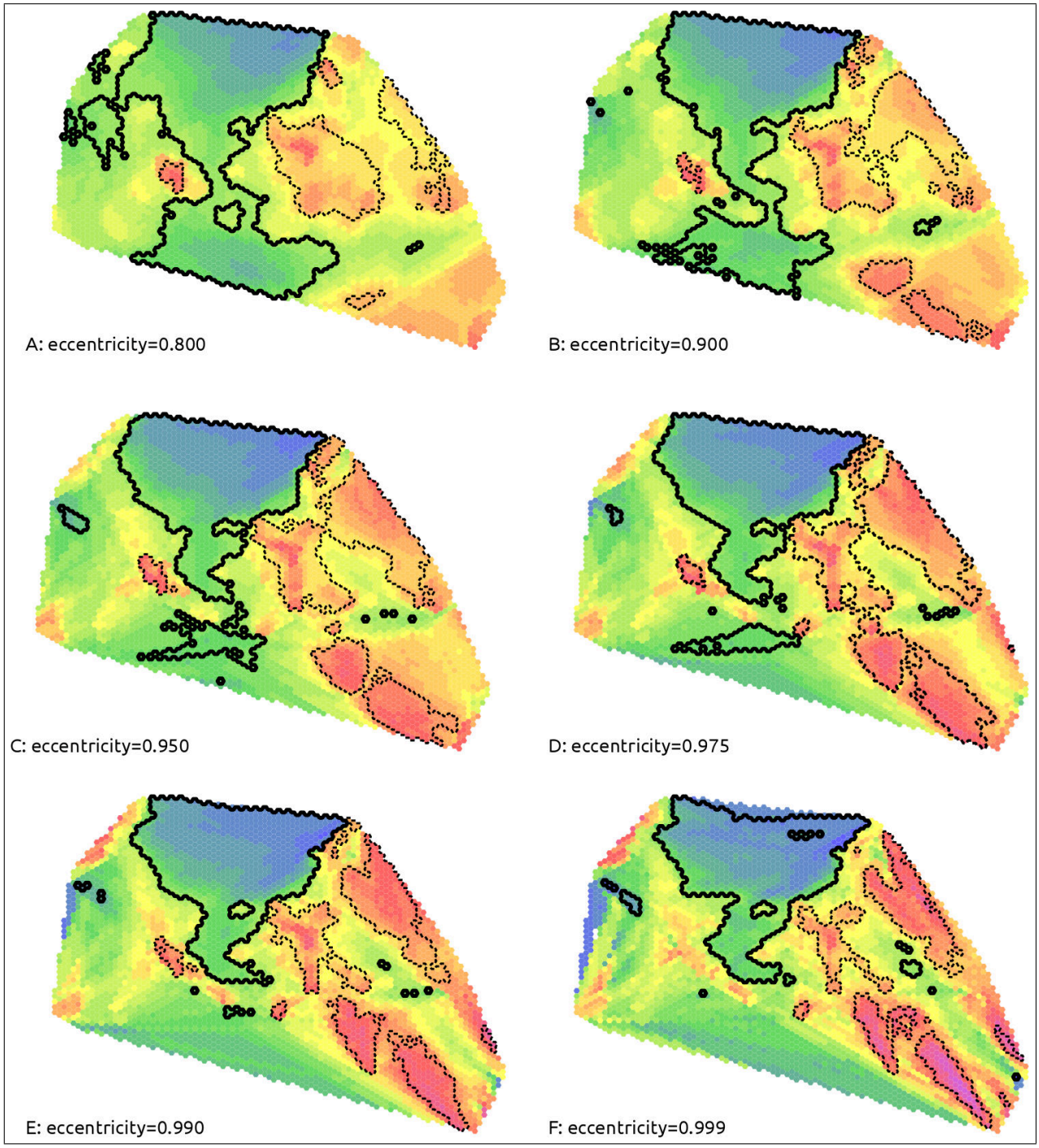


Figure 9: Illustration of the influence of eccentricity on resulting maps (true dataset of plant virus sequences). Shared MAPI parameters for all maps are: 1,000 permutations, minimal distance between samples = 500m, no maximal distance. Colorscale is such as blue stands for high genetic distances, red to purple for small genetic distances; same colorscale applies to all maps.

nevertheless, we recommand this kind of colour ramp, rainbow-styled or not, as it can help readers in interpreting MAPI results. You can easily build your own color ramps in R or in QGis (see screen capture 3 on page 21) to improve map lisibility. In QGIS, the colorscale should be applied with constant intervals on the range of MAPI values rather than the "jenks" or quantiles options. The two last splitting methods lead to uneven slices and enhance variation in map colours, which maybe nice but also, sometimes, misleading as small variations of the metric are translated in distinct colours in abundant classes. For example, on the figure 6 on page 27, the maps are all done with constant intervals. As you can see, the map A is very "flat-coloured" while it exhibits a range of cell values much wider than the maps and B and C. Actually, on map A, the full-range of variation in cell values is actually driven by local information reflecting both vole

social system and sampling protocol. Automatic breaks ("jenks") would be more robust to such extreme cell values, however it may lead to misinterpretations due to uneven colour scaling.

