

SEXTANTE User's manual (v1.0)

Víctor Olaya
Edition 1.0 — Rev. December 12, 2011

SEXTANTE User's manual
Copyright ©2012 Victor Olaya

Edición 1.0
Rev. December 12, 2011

Permission is granted to copy, distribute and modify this work according to the terms of the Creative Common Attribution license under which it is distributed. More information can be found at <http://www.creativecommons.org>. License applies to the text, as well as to the images created by the author, which are all the ones contained in this text except when otherwise stated.

This text can be downloaded in several formats, including editable ones, at <http://www.sextantegis.com>.

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Basic elements of the SEXTANTE GUI	1
2	The SEXTANTE toolbox	5
2.1	Introduction	5
2.2	The algorithm dialog	6
2.2.1	The parameters tab	7
2.2.2	The analysis region tab	9
2.3	Data objects generated by SEXTANTE algorithms	11
2.4	Context help	13
2.5	Configuring SEXTANTE	15
2.5.1	General	15
2.5.2	Folders	16
2.5.3	Model	16
2.5.4	GRASS, SAGA and other additional algorithm providers	16
2.6	Iterative execution of algorithms	16
3	The SEXTANTE graphical modeler	19
3.1	Introduction	19
3.2	Definition of inputs	20
3.3	Definition of the workflow	21
3.4	Editing the model	23
3.5	Saving and loading models	23
4	The SEXTANTE batch processing interface	25
4.1	Introducción	25
4.2	The parameters table	25
4.3	Filling the parameters table	26
4.4	Setting the output region	28
4.5	Executing the batch process	29
5	The SEXTANTE command–line interface	31
5.1	Introduction	31
5.2	The interface	31

5.2.1	Getting information about data	32
5.3	Getting information about algorithms	33
5.4	Running an algorithm	34
5.5	Adjusting the analysis region	35
5.6	Managing layers from the command-line interface	36
5.7	Creating scripts and running them from the toolbox	36
6	The SEXTANTE history manager	39
6.1	Introduction	39
7	Configuring algorithm providers	41
7.1	Introduction	41
7.2	SAGA	41
7.3	GRASS	41
7.3.1	R	42
7.4	Some important notes	42
7.5	More about the GRASS interface	43
7.5.1	Usage notes and limitations	43
	Message output from GRASS modules	44
	Graphical interface	44
	Vector data exchange	44
	Raster data exchange	45
	Topology	45
	The GRASS region	45
7.5.2	Windows notes	45
7.5.3	Notes on specific modules	46
	r.colors(.stddev)	46
	r.in/out.gdal	46
	r.mapcalculator	46
	r.null	46
	v.in/out.ogr	46
	v.surf.bspline	47
	v.surf.idw	47
	v.to.3d	47
7.5.4	Technical details	47
7.6	Creating R scripts	48

Introduction

1.1 Introduction

Welcome to this introduction to SEXTANTE. This text is targeted at those using geospatial algorithms from the SEXTANTE library through the graphical elements also included in the library.

SEXTANTE is integrated into some of the most popular Java desktop GIS, and using it is identical no matter which GIS you are using. If you have a GIS application which incorporates SEXTANTE as its analysis platform, then this manual is for you.

Particular information about SEXTANTE algorithms is not found in this text. The user should refer to the context help system instead.

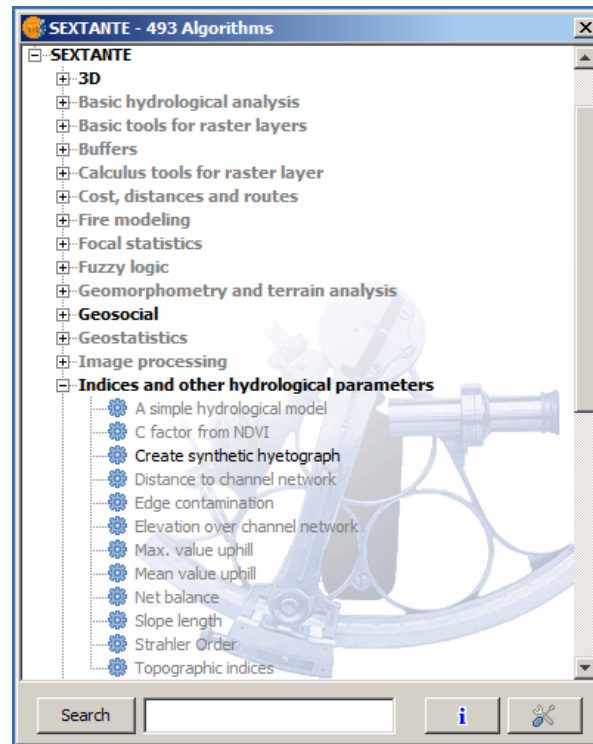
1.2 Basic elements of the SEXTANTE GUI

There are four basic elements in the SEXTANTE GUI, which are used to run SEXTANTE algorithms for different purposes. Choosing one tool or another will depend on the kind of analysis that is to be performed and the particular characteristics of each user an project.

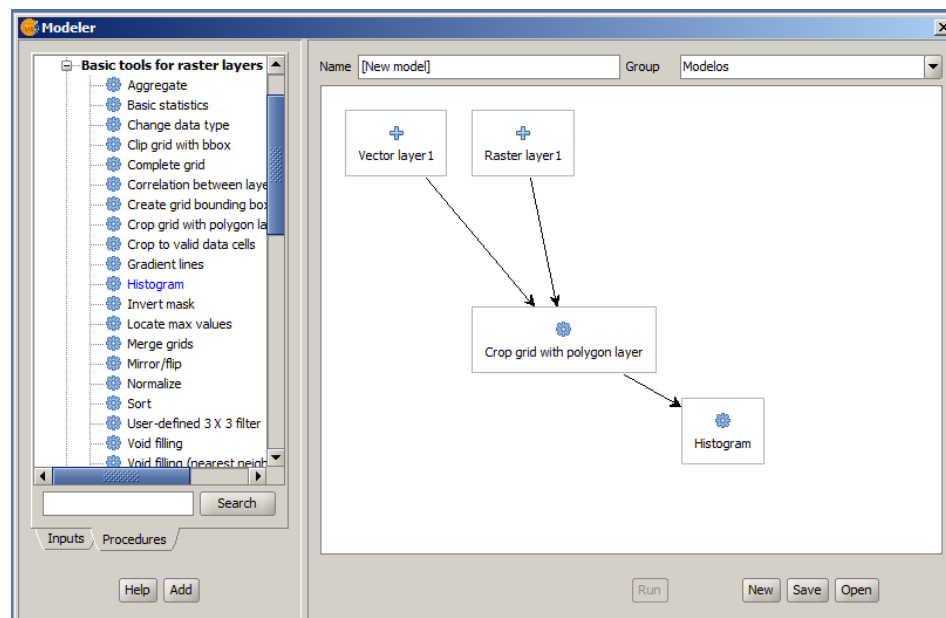
Depending on the implementation of the GIS application you are using, these elements can be accesed through menu entries (usually under a menu group named “SEXTANTE”) or a toolbar like the one shown next.



- The SEXTANTE toolbox. The main element of the SEXTANTE GUI, it is used to execute a single algorithm or run a batch process based on that algorithm.



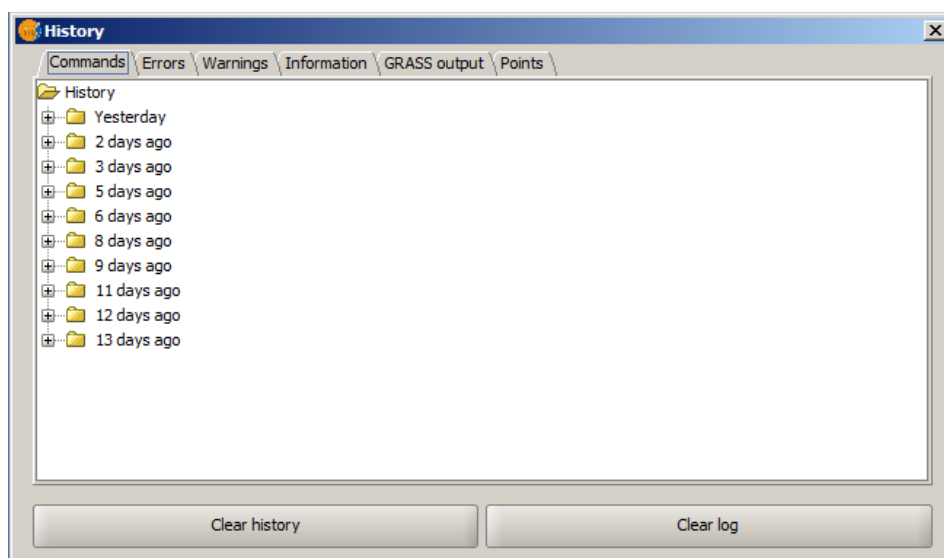
- The SEXTANTE graphical modeler. Several algorithms can be combined graphically using the modeler to define a workflow, creating a single process that involves several sub-processes



- The SEXTANTE command-line interface. Advanced users can use this interface to create small scripts and call SEXTANTE algorithms from them



- The SEXTANTE history manager. All actions performed using any of the aforementioned elements are stored in a history file and can be later easily reproduced using the history manager

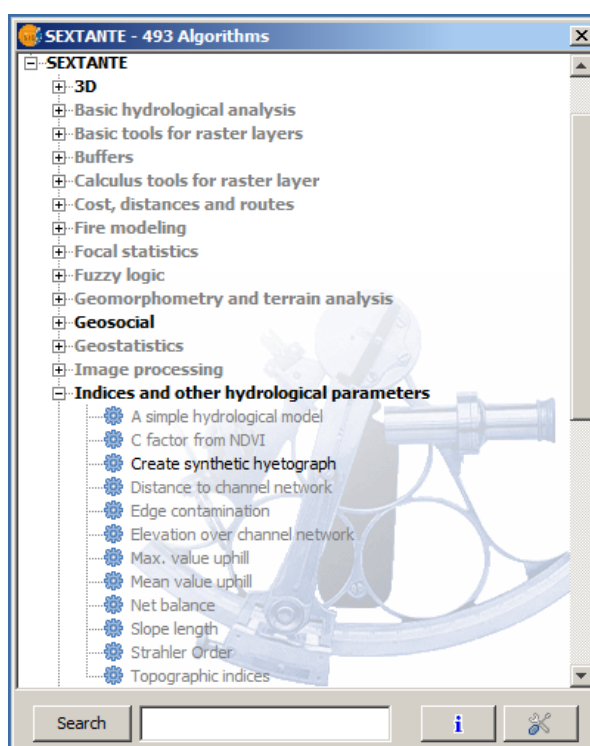


Along the following chapters we will review each one of this elements in detail.

The SEXTANTE toolbox

2.1 Introduction

The *Toolbox* is the main element of the SEXTANTE GUI, and the one that you are more likely to use in your daily work. It shows the list of all available algorithms grouped in different blocks, and is the access point to run them whether as a single process or as a batch process involving several executions of a same algorithm on different sets of inputs.



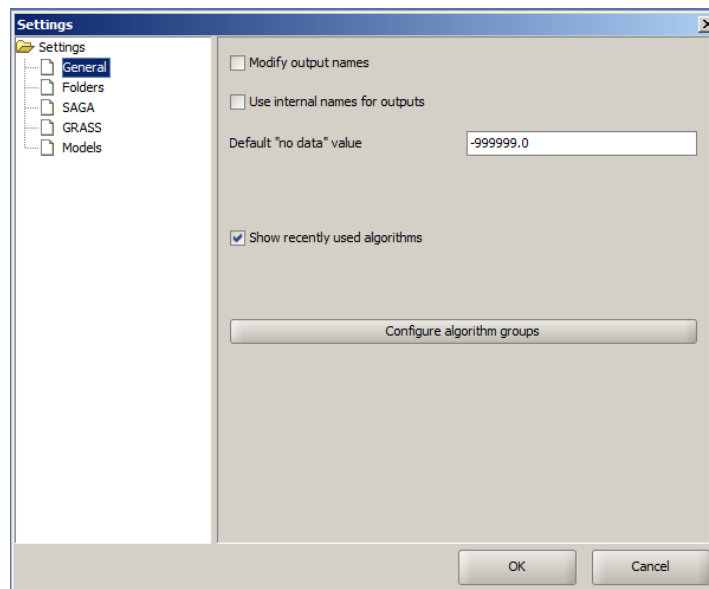
Depending on the data available in the GIS, you will be able to execute an algorithm or not. When there is enough data for the algorithm to be executed (i.e. the algorithm requires raster layers and you have raster layer already loaded into the GIS), its name is shown in black, otherwise, it is shown in grey.

In the lower part of the toolbox you can find a text box and a search button. To reduce the number of algorithms shown in the toolbox and make it easier to find the one you need, you can enter any word or phrase on the text box and click on the search button. SEXTANTE

will search the help files associated to each algorithm and show only those algorithms that include the word or phrase in their corresponding help files. To show all the algorithms again, make a search with an empty string.

Notice that, as you type, the number of algorithms in the toolbox is reduced. A search is performed as you type, but only on the algorithm names, not the help files. You can use this also to quickly find an algorithm. In case you want to perform a full search and look for a given word not just in the algorithm names but also in their associated help files, click the Search button or hit the Enter key.

The configuration button can be found on the search panel as well. It gives access to a new dialog that you can use to configure SEXTANTE.

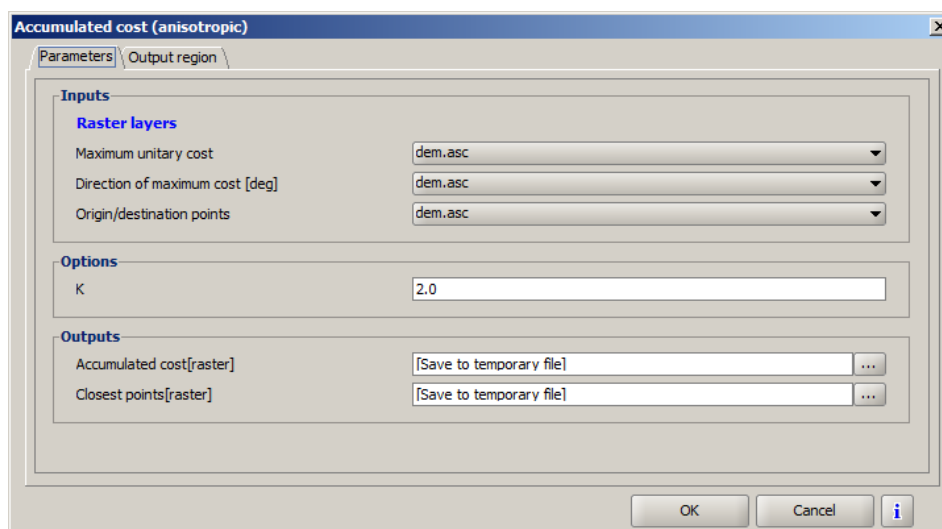


The meaning of each one of its parameters will be explained in the following pages.

To execute an algorithm, just double-click on its name in the toolbox.

2.2 The algorithm dialog

Once you double-click on the name of the algorithm that you want to execute, a dialog similar to the next one is shown (in this case, the dialog corresponds to the *Anisotropic cost* algorithm).



This dialog is used to set the input values that the algorithm needs to be executed.

There is a main tab named *Parameters* where input values and configuration parameters are set. This tab has a different content depending on the requirements of the algorithm to be executed, and is created automatically based on those requirements. On the left side, the name of the parameter is shown. On the right side the value of the parameter can be set.

Most algorithms have an additional tab named *Output extent*. This tab is used to define the region that wants to be analyzed, in case you do not want to perform analysis on the whole extent defined by the input layers. Also, it should be used to set the characteristics of output raster layers, specifying its extent and its cell size.

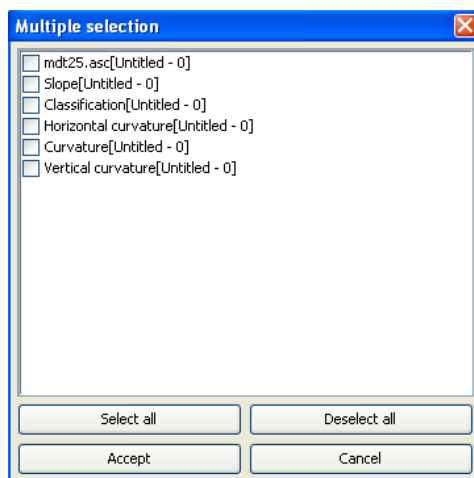
On the lower part of the window there is a help button. Click on it to see the context help related to the current algorithm, where you will find detailed description of each parameter and each output generated by the algorithm.

2.2.1 The parameters tab

Although the number and type of parameters depends on the characteristics of the algorithm, the structure is similar for all of them. The parameters found on the parameters tab can be of one of the following types.

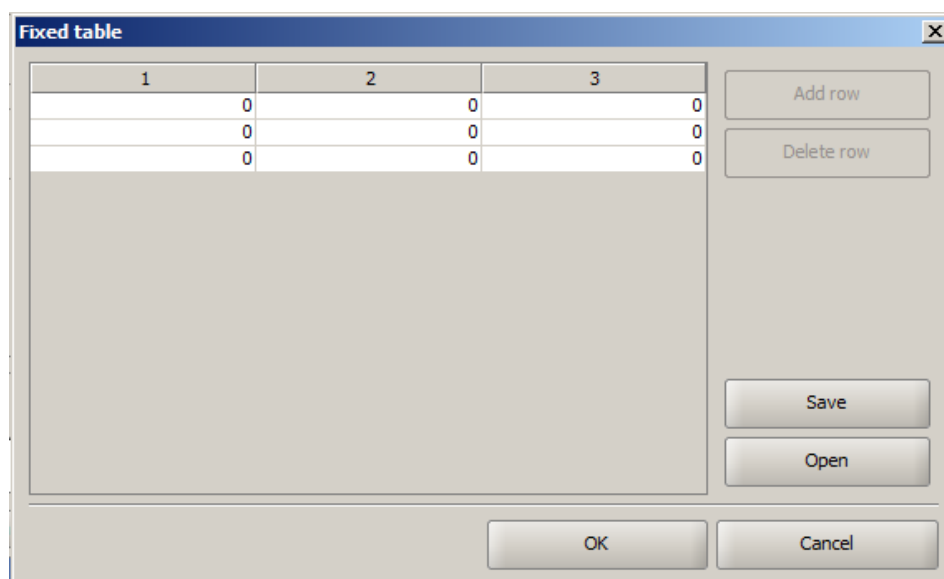
- A raster layer, to select from a list of all the ones available in the GIS application
- A vector layer, to select from a list of all the ones available in the GIS application
- A table, to select from a list of all the ones available in the GIS application
- A method, to choose from a selection list of possible options
- A numerical value, to be introduced in a text box.
- A text string, to be introduced in a text box
- A field, to choose from the attributes table of a vector layer or a single table selected in another parameter.
- A band, to select from the ones of a raster layer selected in another parameter. In both this and the previous type of parameter, the list of possible choices depends on the value selected in the parent parameter.

- A list of elements (whether raster layers, vector ones or tables), to select from the list of the ones available in the GIS application. To make the selection, click on the small button on the left side of the corresponding row to see a dialog like the following one.



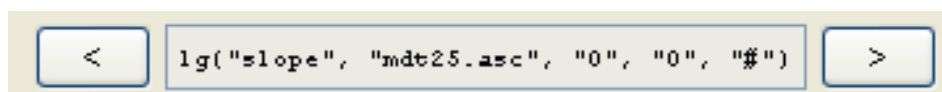
- A file or folder
- A point, to be introduced as a pair of coordinates in two text boxes (X and Y coordinates). Alternatively, you can click on the button on the right side and select one of the points captured using the *Coordinate Capture* tool (you will find it along with the other SEXTANTE tools. Just select it and click on a view or map in your GIS, and it will get the coordinates of the point where you have clicked).
- A small table to be edited by the user. These are used to define parameters like lookup tables or convolution kernels, among others.

Click on the button on the right side to see the table and edit its values.



Depending on the algorithm, the number of rows can be modified or not, using the buttons on the right side of the window.

If you have previously executed an algorithm (whether in this work session or in another one), you will find an additional component in the lower left part of the parameters tab.



By default, in this the parameters are set to the values they had in the last execution. Using the arrow buttons you can change to the values used in previous executions, browsing the SEXTANTE history.

2.2.2 The analysis region tab

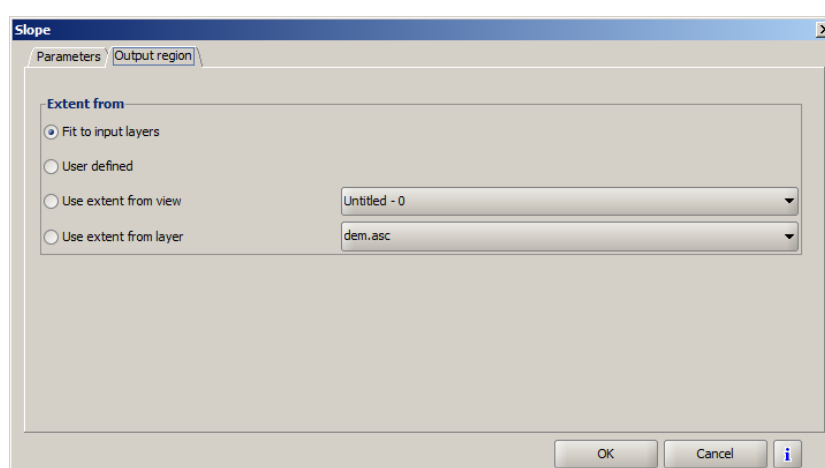
The *Analysis region* tab is found in those algorithms in which the user can select the extent of the region to be used for analysis. In most cases, this extent is also used to generate new layers. This is particularly important in the case of raster layers, and in that case not only the extent is needed, but also a value for the cellsize.

Unlike in most GIS, when combining several raster layers as input for an algorithm, they do not have to have the same extent and cellsize in order to process them together. That is, layers don't have necessarily to "match" between them. Instead, the characteristics of the analysis region (which are the ones used for the resulting raster layers, in case the algorithm generates such an output) are defined and SEXTANTE performs the corresponding resampling and cropping needed to generate layer with those characteristics.