Never forget to add the color legend along with the map[12] to allow readers to interpret patterns and (dis)continuituous areas with their own eyes. When mapping MAPI results, adding some transparency can help to see underlying landscape features from background maps (if any) such as OpenStreetMap or teledetection products (Landsat or finer resolution images, NDVI or LAI indexes, landcovers), Digital Elevation Model or derived slopes, etc.

5.2 Significant areas

Detection of significant areas depends not only on parameter setting (eccentricity, number of permutations, number of cells in the grid) but also on your dataset (number of loci, number of samples, spatial coverage of the sampling...). Poor datasets are likely to give poor results.

5.2.1 Continuous areas

When distance-like metrics are used continuous areas will be identified by using the lower-tailed test. They "delineate" spatial areas where similarity between samples is significantly higher than expected randomly given an risk (e.g. high level of gene-flow or a highly-shared ancestry!). You may illustrate, with distinct linetypes, 5% and 2% significant levels in order to assess more precisely which cells carry well-supported significant information.

5.2.2 Discontinuous areas

Discontinuities may arise from various processes. A physical barrier, as simulated in Piry et al. (2016), will clearly produce a significant discontinuous area with a shape that is dependant on parameter settings (i.e. smoothing intensity). Other process such as local genetic drift can also result in significant discontinuous area. Spatially nested MAPI analyses may help to disentangle between confounding effects.

6 Hints & tricks...

6.1 Convert a matrix to a vector

MAPI uses a vector of genetic distances/similarities as input, and not a matrix. As shown in section 2 on page 11, the SPAGeDi (Hardy and Vekemans, 2002) software provides this kind of output format. Other softwares used for computing genetic distances don't, and you may need another tool to do so.

In R, the `melt` function in the package `reshape2` allows this kind of transformation. Refer to the documentation of the package for such examples or do a bit of "google-ing". Among others, the webpage http://seananderson.ca/2013/10/19/reshape.html gives some good examples for the `melt` function.

By the way, note that the half-matrix of relationships is enough as MAPI recreates symmetrical relationships and filter out the diagonal (ie. the relation between a sample and itself) when building inner temporary tables.

[12] Yes, the legend is lacking figure 9 on the preceding page, as only the respective shapes of significant areas were meant to be commented.

6.2 Which metric should I use?

In populations genetics, the software SPAGeDi (Hardy and Vekemans, 2002) offers the possibility to compute various individual- and population-based pairwise metrics. Part 6.1 of SPAGeDi manual[13] shows a synthetic table with pro and cons of each metric implemented, along with guidelines for interpretation.

6.3 How to run MAPI faster?

Memory size available to PostgreSQL is a huge issue for increasing speed. Allowing postgres to access more memory may be done by:

- optimizing computer usage by stopping other running softwares, or nightly backups, ...
- tweaking PostgreSQL configuration. Depending on your PostgreSQL version and your Operating System, check for hints about memory management and configuration. Memory tweaking of PostgreSQL 9.1 under linux greatly improved computations. PostgreSQL 9.3 should not be sensitive to these tweaks.

Otherwise, you can reduce MAPI memory-print by changing different parameters:

- reduce the resolution of the grid (*ie.* increase beta (β) parameter, or increase halfwidth of the cells). If you are using "fat" ellipses (eccentricity ≤ 0.9), which cover more space but are less resolutive, you don't need a very fine grid.
- reduce the number of permutations (linear effect). A try with 0 (zero) permutation may already gives interesting information but does not allow to identify significant areas.
- if your sampling is more or less aggregated around sampling points, replace individual coordinates by population ones. Up tou you to decide wether this is pertinent or not.
- when calling the "MAPI_RunOnGrid", use the "pointsToCentroid" option that moves individual locations to the center of the cell they belong to.
- for a very large study area crowded with sampling points, long-distance relationships may be more noisy than useful. Filtering out long distances (*max_distance* parameter) can help speeding up computations.

Anyway, it will probably be impossible to run a full MAPI analysis with 5,000 samples spreaded on a 100,000 grid on a old laptop with only 2Gio RAM...

6.4 How to kill a MAPI run?

Using pgAdmin3, connect to the PostgreSQL server. From the "Tools" menu, select the "State of the server" item (bottom). This open an new window with currently running queries.

Refresh the list with the button (two rounded arrows, one green one red). Yours should be in orange; check that it is truly yours (start date and hour are displayed)! Click on the orange-square button in order to ask the selected job to terminate. After a few seconds of memory cleaning, the query should be terminated (force a refresh to be sure).

7 Conclusion

In order to obtain the best amazing results while playing with MAPI (and, by the way, for every spatial data-processing), let's remind some key points:

[13] Download from: http://ebe.ulb.ac.be/ebe/SPAGeDi_files/SPAGeDi_1.5_Manual.pdf

- plan a **good sampling coverage** over the whole study area;
- try to avoid large gaps between sampling locations or distant outlyers;
- collect a large number of samples (100 is really short, 500 to 1000 is better);
- limit the variance of the metric (for geneticists: genotype enough loci) for reliable results;
- explore different metrics in the light of their meanings;
- try different values of parameter combinations (eccentricity, distance filtering) in order to assess the robustness of your results.