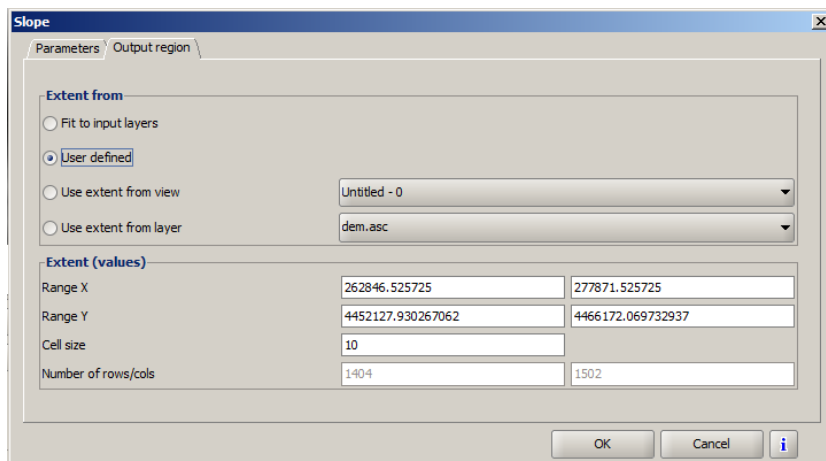
It is responsibility of the user to enter adequate values and be aware of the limitations of this mechanism, so as to generate cartographically sound results. (i.e. you can select a small cell size for the resulting raster layers, but if the input layers you are using have a coarse resolution the results will not be geographically sound).

The following options are available in the *Analysis region* tab:

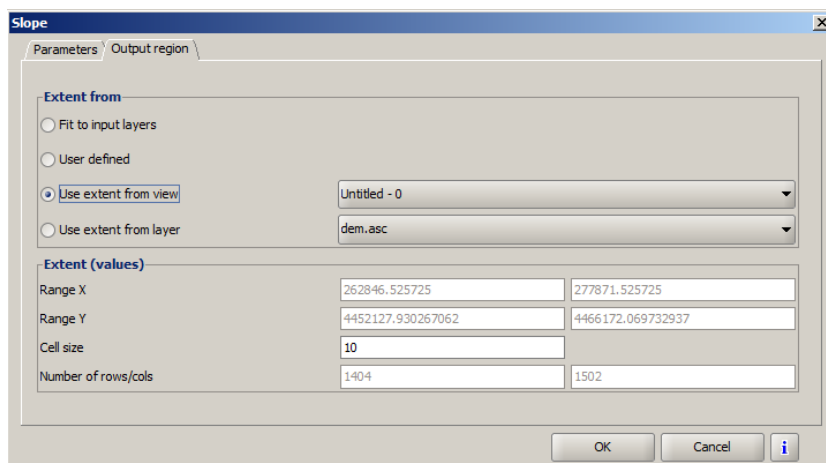
- Fit to input layers. By default, the extent is set based on the input layers. The minimum extent needed to cover all the input layers is used.



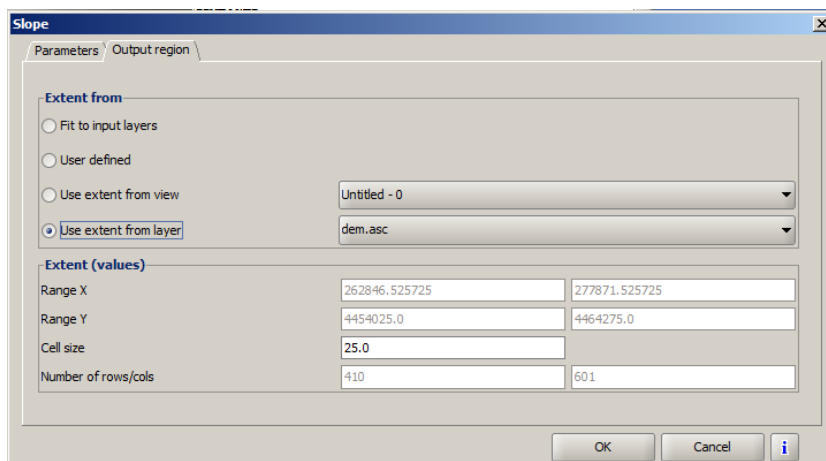
- User defined. The coordinates of the boundaries of the extent and the cellsize are both defined manually, entering the desired values in the corresponding text boxes. You will be prompted to enter a cellsize always, no matter if the algorithm is just a vector one. Just ignore it and leave the default value, since the algorithm will not use it.



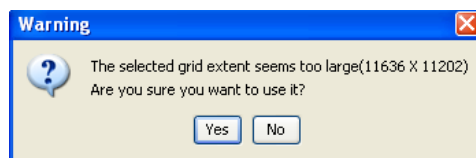
- Use predefined extent. Depending on the GIS you are using, this option will let you use predefined extents like, for instance, the extent of one of the views currently opened.



- Use extent from layer. The extent of a layer can be used as well to define the analysis region, even if the layer is not used as input to the algorithm. If the selected layer is a vector one, the cellsize will have to be entered manually, since vector layers do not have an associated cellsize.



If an option other than the automatic fitting is selected, SEXTANTE will check that the values are correct and the resulting raster layers will not be too large (due to, for instance, a wrong cell size). If the output layers seems to large, SEXTANTE will show the next message dialog to ensure that the user really wants those layers to be created.



Not all algorithms have the first option available, since not all algorithms that generate layers take some other similar layer as input. The interpolation algorithms, for instance, take a vector layer and create a raster one. The extent and cellsize of the latter has to be manually defined, since it cannot be set based solely on the input vector layer (vector layers do not have a cellsize value).

2.3 Data objects generated by SEXTANTE algorithms

Data objects generated by SEXTANTE can be of any of the following types:

- A raster layer
- A vector layer
- A table
- A graphical result (chart, graph, etc.)
- A text-only HTML-formatted result

Layers and tables can be saved, and the parameters window will contain a text box corresponding to each one of these outputs, where you can type the output channel to use for saving it. An output channel contains the information needed to save the resulting object somewhere. In the most usual case, you will save it to a file, but the architecture of SEXTANTE allows for any other way of storing it. For instance, a vector layer can be stored in a database or even uploaded to a remote server using a WFS-T service. Although solutions like these are not yet implemented, SEXTANTE can now easily handle them, and we expect to add new kinds of output channels in a near future.

To select an output channel, just click on the button on the right side of the text box. You will see a dialog with several tabs, each of them containing the elements needed to define a certain kind of output channel. Just go to the tab representing the one that you want and enter the information required. For instance, the File tab is just a file chooser where you have to select a file or type its name. When you are done, click on the OK button.

The first tab you see contains two general options, with a button for each one:

- Save to temporary file
- Do not create this output

Clicking on the “Save to temporary file” button will cause SEXTANTE to select a temporary file for saving the resulting data object. That file will be deleted once you exit the GIS and SEXTANTE is shut down. This options should be used when you are executing an

algorithm and want to use its results as intermediate data, but do not want to keep them permanently.

Clicking on the “Do not create this output” output will cause the output to be generated, but not stored anywhere and not added to your GIS. This is useful if the algorithm you are running generates several outputs but you are not interested in all of them.

An “Overwrite” option will appear in those algorithms that allow overwriting of input layers. In that case, the algorithm will replace the data of an input layer (the algorithm itself has to *know* which layer from all the input ones) instead of generating a new one. A warning might be shown if the input layer cannot be overwritten (such as, for instance, a layer on a database that the user has no rights to edit). This option is currently only available for vector layers, and just in some algorithms where it makes sense.

Since file outputs are the most common ones, we will give some extra information about them.

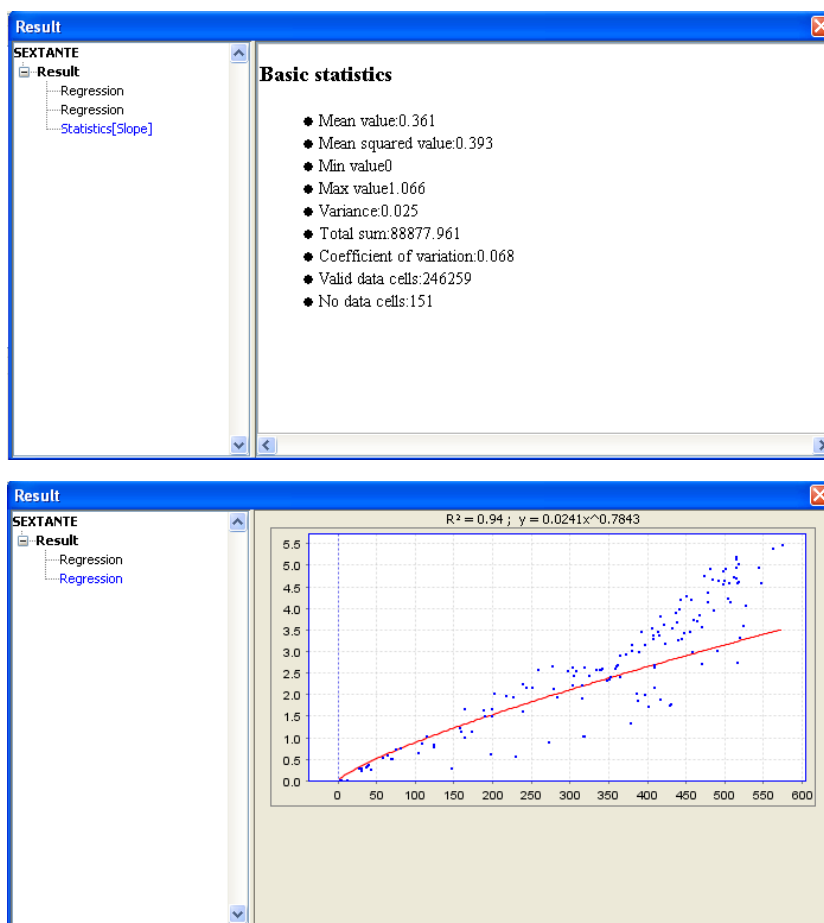
The format of the output is defined by the filename extension. The supported formats depend on the ones supported by the GIS onto which SEXTANTE is running. That means that SEXTANTE usually supports all the formats that the GIS is capable of writing. To select a format, just select the corresponding file extension. If the extension of the filepath you entered does not match any of the supported ones, a default extension (usually dbf for tables, tif for raster layers and shp for vector ones) will be appended to the filepath and the file format corresponding to that extension will be used to save the layer or table.

You can set a default folder for output data objects. Go to the configuration dialog (you can open it from the toolbox), and in the “Folders” tab you will find a text box named “Output folder”. This output folder is used as the default path in case you type just a filename with no path (i.e. `myfile.shp`) when executing an algorithm.

Sometimes, layers might have names that include special characters. For example, if you rasterize a layer named “mylayer”, the result will be a new layer named “mylayer[rasterized]”. Those brackets can cause you some problems if you later want to use that layer as input for the raster calculator, or from the command-line interface, so it can be a good idea to remove them (the same happens with other characters such as “á” or “ñ” that might appear if you use SEXTANTE in spanish). You can tell SEXTANTE to automatically replace those characters with valid standard ones. To do so, open the configuration dialog and select the “General” group. Select the check box with the label “Modify output names”.

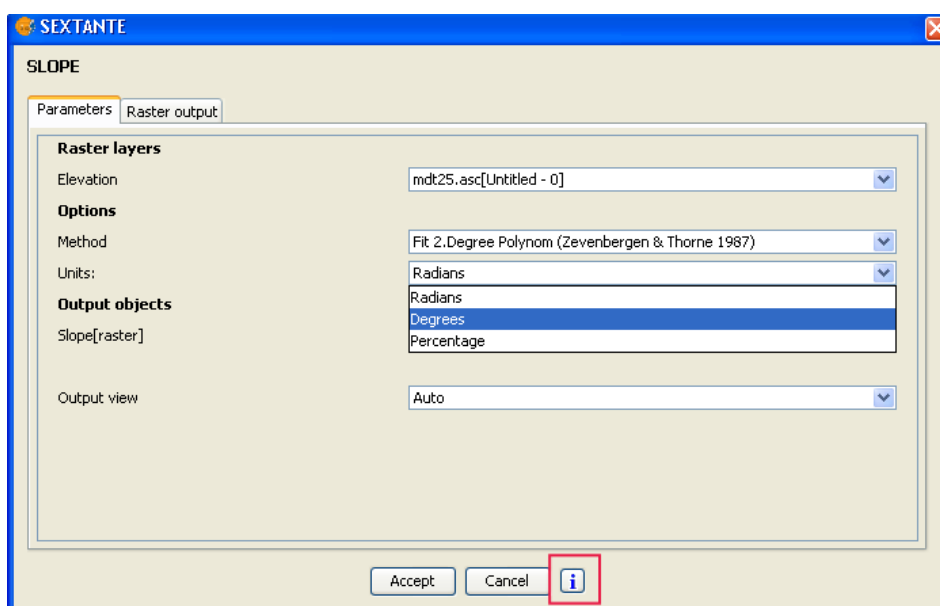
Apart from raster layers and tables, SEXTANTE also generates graphics and texts. These results are kept in memory and shown at the end of the algorithm execution in a new dialog. This dialog will keep the results produced by SEXTANTE during the current session, and can be shown at any time using the *Results* button.

You can save graphical results as images in png format, and texts as HTML files. Right-click on the name of the result in the tree on the left hand of the window and select *Save as....*



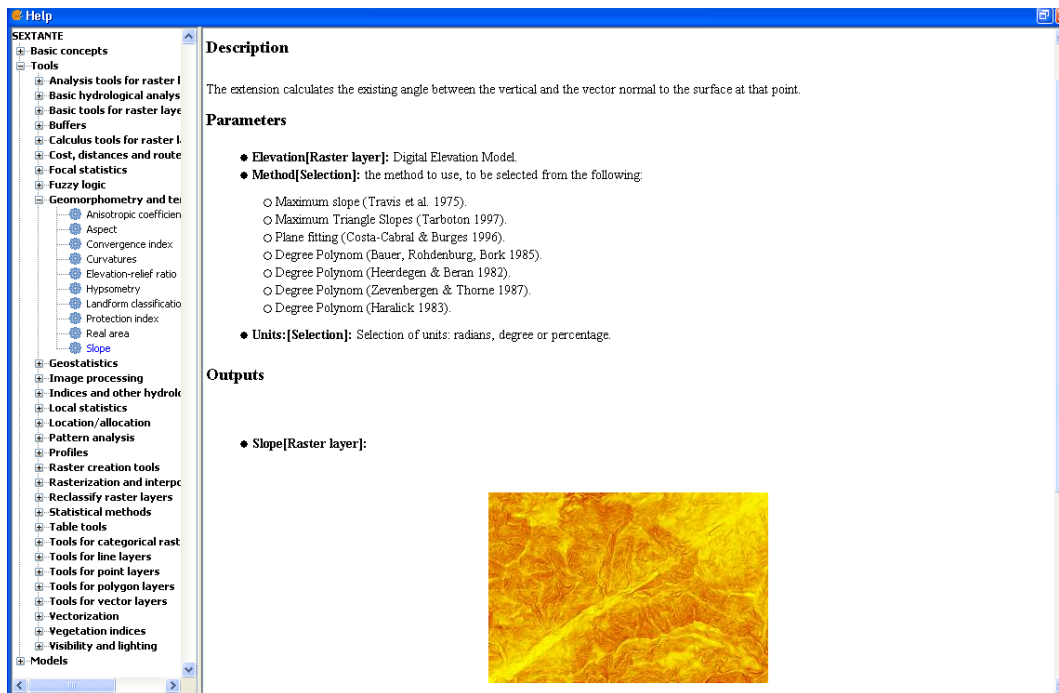
2.4 Context help

Each SEXTANTE algorithm has its own context help file, which provides detailed information about the meaning of each input parameter and each output object, and gives hints about its usage. To access the context help system, click on the button that you will find in the algorithm dialog, or right-click on its name on the toolbox and then select *Show help*.



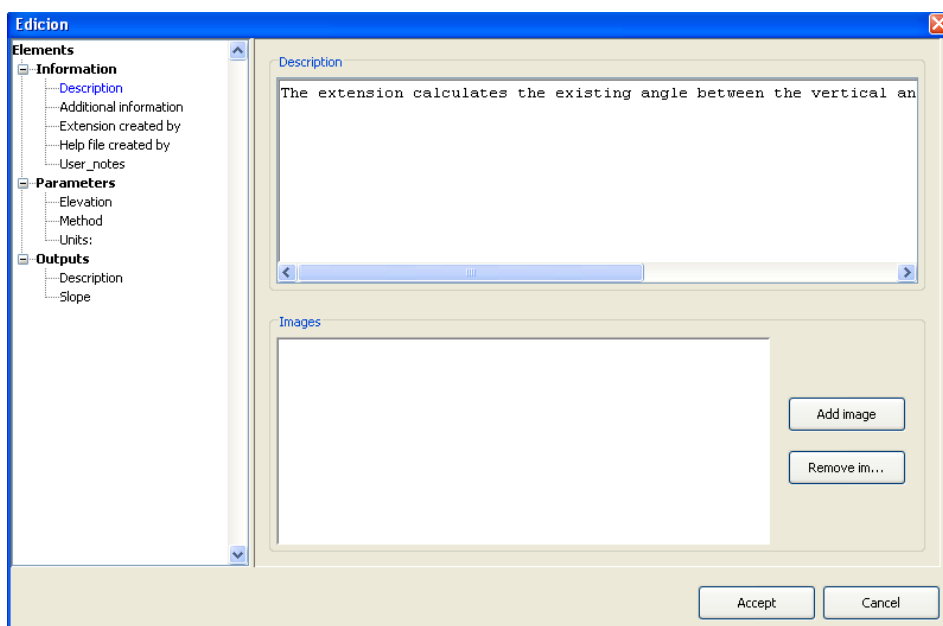
The context help system contains not only information about each algorithm, but also descriptions of each one of the elements of the SEXTANTE GUI like the text you are reading now. You will find a help button in each element, which will take you to the corresponding help file.

Selecting the *Show help* option from the toolbar, you can access the whole SEXTANTE context help. You will see a window like the one shown next.



Just click on the topic you want to read on the left-hand side of the window, and its corresponding text will be shown in the right-hand side.

Help files associated to SEXTANTE algorithms are stored as XML files, and can be edited using the help authoring tools included with SEXTANTE. Right click on the name of the algorithm in the context help window and select *Edit help* to get to the following window:



On the left-hand side you can select any of the elements to be documented (input parameter and outputs, along with other fixed field such as a general description of the algorithm). Then use the right-hand side boxes to enter the text associated to that element or add images.

2.5 Configuring SEXTANTE

As we have seen, the configuration button in the lower part of the toolbox gives access to a new dialog where you can configure how SEXTANTE works. Configuration parameters are structured in separate blocks that you can select on the left-hand side of the dialog.

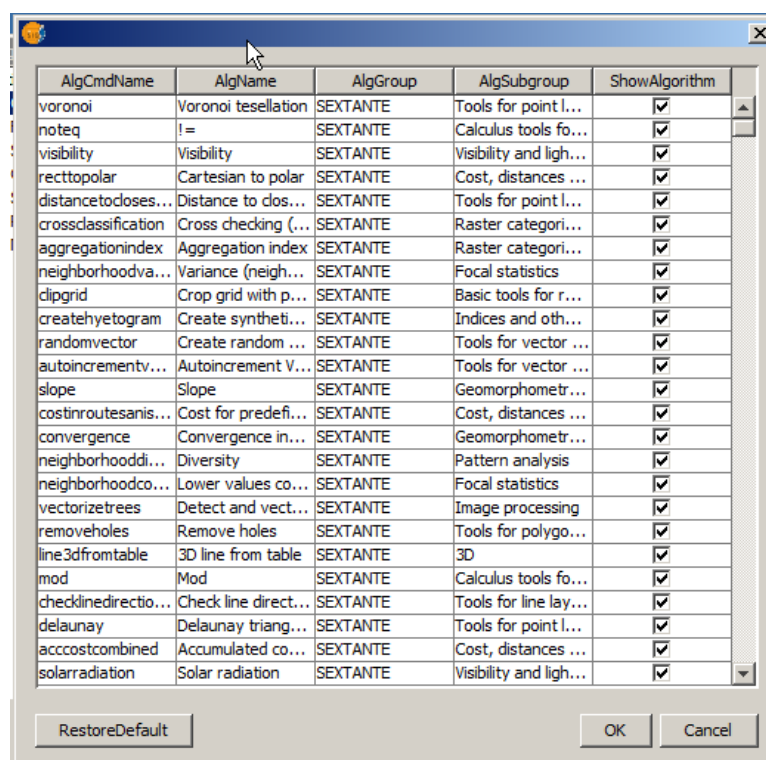
2.5.1 General

You will see two check boxes in the upper part of this group:

- **Modify output names:** if you check this option, output names will be modified to avoid characters such as brackets, blank spaces or stress marks.
- **Use internal names for outputs:** when this option is selected, SEXTANTE uses internal names to name output layers. This is useful if you plan to use the command-line interface to write scripts, since the names of the outputs can be known in advance (having a look at the help files, under the *Command-line usage* will inform you of those names). If this option is not selected, SEXTANTE will produce layers with names that depend on the current language or sometimes on the names of the input layers, which can cause you trouble if you plan to use those layers for your script.

Under that, you will find two elements to configure the list of algorithms in the toolbox. The first one can be used to remove the *Most recently used* group in the toolbox. Just uncheck the check box and you will not see that group.

The second one is a button that can be used to redefine how algorithms are organized in the toolbox. Click on the button to get to the following dialog.



In it, you can change the group and subgroup each algorithm belongs to, and also you can select whether to show or not a given algorithm. Type on the table cells to enter an alternative group or subgroup for an algorithm. Once you are done, click on the OK button. You can restore the default groupings by clicking on the "Restore default" button.

2.5.2 Folders

One folder can be defined:

- **Output folder:** when entering the filename for an output layer, if it does not include a valid path, it will be saved to this default output folder.

2.5.3 Model

The folder where models are stored has to be set in this field. This will be explained in detail in the following chapter. Once you have entered the path to the folder that you want to use, click on the button below to make SEXTANTE load all the models found there.

2.5.4 GRASS, SAGA and other additional algorithm providers

The set of algorithms of SEXTANTE can be extended by using additional algorithm providers that wrap algorithms from a third-party software. SEXTANTE acts as a front-end to those applications and can reuse their algorithms. Configuration of algorithm providers depends on the characteristics of the provider itself and the software being called from SEXTANTE. This is explained in detail in a separate chapter at the end of this manual.

2.6 Iterative execution of algorithms

SEXTANTE algorithms can be executed iteratively when they include some kind of vector input. In this case, for a layer containing n features, the algorithm is executed n times, each time taking an input layer that contains just one feature from the original layer.

This is useful for certain processes, like, for instance, cropping a raster layer using a vector layer with polygons. Given a polygon layer, the raster layer can be cropped in smaller ones, each of them covering the minimum extent needed to include each one of the input polygons. This will require executing the corresponding algorithm as many times as polygons are contained in the vector layer, which might be a long and tedious process. Instead, executing the algorithm iteratively will automate the task, solving the problem in just one single step.