Overall, the quality of the spatial coverage - as far as the species exists - is of uttermost importance as stated by Thomassen et al. (2010) in the section *"Sampling design for spatial analyses"*, referring to Anderson et al. (2010). We also recommand the reading of Richardson et al. (2016) in order to avoid pitfalls when sampling and also interpreting results in landscape genetics.

When -- one day, let's hope -- hundreds of analyses will be published using MAPI, involving various situations, datasets and interpretations, a wide corpus of expertises will be at hand as a feedback over this new method.

Meanwile, we wish you a pleasant, funny, and original data mining to be carried along with careful interpretations of the results, always enlighted by your knowledge of the model and biological processes involved.

 The MAPI team.

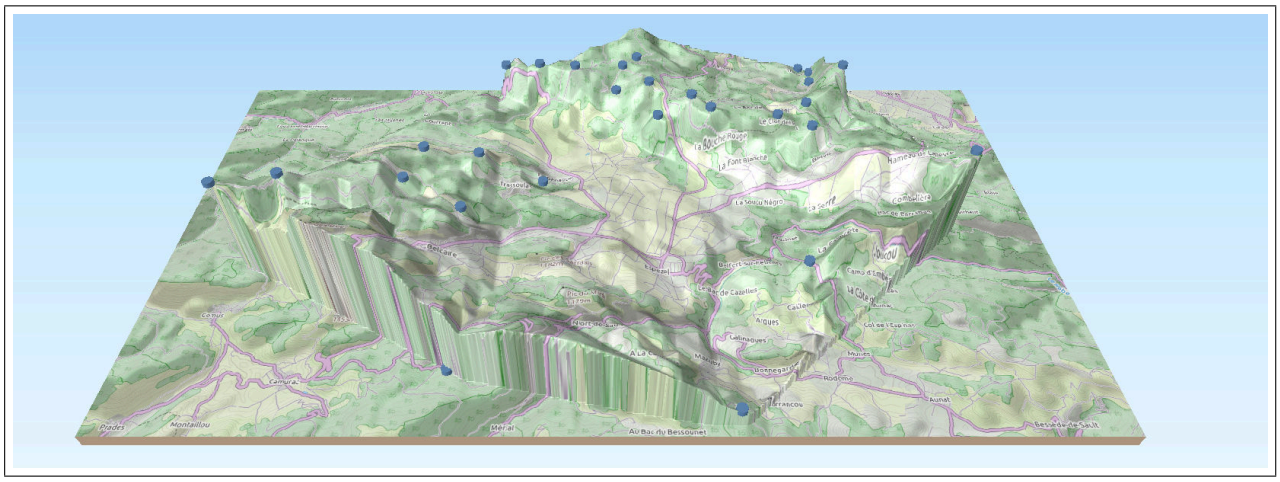


Figure 10: Three-dimentional view of MAPI results for *C. punctatoauratus* (compare with figure 7 on page 28, frame B).
The fake-"elevation" represents in fact the MAPI m_w value and an OpenStreetMap layer is super-imposed: the higher the "elevation", the smallest the genetic distance.
See https://www1.montpellier.inra.fr/CBGP/software/MAPI/showroom/Ecofor2000_3Dmodel/C_punctatoauratus_as_DEM.html for animation.

8 References

Anderson, C. D., Epperson, B. K., Fortin, M.-J., Holderegger, R., James, P. M. A., Rosenberg, M. S., Scribner, K. T., and Spear, S. (2010). Considering spatial and temporal scale in landscape-genetic studies of gene flow. *Molecular Ecology*, 19(17):3565--3575.

Benjamini, Y. and Yekutieli, D. (2001). The control of the False Discovery Rate in multiple testing under dependency. *Annals of Statistics*, 29(4):1165--1188.

Brouat, C., Sennedot, F., Audiot, P., Leblois, R., and Rasplus, J.-Y. (2003). Fine-scale genetic structure of two carabid species with contrasted levels of habitat specialization. *Molecular Ecology*, 12(7):1731--1745.

Desbiez, C., Joannon, B., Wipf-Scheibel, C., Chandeysson, C., and Lecoq, H. (2009). Emergence of new strains of Watermelon mosaic virus in South-eastern France: Evidence for limited spread but rapid local population shift. *Virus research*, 141:201--208.

DiMiceli, C., Carroll, M., Sohlberg, R., Huang, C., Hansen, M., and Townshend, J. (2011). Vegetation Continuous Fields MOD44B, Collection 5, 2000 Percent Tree Cover. ftp://ftp.glcf.umd.edu/glcf/Global_VCF/Collection_5/. University of Maryland, College Park, Maryland. Accessed Feb. 2016.

Gauffre, B., Berthier, K., Inchausti, P., Chaval, Y., Bretagnolle, V., and Cosson, J. F. (2014). Short-term variations in gene flow related to cyclic density fluctuations in the common vole. *Molecular Ecology*, 23:3214--3225.

Gauffre, B., Estoup, A., Bretagnolle, V., and Cosson, J. F. (2008). Spatial genetic structure of a small rodent in a heterogeneous landscape. *Molecular Ecology*, 17:4619--4629.

Hardy, O. and Vekemans, X. (2002). SPAGeDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes*, 2:618--620.

Hengl, T. (2006). Finding the right pixel size. *Computers & Geosciences*, 32(9):1283--1298.