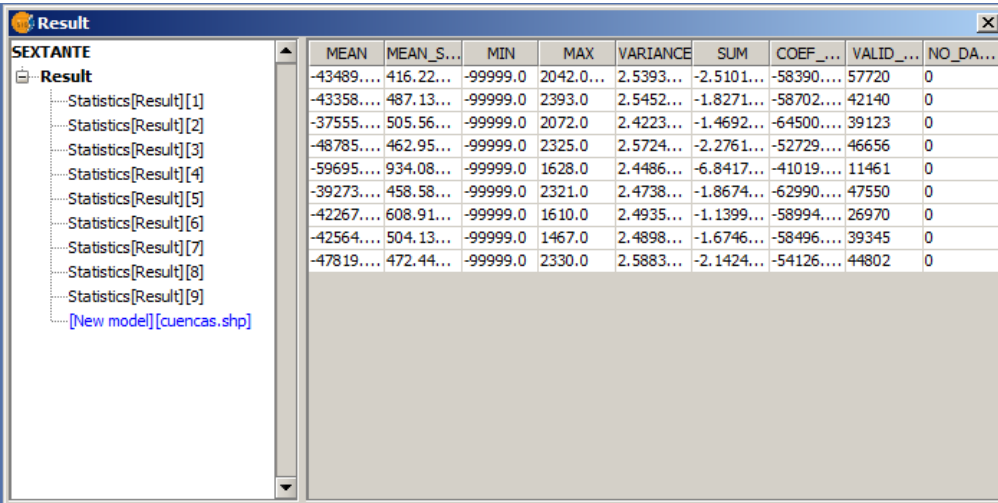
To execute an algorithm iteratively, right click on its name in the toolbox. You will see a new menu named *Execute iteratively [parameter name]*. There will be as many menus as vector layers the algorithm takes. You can only iterate over one of them, so you have to select the right one depending on what you want to do.

Clicking on the chosen menu, the usual algorithm dialog is shown. It is exactly like the dialog you would see if executing the algorithm the usual way. Just fill in the parameter values and click on *OK*. The progress dialog will inform you of the step that is currently being executed. Resulting layers will be added to the GIS GUI as usual.

When executing iteratively an algorithm that produces text output with numerical values (such as, for instance, statistics of a raster layer), it might be convenient to present those results in a different manner. Otherwise, combining a large number of such text outputs would be difficult and not practical. For this reason, when an algorithm is executed iteratively, its numerical outputs (in case the algorithm generates them) are presented in a table in the

results manager. Each row of the table represents an execution of the algorithm, while each column contains the values of one of the numerical variables being calculated.

The following picture shows an example of one of such tables.



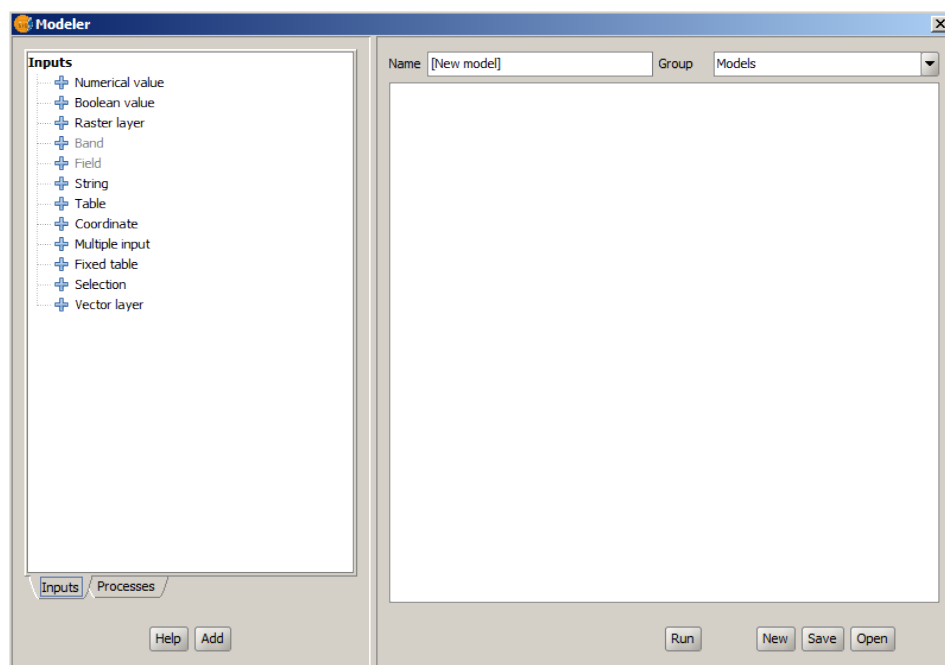
	MEAN	MEAN_S...	MIN	MAX	VARIANCE	SUM	COEF_...	VALID_...	NO_DA...
Statistics[Result][1]	-43489....	416.22...	-99999.0	2042.0...	2.5393...	-2.5101...	-58390....	57720	0
Statistics[Result][2]	-43358....	487.13...	-99999.0	2393.0	2.5452...	-1.8271...	-58702....	42140	0
Statistics[Result][3]	-37555....	505.56...	-99999.0	2072.0	2.4223...	-1.4692...	-64500....	39123	0
Statistics[Result][4]	-48785....	462.95...	-99999.0	2325.0	2.5724...	-2.2761...	-52729....	46656	0
Statistics[Result][5]	-59695....	934.08...	-99999.0	1628.0	2.4486...	-6.8417...	-41019....	11461	0
Statistics[Result][6]	-39273....	458.58...	-99999.0	2321.0	2.4738...	-1.8674...	-62990....	47550	0
Statistics[Result][7]	-42267....	608.91...	-99999.0	1610.0	2.4935...	-1.1399...	-58994....	26970	0
Statistics[Result][8]	-42564....	504.13...	-99999.0	1467.0	2.4898...	-1.6746...	-58496....	39345	0
Statistics[Result][9]	-47819....	472.44...	-99999.0	2330.0	2.5883...	-2.1424...	-54126....	44802	0

The SEXTANTE graphical modeler

3.1 Introduction

The *graphical modeler* allows to create complex models using a simple and easy-to-use interface. When working with a GIS, most analysis operations are not isolated, but part of a chain of operations instead. Using the graphical modeler, that chain of processes can be wrapped into a single process, so it is easier and more convenient to execute than a single process later on a different set on inputs. No matter how many steps and different algorithms it involves, a model is executed as a single algorithm, thus saving time and effort, specially for larger models.

The modeler has a working canvas where the structure of the model and the workflow it represents are shown. On the left part of the window, a panel with two tabs can be used to add new elements to the model.



Creating a model is a two-step process.

- *Definition of necessary inputs.* These inputs will be added to the parameters window, so the user can set their values when executing the model. The model itself is a SEXTANTE

algorithm, so the parameters window is generated automatically as it happens with all the algorithms included in the library.

- *Definition of the workflow.* Using the input data of the model, the workflow is defined adding algorithms and selecting how they use those inputs or the outputs generated by other algorithms already in the model

3.2 Definition of inputs

The first step to create a model is to define the inputs it needs. The following elements are found in the *Inputs* tabs on the left side of the modeler window:

- Band
- Raster layer
- Vector layer
- String
- Table field
- Coordinate (Point)
- Table
- Fixed table
- Multiple input
- Selection
- Numerical value
- Boolean value

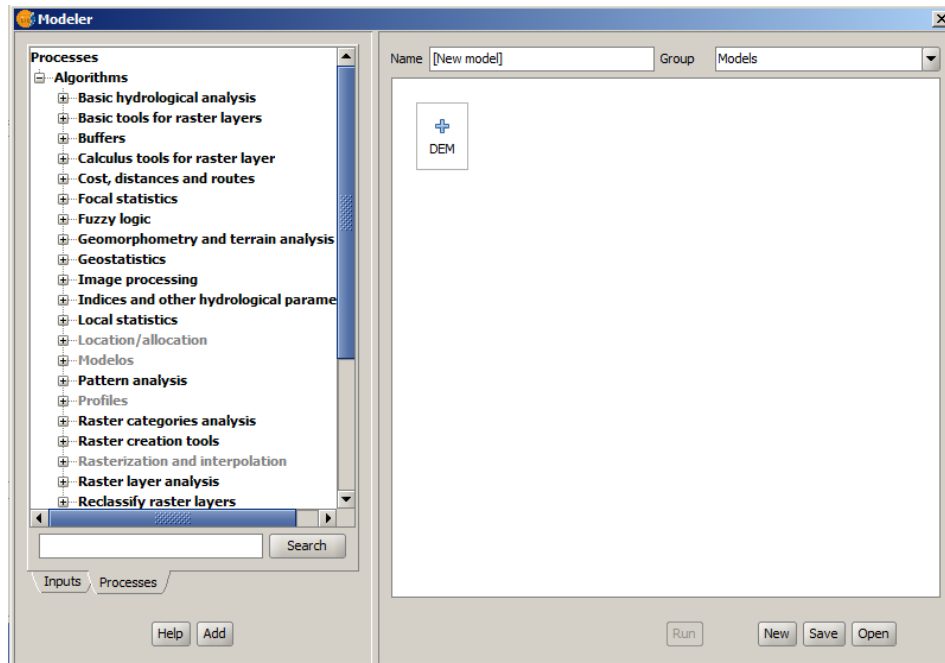
Double-clicking on any of them, a dialog is shown to define its characteristics. Depending on the parameter itself, the dialog will contain just one basic element (the description, which is what the user will see when executing the model) or more of them. For instance, when adding a numerical value, as can be seen in the next figure, apart from the description of the parameter is needed to set a default value, the type of numerical value and a range of valid values.

For each added input, a new element is added to the modeler canvas.

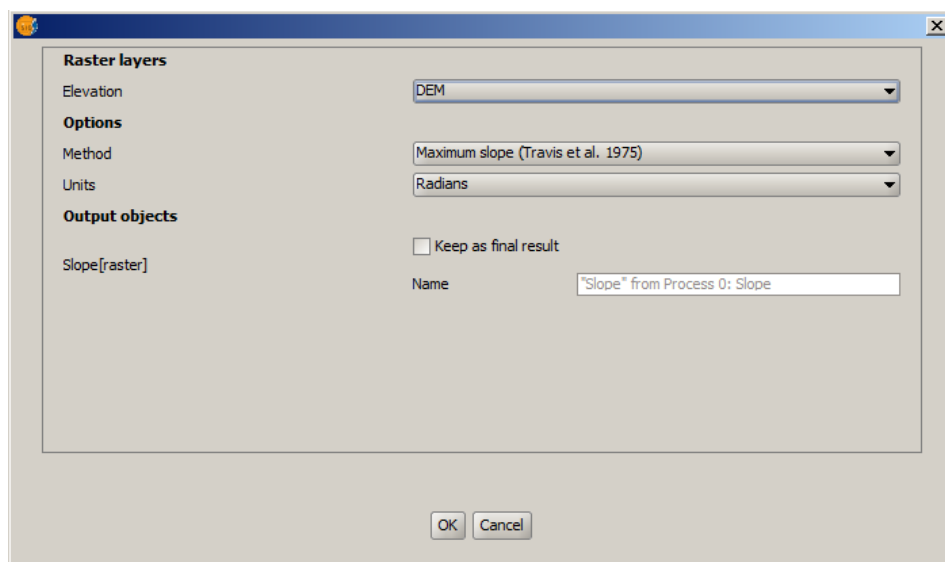


3.3 Definition of the workflow

Once the inputs have been defined, it is time to define the algorithms to apply on them. Algorithms can be found in the *Processes* tab, grouped much in the same way as they are in the toolbox.