Joannon, B., Lavigne, C., Lecoq, H., and Desbiez, C. (2010). Barriers to gene flow between emerging populations of Watermelon mosaic virus in Southeastern France. *Phytopathology*, 100(12):1373--1379.

Piry, S., Chapuis, M.-P., Gauffre, B., Papaïx, J., Cruaud, A., and Berthier, K. (2016). Mapping Averaged Pairwise Information (MAPI): A new exploratory tool to uncover spatial structure. *Methods in Ecology and Evolution (accepted)*.

Richardson, J. L., Brady, S. P., Wang, I. J., and Spear, S. F. (2016). Navigating the pitfalls and promise of landscape genetics. *Molecular Ecology*, 25(4):849--863.

Rousset, F. (2000). Genetic differentiation between individuals. *Journal of Evolutionary Biology*, 13:58--62.

Sahr, K. (2011). Hexagonal discrete global GRID systems for geospatial computing. *Archiwum Fotogrametrii, Kartografii i Teledetekcji*, Vol. 22:363--376.

Tamura, K. and Nei, M. (1993). Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Molecular Biology and Evolution*, 10:512--526.

Thomassen, H. A., Cheviron, Z. A., Freedman, A. H., Harrigan, R. J., Wayne, R. K., and Smith, T. B. (2010). Spatial modelling and landscape-level approaches for visualizing intra-specific variation. *Molecular Ecology*, 19(17):3532--3548.

9 Appendix: MAPI functions description and parameters

Following function names, parameters and comments are extracted from an installed MAPI extension, like in the "readme.mapi" file. Examples are given in SQL for clarity. You can transer it into R with "paste", "dbSendQuery" and "dbGetQuery" like in the scripts provided.

9.1 Main functions

The tools you will most often use.

MAPI_RunAuto: The easiest way to run MAPI. A good start for a new dataset. Largely commented in section 2.2 on page 14. You may re-use the grid table produced for further computations.

```
FUNCTION: public.MAPI_runauto (
        points_view_name text,
        relations_view_name text,
        beta double precision,
        results_table_name text,
        my_schema_name text DEFAULT NULL::text,
        eccentricity double precision DEFAULT 0.975,
        err_radius double precision DEFAULT 10,
        nb_permutations integer DEFAULT 0,
        inter_groups boolean DEFAULT false,
        d_min double precision DEFAULT NULL::double precision,
        d_max double precision DEFAULT NULL::double precision,
        alpha double precision DEFAULT 0.05
)
RETURNS: integer
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2013-2016
Run a MAPI analysis without grid provided.
The grid is computed on the basis of a beta parameter wich may vary
    from 0.25 (random sampling) to 0.5 (regular sampling) in order
    to estimate the resolution of tje grid.
In the end, lower- and upper- tails are delineated after computation
    of FDR by Benjamini-Yekutieli method.
Input parameters:
  1/ points_view_name: name of the table or view containing sampling points.
      Mandatory fields: point_name text, point_geom geometry(point), point_group text.
      The geometry must be projected (not lat/lon).
  2/ relations_view_name: name of the table or view containing the distance
      (or similarity) values between pairs of points.
      Mandatory fields: point_1_name text, point_2_name text, relation_value float8.
  3/ beta: Value of the sampling parameter from 0.5 (regular sampling)
      to 0.25 (random sampling). Smaller values will lead to finer grids.
  4/ results_table_name: the name of the table that will be built with results
      in schema "my_schema_name".
      Warning! If the table already exists it is erased.
  5/ my_schema_name: the name of the schema containing the tables.
      Defaults to "public"
  6/ eccentricity: the eccentricity of the ellipses.
      Defaults to 0.975.
  7/ err_radius: minimal circle around points.
      Defaults to 10 projection units.
  8/ nb_permutations. Number of permutations for significant areas detection.
      Defaults to 0 (no permutations).
  9/ inter_groups: also analyse relations between different groups. Boolean.
      Defaults to false.
```

```
 10/ d_min: minimal distance between two samples. Float.
      Defaults to NULL (unused).
 11/ d_max: maximal distance between two samples. Float.
      Defaults to NULL (unused).
 12/ alpha for significant areas. Float.
      Defaults to 0.05.
Returns the number of rows in results table.
```