To add a process, double-click on its name. An execution dialog will appear, with a content similar to the one found execution panel that SEXTANTE shows when executing the algorithm from the toolbox.



Some differences exist, however, the main one being the absence of a raster output tab, even if the selected algorithm generates raster layers as output.

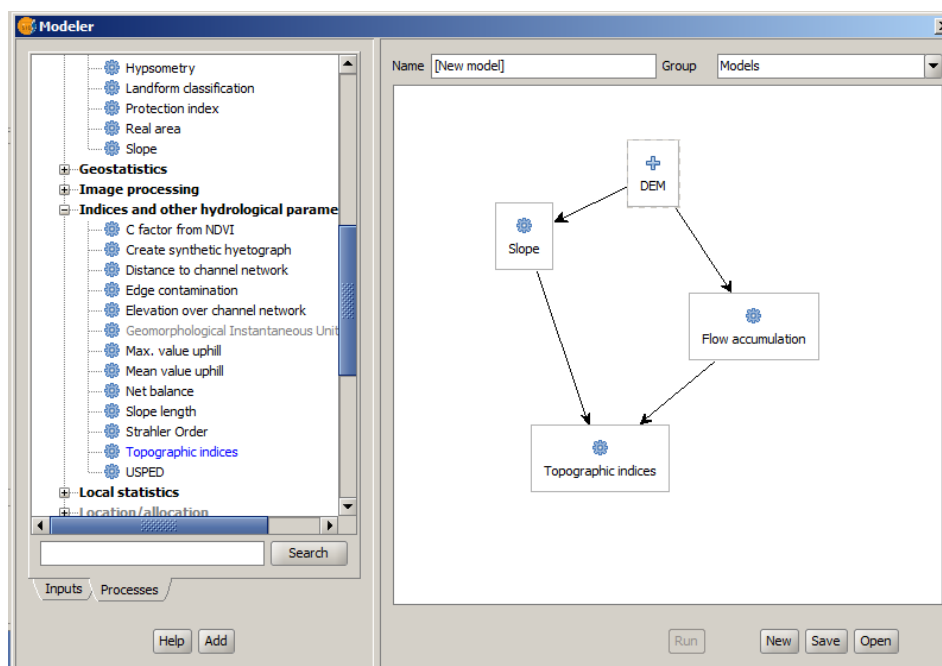
Instead of the textbox that was used to set the filepath for output layers and tables, a checkbox and a text box are found. If the layer generated by the algorithm is just a temporary result that will be used as the input of another algorithm and should not be kept as a final

result, the check box should be left unchecked. Checking it means that the result is a final one, and you have to supply also a valid description for the output, which will be the one the user will see when executing the model.

Selecting the value of each parameter is also a bit different, since there are important differences between the context of the modeler and the toolbox one. Let's see how to introduce the values for each type of parameter.

- Layers (raster and vector) and tables. They are selected from a list, but in this case the possible values are not the layers or tables currently loaded in the GIS, but the list of model input or the corresponding type, or other layers or tables generated by algorithms already added to the model.
- Numerical values. Literal values can be introduced directly on the textbox. This textbox is a list that can be used to select any of the numerical value input of the model. In this case, the parameter will take the value introduced by the user when executing the model.
- String. Like in the case of numerical values, literal strings can be typed, or an input string can be selected
- Points. Coordinates cannot be directly introduced. Use the list to select one of the coordinate inputs of the model
- Bands. The number of bands of the parent layer cannot be known at design-time, so it is not possible to show the list of available bands. Instead, a list with band numbers from 1 to 250, as well as the band parameters of the model, is shown. At run-time, SEXTANTE will check if the parent raster layer selected by the user has enough bands and the given band has therefore a valid value, and if not it will generate an error message.
- Table field. Like in the previous case, the fields of the parent table or layer cannot be known at design-time, since they depend of the selection of the user each time the model is executed. To set the value for this parameter, type the name of a field directly in the textbox, or use the list to select a table field input already added to the model. The validity of the selected field will be checked by SEXTANTE at run-time
- Selection. The list contains in this case not only the available option from the algorithm, but also the selection inputs already added to the current model

Once all the parameter have been assigned valid values, click on *OK* and the algorithm will be added to the canvas. It will be linked to all the other elements in the canvas, whether algorithms or inputs, which provide objects that are used as inputs for that algorithm.



3.4 Editing the model

Once the model has been designed, it can be executed clicking on the *Execute* button. The execution window will have a parameters tab automatically created based on the requirements of the model (the inputs added to it), just like it happens when a simple algorithm is executed. If any of the algorithms of the model generates raster layers, the *Raster output* tab will be added to the window.

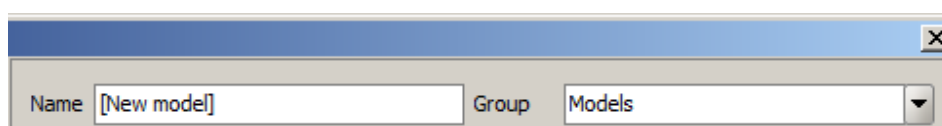
Elements can be dragged to a different position within the canvas, to change the way the module structure is displayed and make it more clear and intuitive. Links between elements are update automatically.

To change the parameters of any of the algorithms of a model, double-click on it to access its parameters window.

To delete an element, right-click on it and select *Delete*. Only those elements that do not have any other one depending on them can be deleted. If you try to delete an element that cannot be deleted, SEXTANTE will show the following warning message.

3.5 Saving and loading models

Models can be saved to be executed or edited at a later time. Use the *Save* button to save the current model and the *Open* model to open any model previously saved. Models are saved in an XML file with the .model extension.



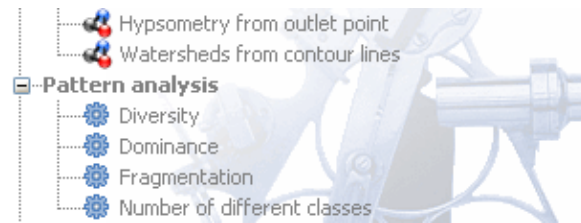
Models saved on the models folder will appear in the toolbox in a group that you can set using the boxes in the top of the modeler window. Type in the name of the model and then select a group from the drop-down list. The list contains all the names of the already existing

groups, and also an additional group named “Models”. If none of this group suits your needs, you can type a new name directly in that box, which is editable.

When the toolbox is invoked, SEXTANTE searches the models folder for files with .model extension and loads the models they contain. Since a model is itself a SEXTANTE algorithm, it can be added to the toolbox just like any other algorithm.

The models folder can be set from the SEXTANTE toolbox, clicking the configuration button and then introducing the path to the folder in the corresponding field. Go to the “Folders” tab to find it.

Models loaded from the models folder appear not only in the toolbox, but also in the algorithms tree in the *Processes* tab of the modeler window. That means that you can incorporate a model as a part of a bigger model, just as you add any other algorithm. However, models are shown with a different icon, to make it easy to recognize them.

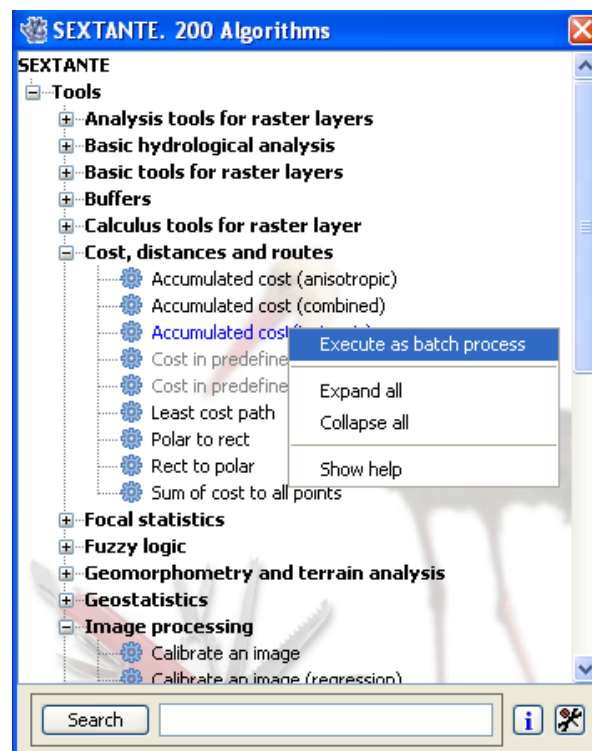


By default, the models folder is the same one as the folder where SEXTANTE help files are located. This folder contains a small set of example models, that you can use to better understand how the modeler works. Open them and study how they are constructed. You can also check their associated help files. As it has been said, models are themselves SEXTANTE algorithms, so they can have their own help files, and these can be edited as we have already seen in the previous chapter.

The SEXTANTE batch processing interface

4.1 Introducción

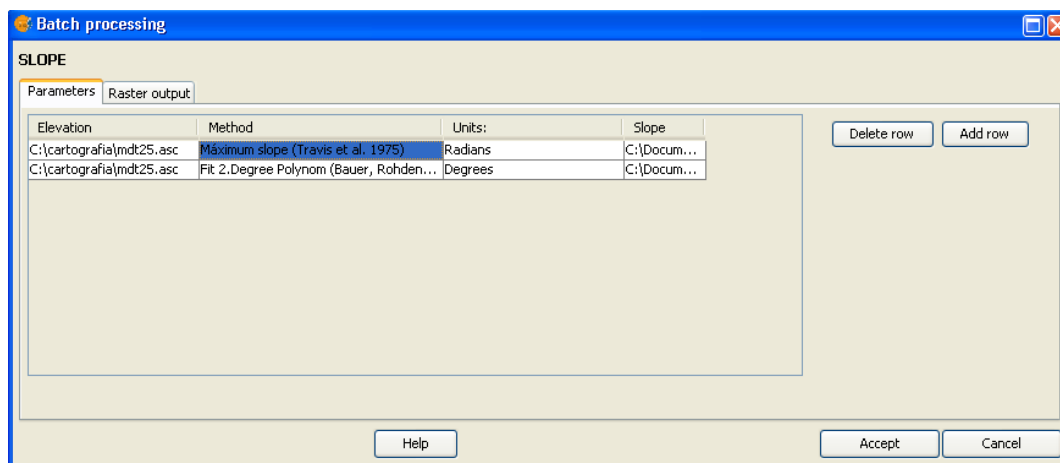
SEXTANTE algorithms (including models) can be executed as a batch process. That is, they can be executed using not a single set of inputs, but several of them, executing the algorithm as many times as needed. This is useful when processing large amounts of data, since it is not necessary to launch the algorithm many times from the toolbox.



4.2 The parameters table

Executing a batch process is similar to performing a single execution of an algorithm. Parameter values have to be defined, but in this case we need not just a single value for each

parameter, but a set of them instead, one for each time the algorithm has to be executed. Values are introduced using a table like the one shown next.



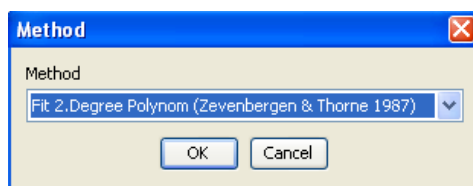
Each line of this table represents a single execution of the algorithm, and each cell contains the value of one of the parameters. It is similar to the parameters tab that you see when executing an algorithm from the toolbox, but with a different arrangement.

By default, the table contains just two rows. You can add or remove rows using the buttons on the right hand side of the window.