MAPI_RunOnGrid: Intended for more advanced users, or customized runs. You have to provide the grid (which may come from a previous "RunAuto" output). Significant areas detection of significant areas must be run manually (not done automatically). Only one option is available by now: `pointsToCentroid`. It allows to move the sampling locations to the centroid of the cells they belong to. Simplifying the geographical network speed up computations. It is very efficient if your sampling is "patchy" (ie. numerous samples around the same point). The bias on precise location is low for high-resolution grids (large number of cells).

```
FUNCTION: public.MAPI_runongrid (
        points_view_name text,
        relations_view_name text,
        template_grid_name text,
        results_table_name text,
        my_schema_name text DEFAULT NULL::text,
        eccentricity double precision DEFAULT 0.975,
        err_radius double precision DEFAULT 10,
        nb_permutations integer DEFAULT 1000,
        inter_groups boolean DEFAULT false,
        d_min double precision DEFAULT NULL::double precision,
        d_max double precision DEFAULT NULL::double precision,
        options text DEFAULT NULL::text
)
RETURNS: integer
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2013-2016
Runs a MAPI analysis provided views (or tables) containing points localizations and
    relationships between points and a template grid.
Input parameters:
  1/ points_view_name: name of the table or view containing sampling points.
      Mandatory fields: point_name text, point_geom geometry(point), point_group text.
      The geometry must be projected (not lat/lon).
  2/ relations_view_name: name of the table or view containing the distance (or similarity)
      values between pairs of points.
      Mandatory fields: point_1_name text, point_2_name text, relation_value float8.
  3/ template_grid_name: name of the table or view containing the grid.
      Mandatory fields: gid integer primary key, geom geometry(polygon).
  4/ results_table_name: the name of the table that will be built for storing results
      in schema "my_schema_name".
      Warning! If the table already exists it is erased.
  5/ my_schema_name: the name of the schema containing the tables. Defaults to "public"
  6/ eccentricity: the eccentricity of the ellipses. Defaults to 0.975.
  7/ err_radius: minimal circle around points. Defaults to 10 projection units.
  8/ number of permutations. Defaults to 0 (no permutations).
  9/ also analyse relations between different groups. Boolean. Defaults to false.
 10/ minimal distance between two samples. Float. Defaults to NULL (unused).
 11/ maximal distance between two samples. Float. Defaults to NULL (unused).
 12/ options. Text "pointsToCentroid". Defaults to NULL (no options activated).
Returns the number of rows in results table.
```

9.2 Grid building functions

MAPI_GridAuto: This function automatically creates a table containig the geometry of each hexagonal cell. The size of the cells is computed from the table containing sampling points, which allows to determine the study area (convex hull of the sampling points) and the number of sampling locations. Given a beta (β) value (0.5 for regular sampling, 0.25 for random sampling), cell's size is computed in repect to the Nyquist-Shannon sampling theorem.

```
FUNCTION: public.MAPI_gridauto (
        points_view_name text,
        grid_table_name text,
        my_schema_name text,
        beta double precision DEFAULT 0.25,
        full_extent boolean DEFAULT false
)
RETURNS: integer
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2015-2016
Builds an grid of hexagonal cells on the basis of the samples locations.
By default (full_exent = false), the study area is the convex hull of sampling locations.
When full_extent is set to true, the grid will encompass the most external ellipses.
The beta parameter defines the density of cells within this area based on Shannon-Nyquist theorem.
beta=0.5 is suitable for regular samplings, beta=0.25 for random samplings.
Smaller beta values will increase the spatial resolution along with computation time.
Input parameters:
  1/ samples_table name (text),
  2/ grid_table name (text),
  3/ schema name (text),
  4/ beta parameter (float),
  5/ full_extent (boolean, default FALSE).
Returns the number of cells in the grid.
```

MAPI_GridHexagonal: This function allows users to create their own grid, by providing a geographical area and cell half-width (ie. the size of hexagon side). Geographical extent (the_zone) has to be computed apart, eg. by aggregating points, picking the convex hull and buffering the whole stuff.

```
FUNCTION: public.MAPI_gridhexagonal (
        schemaname text,
        tablename text,
        the_zone geometry,
        halfwidth double precision,
        temporary_table boolean DEFAULT false
)
RETURNS: integer
DESCRIPTION:
Modified by Sylvain Piry CBGP - INRA, 2013, from:
http://www.dimensionaledge.com/main/postgis/how-to-create-hexagonal-grids-in-postgis/
Builds hexagonal grid in "schemaname"."tablename" PostGis table covering a geomery set
    in the same SRID given the half-width of each hexagon.
Input parameters:
  1/ schemaname (text),
  2/ tablename (text),
  3/ the_zone (geometry),
  4/ halfwidth (double precision).
Returns the number of cells.
```

MAPI_EstimateHalfwidth: Given the study area and the number of cells expected, this function will compute the cell half-width.

```
FUNCTION: public.MAPI_estimatehalfwidth (
        the_zone geometry,
        nb_cells integer
)
RETURNS: double precision
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2013
Computes the half-width of hexagonal cells given a geometry and
        the approximative number of hexagonal cells expected.
Input parameters:
  1/ the_zone (geometry),
  2/ nb_cells (integer)
Returns the value of the halfwidth.
```

Usage example in SQL: Here we build a permanent (ie. non-temporary) table in the "public" schema with a grid of 5,000 cells covering the whole sampling area, with an additional 1km buffer around. The table named "t_samples" is supposed to contain your samples points with geometry in the field "point_geom". Note the nested query, here named "z", computing the area:

```
SELECT MAPI_GridHexagonal('public', 'my_grid', my_study_area,
                        MAPI_estimateHalfWidth(my_study_area, 5000), false)
FROM (
        SELECT ST_Buffer(ST_ConvexHull(ST_Collect(point_geom)),1000) AS my_study_area
        FROM t_samples
) AS z
```