Once the size of the table has been set, it has to be filled with the desired values

4.3 Filling the parameters table

Whatever the type of parameter it represents, every cell has a text string as its associated value. Double-clicking on a cell, this string can be edited, directly typing the desired value. For most of the parameters, however, it is more convenient to use the button on the right hand side of the cell. Clicking on it, a dialog is shown to select the value of the parameter. The content of this dialog depends on the kind of parameter, and it features elements that make it easier to introduce the desired value. For example, for a selection parameter the list of all possible values is shown and the value can be chosen from them.



For all parameter cells, if the introduced value is correct, it will be shown in black. If the value is wrong (for instance, a numerical value out of the valid range or an option that does not exists for a selection parameter), the text will be shown in red.

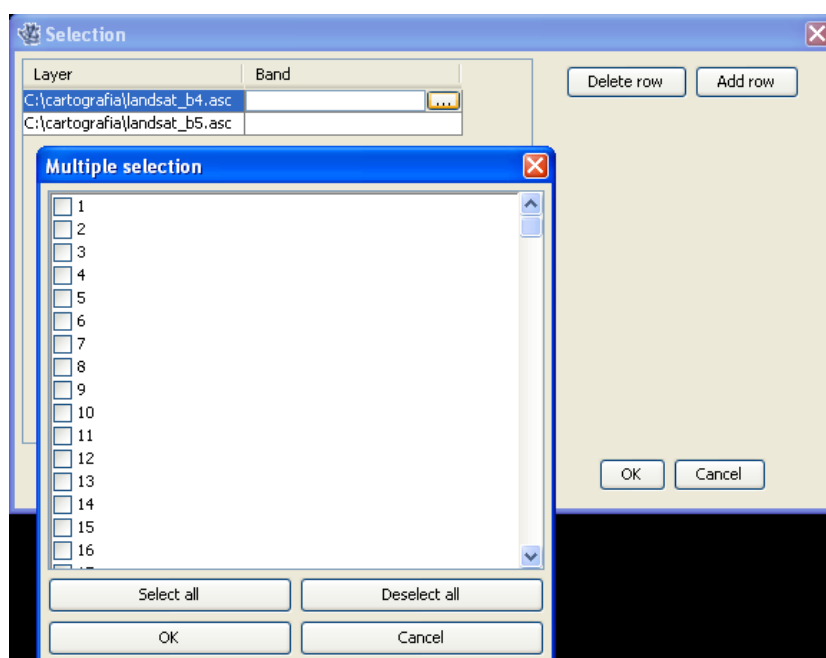
Parameters		Raster output	
Elevation	Method	Units	Slope
C:\cartografia\mdt25.asc	Máximum slope (Travis et al. 1975)	Radians	C:\Docum...
C:\cartografia\mdt25.asc	Fit 2.Degree Polynom (Bauer, Rohden...	DDegrees	C:\Docum...

The most importante different between executing an algorithm from the toolbox and executing it as part of a batch process is that input data objects are taken directly from files,

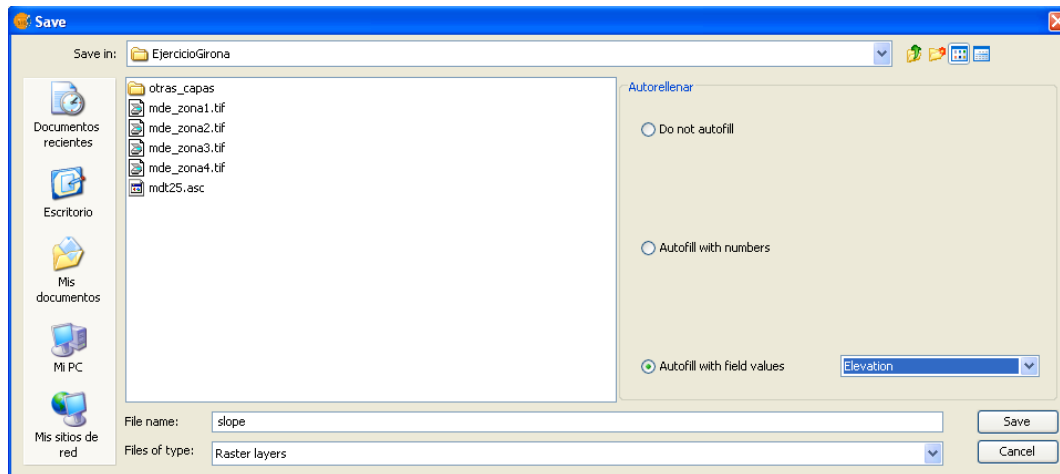
and not from the set of layers already opened in the GIS. For this reason, any algorithm can be executed as a batch process even if no data objects at all are opened and the algorithm cannot be called from the toolbox.

Filenames for input data objects are introduced directly typing or, more conveniently, clicking on the button on the right hand of the cell, which shows a typical file chooser dialog. Multiple files can be selected at once. If the input parameter represents a single data object and several files are selected, each one of them will be put in a separate row, adding new ones if needed. If it represents a multiple input, all the selected files will be added to a single cell, separated by commas.

If multiple bands are required, a more complex dialog is shown, which incorporates a table for selecting both layer files and bands. Click on the cells on the left side to select the file which contains the raster layer. Then click on the left side to select the bands you want to use from that layer. To know the number of bands in a layer it would be necessary to open it. However, SEXTANTE does not open the layer, and shows instead a list of bands from 1 to 250 to select from. If you select a band that does not exist in the selected layer, an error message will be shown at execution time.



Output data objects are always saved to a file and, unlike when executing an algorithm from the toolbox, saving to a temporary one is not permitted. You can type the name directly or use the file chooser dialog that appears when clicking on the accompanying button. This dialog differs slightly from the standard one, incorporating some additional fields for autocompletion.



If the default value (**Do not autocomplete**) is selected, SEXTANTE will just put the selected filename in the selected cell from the parameters table. If any of the other options is selected, all the cells below the selected one will be automatically filled based on a defined criteria. This way, it is much easier to fill the table, and the batch process can be defined with less effort.

Automatic filling can be done simply adding correlative numbers to the selected filepath, or appending the value of another field at the same row. This is particularly useful for naming output data object according to input ones.

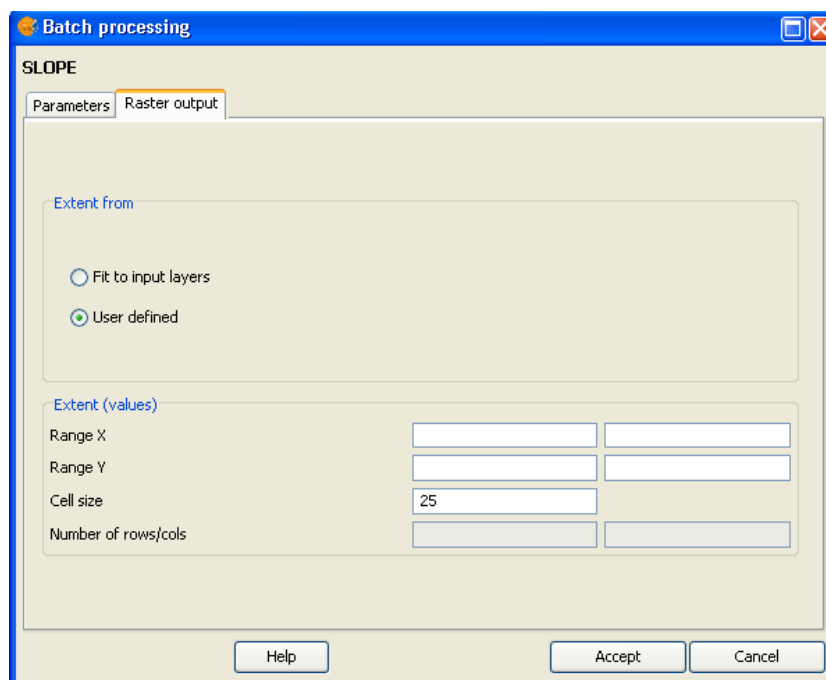
Slope
C:\Documents and Settings\usuario\Mis documentos\slope1.tif
C:\Documents and Settings\usuario\Mis documentos\slope2.tif
C:\Documents and Settings\usuario\Mis documentos\slope3.tif
C:\Documents and Settings\usuario\Mis documentos\slope4.tif

Cells can be selected just clicking and dragging. Selected cells can be copied and pasted in a different place of the parameters table, making it easy to fill it with repeated values.

4.4 Setting the output region

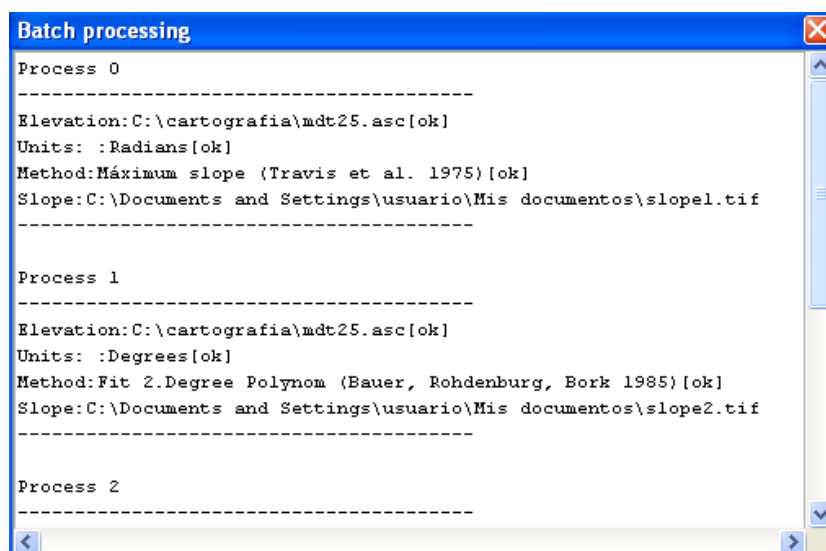
Just like when executing a single algorithm, when running a batch process you must define the extent of the region to be analyzed. The corresponding *Output region* tab is similar to the one found when running a single algorithm, but only contains two options: *fit to input layers* and *used-defined*.

The selection will be applied to all the single executions contained in the current batch process. If you want to use different output configurations, then you must define different batch processes.



4.5 Executing the batch process

To execute the batch process once you have introduced all the necessary values, just click on *OK*. SEXTANTE will show the progress of each executed algorithm, and at the end will show a dialog with information about the values used and the problems encountered during the execution of the whole process.



As it happened with the iterative execution of algorithms, when executing in a batch process an algorithm that produces text output with numerical values (such as, for instance, statistics of a raster layer), its numerical outputs are presented in a table in the results manager. Each row of the table represents an execution of the algorithm, while each column contains the values of one of the numerical variables being calculated.

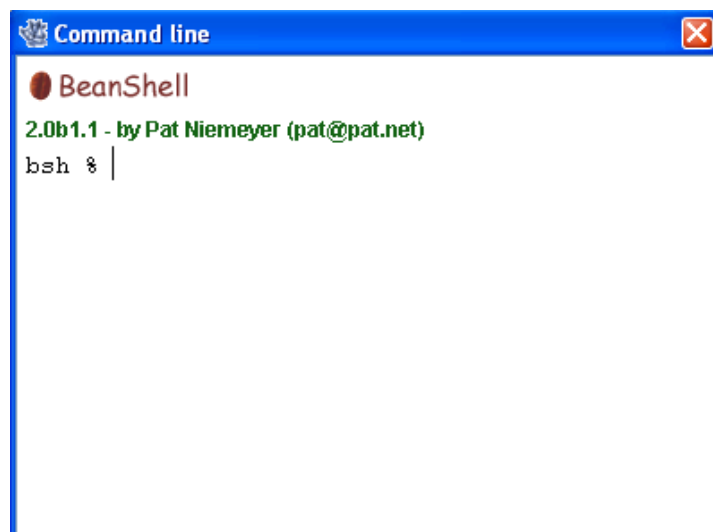
The SEXTANTE command-line interface

5.1 Introduction

The command-line interface allows advanced users to increase their productivity and perform complex operations that cannot be performed using any of the other elements of the SEXTANTE GUI. Models involving several algorithms can be defined using the command-line interface, and additional operations such as loops and conditional sentences can be added to create more flexible and powerful workflows.