9.3 Function do detect significant areas

MAPI_Significance: This function applies a False Discovery Rate (FDR) procedure (Benjamini and Yekutieli, 2001). Alternatively, you can run the analyse without this correction ("M_raw" value).

```
FUNCTION: public.MAPI_significance (
        results_table_name text,
        my_schema_name text,
        my_method character varying DEFAULT 'M_BY'::character varying,
        my_alpha double precision DEFAULT 0.05,
        append boolean DEFAULT false
)
RETURNS: integer
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2015-2016
Delineates signifcant polygons by merging contiguous significant cells.
Significance is computed given a alpha between 0.0 and 0.5 (typically 0.05 for 5%).
The "M_raw" method is a simple thresholding of p-values without False Discovery Rate processing.
Method "M_BY" (default) estimate the False Discovery Rate using Benjamini-Yekutieli algorithm.
Cells marked as significant are aggregated together in polygons.
These polygons are stored in two tables, one for the upper tail the other for the lower tail.
The table is named upon results table name and ends in _tails.
The table contains an unique id, the alpha value, the relation group name, the tail
    (lower or upper), the geometry of the polygon and its area.
If run with different alphas, polygons are added to the table with the alpha value used.
Input Parameters:
  1/ The name of MAPI results table (without schema).
  2/ The name of the schema.
  3/ Name of the algorithm among:
```

```
      M_BY: Benjamini-Yekutieli FDR (default),
      M_raw: simple thresholding without FDR.
  4/ alpha, default 0.05 (5%).
  5/ Append - or not - to existing tables. If the table does not exists is created.
Returns the total number of polygons in lower and upper tail.
```

9.4 Other functions

MAPI_Version: Just call: `SELECT MAPI_version();` in order to get MAPI's version number installed in the database.

```
FUNCTION: public.MAPI_version (
)
RETURNS: text
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2013-2016
Returns MAPI current version number and build date
```

9.5 Internal functions

These functions are used within MAPI and you will probably never call them, except if you wish to tweak, hack and/or develop new MAPI functions.

MAPI_MakeEllipse: This function builds an ellipse-shaped polygon, based on samples locations, eccentricity and error-circle radius values. See figure 2 on page 6 for an illustration.

```
FUNCTION: public.MAPI_makeellipse (
        p1 geometry,
        p2 geometry,
        e double precision,
        err_radius double precision DEFAULT NULL::double precision,
        nb_seg integer DEFAULT 8
)
RETURNS: geometry
DESCRIPTION:
Designed by Sylvain Piry & Astrid Cruaud, CBGP - INRA, 2013-2015
This function primarily builds a polygon approaching an ellipse, provided the two focus points,
    given the excentricity and the number of segments per quarter.
The geometry of the points have to be cartesian (no latitude / longitude).
An error circle may be defined around points. If the points are located at the same place a
    circle with given "err_radius" radius (in geometry projection units) is created instead.
Finally, the resulting polygon is the convex hull between the ellipse and a buffer of width
    "err_radius" around the segment joining the two points.
The number of segments per quarter allow to define the quality of the ellipsoidal polygon.
    Default is OK for current uses.
Input parameters:
  1/ point1 (geometry),
  2/ point2 (geometry),
  3/ eccentricity (double precision),
  4/ error circle radius "err_radius" (double precision, optional),
  5/ number of segments per quarter (integer, 8 by default).
Returns the polygon geometry.
```

MAPI_Polygonize: This function aggregates contiguous cells and returns a table of polygons delineating significant areas.

```
FUNCTION: public.MAPI_polygonize (
        cells geometry
)
RETURNS: TABLE(gid integer, geom geometry, area double precision)
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2013-2015
Aggregates contiguous cells from input collection in polygons.
Input parameters:
  1/ cells: geometry collection of polygons
Returns a set of rows: (geom geometry(polygon), area float8)
```

MAPI_RankInDistribution: This function returns the numbre of array members strictly below the value of the *val* parameter, and returns its rank divided by the array length (computation of p-value).

```
FUNCTION: public.MAPI_rankindistribution (
        val double precision,
        perms double precision[]
)
RETURNS: double precision
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2015
Returns the rank in [0,1] of a given value in an array of values.
Counts the number of array members stictly below the value and divides by the array length.
```

MAPI_sd2: This function is used during weighted-variance computations. It computes the weighted sum-of-squares of differences between values and the mean.

```
FUNCTION: public.MAPI_sd2 (
        a double precision,
        vv double precision[]
)
RETURNS: double precision
DESCRIPTION:
Designed by Sylvain Piry, CBGP - INRA, 2016
Computes the sum of squared differences between an array of values and a mean.
Parameters:
  1/ a, the mean, float8.
  2/ vv, the array of values, float8[].
Returns the sum of squares of differences.
```

Note: This function is basically written in plPgSQL but a faster C-language version can be compiled and installed.

10 Appendix: R script templates

You may use the following R scripts as templates for your own usage.

10.1 Running MAPI

Edit connection variables according to your database setups. Run the script from the directory which contains the sample and distances files, or edit files paths.

```
#!/usr/bin/Rscript

# load libraries for PostgreSQL and fast datafiles parsing
library(RPostgreSQL)
library(data.table)

## DATABASE CONNECTION PARAMETERS ##
host <- "localhost"
port <- 5432
user <- "USERNAME"
password <- "PASSWORD"
dbname <- "mapi_demo"
schema <- "public"

## FILES ##
file.samples <- "sim2_12_samples.csv" # sample file with sample name, x y coordinates and group if any
file.metrics <- "sim2_12.arp.gp.spa.out" # metrics file with sample names and metric value
file.metrics.columns <- c("name_i", "name_j", "i", "j", "sd", "pw", "all_loci") # spagedi header

## TABLES & VIEWS NAMES ##
table.samples <- "t_sim2_12_samples"
table.metrics <- "t_sim2_12_metrics"
table.results <- "t_sim2_12_results"
view.samples <- paste("v_",table.samples, sep="")
view.metrics <- paste("v_",table.metrics, sep="")

# database connection
con <- dbConnect(PostgreSQL(), host=host, port=port, user=user, password=password, dbname=dbname)
# clean existing views ot tables, if any
dbSendQuery(con, paste("DROP VIEW IF EXISTS ",paste(schema, view.samples, sep="."),";"))
dbSendQuery(con, paste("DROP VIEW IF EXISTS ",paste(schema, view.metrics, sep="."),";", sep=""))
dbSendQuery(con, paste("DROP TABLE IF EXISTS ",paste(schema, table.samples, sep="."),"  CASCADE;", sep=""))
dbSendQuery(con, paste("DROP TABLE IF EXISTS ",paste(schema, table.metrics, sep="."),"  CASCADE;", sep=""))

# load samples
s <- as.data.frame(fread(file.samples, header=TRUE, sep=","))
dbWriteTable(con, c(schema, table.samples), s, overwrite=TRUE, row.names=FALSE)
# load metrics
d <- as.data.frame(fread(file.metrics, header=TRUE))
colnames(d) <- file.metrics.columns
dbWriteTable(con, c(schema, table.metrics), d, overwrite=TRUE, row.names=FALSE)

## VIEWS FORMATTED FOR MAPI ##
# view for samples, rename columns and computes geographical information
dbSendQuery(con, paste("CREATE VIEW ",paste(schema, view.samples, sep="."),"  AS
        SELECT ind::text AS point_name,
                sampling_year::text AS point_group,
                ST_SetSRID(ST_MakePoint(x, y), 3258) AS point_geom
        FROM  ",paste(schema, table.samples, sep="."),";", sep=""))

# view for metrics, keep only mandatory columns
dbSendQuery(con, paste("CREATE VIEW ",paste(schema, view.metrics, sep="."),"  AS
                SELECT name_i AS point_1_name, name_j AS point_2_name, all_loci AS relation_value
                FROM  ",paste(schema, table.metrics, sep="."),";", sep=""))

## MAPI PARAMETERS ##
mapi.beta <- 0.5 # 0.5 for regular sampling, 0.25 for random sampling
mapi.nb_permutations <- 1000 # 1000 is ok for 5%
```

```
mapi.eccentricity <- 0.975 # 0.975 is default value
mapi.error_circle <- 10 # 10 m is typical GPS error
mapi.inter_groups <- "false" # not relevant here (no groups)
mapi.min_distance <- "NULL" # "NULL" for no distance filtering
mapi.max_distance <- "NULL" # "NULL" for no distance filtering
mapi.alpha <- 0.05 # 0.05 for 5% tails

## START AUTOMATIC MAPI COMPUTATION
n <- dbGetQuery(con, paste("SELECT MAPI_RunAuto(
                        '",view.samples,"',
                        '",view.metrics,"',
                        ",mapi.beta,",
                        '",table.results,"',
                        '",schema,"',
                        ",mapi.eccentricity,",
                        ",mapi.error_circle,",
                        ",mapi.nb_permutations,",
                        ",mapi.inter_groups,",
                        ",mapi.min_distance,",
                        ",mapi.max_distance,",
                        ",mapi.alpha,");", sep=""), verbose=TRUE)
# ... be patient (~ half-hour for provided example simulated files)
```

10.2 Building maps (Windows/Linux version)

This version work even when gdal does not include PostgreSQL/PostGIS driver. Geographical information is converted in text in the query and is reverted to geographical format in R. This script re-use variables defined in previous MAPI script (section 10.1 on the facing page). Append it at the end of your main MAPI script file.

```
## MAPPING WITH R (WINDOWS)
library(rgeos)
library(sp)
library(rgdal)
library(RColorBrewer)
library(lattice)
library(latticeExtra)

file.results.pdf <- "sim2_12_win.pdf" # final map file

srid <- as.numeric(unlist(dbGetQuery(con, paste("SELECT DISTINCT ST_SRID(geom)
        FROM ", paste(schema, table.results, sep="."), ";", sep=""))))
EPSG <- make_EPSG()
p4s <- EPSG[which(EPSG$code == srid), "prj4"]

resu0 <- dbGetQuery(con, paste("SELECT id, cell_gid, averaged_value, relation_group,
        ST_AsText(geom) AS tgeom
        FROM ", paste(schema, table.results, sep="."), ";", sep=""))
row.names(resu0) <- resu0$id
for (i in seq(nrow(resu0))) {
  if (i == 1) {
        spResu <- readWKT(resu0$tgeom[i], resu0$id[i], p4s)
  } else {
        spResu <- rbind(spResu, readWKT(resu0$tgeom[i], resu0$id[i], p4s))
  }
}
resu <- SpatialPolygonsDataFrame(spResu, resu0[-2])

tails0 <- dbGetQuery(con, paste("SELECT id, method, tail, relation_group,
        ST_AsText(geom) AS tgeom
        FROM ", paste(schema, paste(table.results, "tails", sep="_"), sep="."), "
        WHERE method LIKE 'M_BY';", sep=""))
row.names(tails0) <- tails0$id
for (i in seq(nrow(tails0))) {
  if (i == 1) {
        spTails <- readWKT(tails0$tgeom[i], tails0$id[i], p4s)
  } else {
```

```
        spTails <- rbind(spTails , readWKT(tails0$tgeom[i], tails0$id[i], p4s))
  }
}
tails <- SpatialPolygonsDataFrame(spTails , tails0[-2])
lower.tail <- tails[tails$tail=="lower", ]
upper.tail <- tails[tails$tail=="upper", ]

samp0 <- dbGetQuery(con, paste("SELECT point_name, point_group,
        ST_AsText(point_geom) AS tgeom
        FROM ", paste(schema, view.samples, sep="."), ";", sep=""))
row.names(samp0) <- samp0$point_name
for (i in seq(nrow(samp0))) {
  if (i == 1) {
        spSamp <- readWKT(samp0$tgeom[i], samp0$point_name[i], p4s)
  } else {
        spSamp <- rbind(spSamp, readWKT(samp0$tgeom[i], samp0$point_name[i], p4s))
  }
}
samp <- SpatialPointsDataFrame(spSamp, samp0[-2])


# colorscale
aec <- rainbow(512, start=0, end=0.8)
vals <- sort(resu$averaged_value)
scale.min <- vals[1]
scale.max <- vals[length(vals)]
scale.by <- (scale.max - scale.min) / 20
at.std <- seq(scale.min, scale.max+scale.by, by=scale.by)
at <- sort(at.std)

pl <- spplot(resu, "averaged_value", col="transparent", col.regions=aec, colorkey=TRUE, at=at,
                main=list(label=table.results), sp.layout=c("sp.points", samp, col="black", pch=15, cex=0.7)
        )
if (nrow(upper.tail@data) > 0) { pl <- pl + layer(sp.polygons(upper.tail, fill=c("transparent"), lwd=5 )) }
if (nrow(lower.tail@data) > 0) { pl <- pl + layer(sp.polygons(lower.tail, fill=c("transparent"), lwd=2 )) }

# print plot to pdf
pdf(file.results.pdf)
print(pl) # print !!!
dev.off()
```

10.3 Building maps (Linux version)

You can use this (smaller) script instead of previous one if your gdal library includes PostgreSQL/PostGIS driver. This script re-use variables defined in previous MAPI script (section 10.1 on page 44). Append it at the end of your main MAPI script file.

```
## MAPPING WITH R (LINUX)
# load libraries for mapping
library(sp)
library(rgdal)
library(RColorBrewer)
library(lattice)
library(latticeExtra)

file.results.pdf <- "sim2_12_lin.pdf" # final map file

# reads spatial tables
OGRstring <- paste("PG:host=",host," port=",port," user=",user," password=",password," dbname=",dbname, sep="")
if (schema == "public") { # surprisingly, OGR don't accept "public" as a schema...
        samp <- readOGR(OGRstring, view.samples)
        resu <- readOGR(OGRstring, table.results)
        table.tails <- paste(table.results,"tails", sep="_")
} else {
        samp <- readOGR(OGRstring, paste(schema, view.samples, sep="."))
        resu <- readOGR(OGRstring, paste(schema, table.results, sep="."))
        table.tails <- paste(schema, paste(table.results,"tails", sep="_"), sep=".")
```

```
}
tails.info <- ogrInfo(OGRstring, table.tails)
if (tails.info$nrows > 0) { # read tails table only if data available
        tails <- readOGR(OGRstring, table.tails)
        # filter "BY" method and separates lower and upper tails
        lower.tail <- tails[tails$tail=="lower" & tails$method=="M_BY", ]
        upper.tail <- tails[tails$tail=="upper" & tails$method=="M_BY", ]
} else { # nothing to plot, but empty new object avoids crash
        lower.tail <- new("SpatialPolygonsDataFrame")
        upper.tail <- new("SpatialPolygonsDataFrame")
}

# colorscale
aec <- rainbow(512, start=0, end=0.8)
vals <- sort(resu$averaged_value)
scale.min <- vals[1]
scale.max <- vals[length(vals)]
scale.by <- (scale.max - scale.min) / 20
at.std <- seq(scale.min, scale.max+scale.by, by=scale.by)
at <- sort(at.std)

# spatial plot
pl <- spplot(resu, "averaged_value", col="transparent", col.regions=aec, colorkey=TRUE, at=at,
                main=list(label=table.results),
                sp.layout=c("sp.points", samp, col="black", pch=15, cex=0.7))
if (nrow(upper.tail@data) > 0) { pl <- pl + layer(sp.polygons(upper.tail, fill=c("transparent"), lwd=5 )) }
if (nrow(lower.tail@data) > 0) { pl <- pl + layer(sp.polygons(lower.tail, fill=c("transparent"), lwd=2 )) }
# print plot to pdf
pdf(file.results.pdf)
print(pl) # print !!!
dev.off()
```