5.2 The interface

Invoking the command-line interface will cause the following dialog to appear.



The SEXTANTE command-line interface is based on BeanShell. BeanShell is a Java source interpreter with object scripting language features, that meaning that it dynamically executes standard Java syntax and extends it with common scripting conveniences such as loose types, commands, and method closures like those in Perl and JavaScript.

A detailed description of BeanShell and its usage can be found at the BeanShell website¹. Refer to it if you want to learn more about generic BeanShell features. This chapter covers only those particular elements which are related to SEXTANTE geocalgorithms.

By using the extension mechanisms of BeanShell, SEXTANTE adds several new commands to it, so you can run geocalgorithms or get information about the geospatial data you are using, among other things.

Java users can create small scripts and programs combining standard elements of Java with SEXTANTE commands. However, those who are not familiar with Java can also use the command-line interface to execute single processes or small sets of them, simply calling the corresponding methods.

A detailed description of all SEXTANTE commands is given next.

5.2.1 Getting information about data

Algorithms need data to run. Layers and tables are identified using the name they have in the table of contents of the GIS (and which usually can be modified using GIS tool). To call a geocalgorithm you have to pass it an identifier which represents the data to use for an input.

The `data()` command prints a list of all data objects available to be used, along with the particular name of each one (i.e. the one you have to use to refer to it). Calling it you will get something like this:

```
RASTER LAYERS
-----
mdt25.asc

VECTOR LAYERS
-----
Contour lines

TABLES
-----
```

Be aware that some GIS allow you to have several layers with the same name. SEXTANTE will just take the first one which matches the specified identifier, so you should make sure you rename your data object so each one of them has a unique name.

To get more information about a particular data object, use the `describe(name_of_data_object)` command. Here are a few examples of the result you will get when using it to get more information about a vector layer, a raster layer and a table.

```
>describe("points")
Type: Vector layer - Point
Number of entities: 300
Table fields: | ID | X | Y | SAND | SILT | CLAY | SOILTYPE | EXTRAPOLAT |

>describe("dem25")
Type: Raster layer
X min: 262846.525725
X max: 277871.525725
Y min: 4454025.0
```

¹www.beanshell.org

```

Y max: 4464275.0
Cellsize X: 25.0
Cellsize Y: 0.0
Rows: 410
Cols: 601

>describe("spatialCorrelation")
Type: TableNumber of records: 156
Table fields: | Distance | I_Moran | c_Geary | Semivariance |

```

5.3 Getting information about algorithms

Once you know which data you have, it is time to know which algorithms are available and how to use them.

When you execute an algorithm using the toolbox, you use a parameters window with several fields, each one of them corresponding to a single parameter. When you use the command line interface, you must know which parameters are needed, so as to pass the right values to use to the method that runs that algorithm. Of course you do not have to memorize the requirements of all the algorithms, since SEXTANTE has a method to describe an algorithm in detail. But before we see that method, let's have a look at another one, the `algs()` method. It has no parameters, and it just prints a list of all the available algorithms. Here is a little part of that list as you will see it in your command-line shell.

```

bsh % algs();
acccost-----: Accumulated cost(isotropic)
acccostanisotropic-----: Accumulated cost (anisotropic)
acccostcombined-----: Accumulated cost (combined)
accflow-----: Flow accumulation
acv-----: Anisotropic coefficient of variation
addeventtheme-----: Points layer from table
aggregate-----: Aggregate
aggregationindex-----: Aggregation index
ahp-----: Analytical Hierarchy Process (AHP)
aspect-----: Aspect
buffer-----: Buffer

```

On the right you find the name of the algorithm in the current language, which is the same name that identifies the algorithm in the toolbox. However, this name is not constant, since it depends on the current language, and thus cannot be used to call the algorithm. Instead, a command-line is needed. On the left side of the list you will find the command-line name of each algorithm. This is the one you have to use to make a reference to the algorithm you want to use.

Now, let's see how to get a list of the parameters that an algorithms require and the outputs that it will generate. To do it, you can use the `describealg(name_of_the_algorithm)` method. Use the command-line name of the algorithm, not the full descriptive name.

For example, if we want to calculate a flow accumulation layer from a DEM, we will need to execute the corresponding module, which, according to the list shown using the `algs()` method, is identified as `accflow`. The following is a description of its inputs and outputs.

```
>describealg("accflow")
```

```
Usage: accflow(DEM[Raster Layer]
              WEIGHTS[Optional Raster Layer]
              METHOD[Selection]
              CONVERGENCE[Numerical Value]
              FLOWACC [output raster layer])
```

If an algorithm has a selection parameter, the value of that parameter should be entered using an integer value. To know the available options, you can use the `options` command, as shown in the following example:

In this case, the `slope` algorithm has two such parameters, the first one of them with 7 options, and the second one with 3. Notice that ordering is zero-based.

5.4 Running an algorithm

Now you know how to describe data and algorithms, so you have everything you need to run any algorithm. There is only one single command to execute algorithms: `runalg`. Its syntax is as follows:

```
> runalg{name_of_the_algorithm, param1, param2, ..., paramN}
```

The list of parameters to add depends on the algorithm you want to run, and is exactly the list that the `describealg` method gives you, in the same order as shown.

Depending on the type of parameter, values are introduced differently. The next one is a quick review of how to introduce values for each type of input parameter

- Raster Layer, Vector Layer or Table. Simply introduce the name that identifies the data object to use. If the input is optional and you do not want to use any data object, write “#”.
- Numerical value. Directly type the value to use or the name of a variable containing that value.
- Selection. Type the number that identifies the desired option, as shown by the `options` command
- String. Directly type the string to use or the name of a variable containing it.
- Boolean. Type whether “true” or “false” (including quotes)
- Multiple selection - data_type. Type the list of objects to use, separated by commas and enclosed between quotes.

For example, for the `maxvaluegrid` algorithm:

```
Usage: runalg("maxvaluegrid",
              INPUT[Multiple Input - Raster Layer]
              NODATA[Boolean],
              RESULT[Output raster layer])
```

The next line shows a valid usage example:

```
> runalg("maxvaluegrid", "lyr1, lyr2, lyr3", "false", "#")
```

Of course, lyr1, lyr2 and lyr3 must be valid layers already loaded into your GIS.

When the multiple input is comprised of raster bands, each element is represented by a pair of values (*layer, band*). For example, for the `cluster` algorithm

```
Usage: runalg( "cluster",
              INPUT[Multiple Input - Band],
              NUMCLASS[Numerical Value],
              RESULTLAYER[output raster layer],
              RESULTTABLE[output table],
              );
```

The next line shows a valid usage example:

```
> runalg("cluster", "lyr1, 1, lyr1, 2, lyr2, 2", 5, "#", "#")
```

The algorithm will use three bands, two of them from lyr1 (the first and the second ones of that layer) and one from lyr2 (its second band).

- [Table Field from XXX]. Write the name of the field to use. This parameter is case-sensitive.
- [Fixed Table]Tabla fija. Type the list of all table values separated by commas and enclosed between quotes. Values start on the upper row and go from left to right. Here is an example:

```
runalg("kernelfilter", "mdt25.asc", "-1, -1, -1, -1, 9, -1, -1, -1, -1", "#")
```

- [Point]. Write the pair of coordinates separated by commas and enclosed between quotes. For instance "220345, 4453616"

Input parameters such as strings or numerical values have default values. To use them, type “#” in the corresponding parameter entry instead of a value expression.

For output data objects, type the filepath to be used to save it, just as it is done from the toolbox. If you want to save the result to a temporary file, type “#”. Use “\$” to indicate that you want to overwrite an input layer. If the algorithm does not support overwriting, it will save the resulting layer to a temporary file. Use “!” to indicate that an output should not be created.

5.5 Adjusting the analysis region

If you execute from the command-line interface an algorithm that allows the user to select the characteristics of the analysis region, it will by default adjust it to the input layers, taking its extent (and cellsize in case of raster layers). You can toggle this behaviour using the `autoextent` command.

```
> autoextent("true"/"false")
```

If you want to define the analysis region manually or using a supporting layer, you have to use the `extent` command, which has three different variants.

```
Usage: extent(raster layer[string])
       extent(vector layer[string], cellsize[double])
       extent(x min[double], y min[double],
              x max[double], y max[double],
              cell size[double])
```

Type "autoextent" to use automatic extent fitting when possible

When this command is used, the autoextent functionality is automatically deactivated.

5.6 Managing layers from the command-line interface

You can perform some operation with layers from the command-line interface, like the following ones:

- Opening a layer. Use the `open(filepath_to_layer, name, view_name)` command. `View_name` is the name of the view where the layer should be added, while `name` is the name to give to the layer in that view.
- Closing a layer. Use the `close(layer_name)` command.
- Changing the no-data value of a raster layer. Use the `setnodata(layer_name, new_value)` command
- Changing the name of a layer. Use the `rename(layer_name, new_layer_name)` command

if you want to have the the names of layers and tables stored in a variable, so you can iterate them, you can use any of the following commands.

- `getRasterLayers()`.
- `getVectorLayers()`.
- `getTables()`.

All of these commands return an array of String values with the names of the corresponding data objects.

5.7 Creating scripts and running them from the toolbox

Scripts can be run using the `source(script_filename)` command. Simply put your commands in a text file and then you can execute them calling them with a single line.

You can define new commands (methods) and save them to a file, so running that file will load your commands and make them available for the current command-line session. For instance, here is an example method that calculates the slope of a DEM by all the available methods and then computes the mean value of all the slope layers and saves it to a temporary file.

```
slopemean(dem, meanslope){
  NUMBER_OF_METHODS = 7;
  multiple = "";
  for(i=0;i<NUMBER_OF_METHODS;i++){
```



```

    runalg("slope", dem, "#", "#", "#");
    rename("Slope", "Slope" + i);
    multiple = multiple + "Slope" + i;
    if (i < NUMBER_OF_METHODS - 1){
        multiple=multiple + ",";
    }
}
runalg("multigridmeanvalue", multiple, "#", meanslope);
for(i=0;i<NUMBER_OF_METHODS;i++){
    close("Slope"+i);
}
}

```

Assuming that this script is saved in a file named `/home/myuser/slopemean.bsh`, then it could be run just entering

```
source("/home/myuser/slopemean.bsh");
```

After doing that, the `slopemean` command would be available and could be called with a line like the following one:

```
slopemean("dem", "meanslope.tif");
```

`dem` being the name of the DEM layer that we want to analyze. The file will be saved to the default output folder.

Scripts can be made available from the toolbox as `geoalgorithms`, following these rules:

- Each script must contain just one method.
- The name of the method must be the same as the name of the script file.
- The file must have the “bsh” extension.

The above example meets these three requirements.

Since `SEXTANTE` needs some additional information to create the parameters window, additional lines must be added to provide that information. This should be added as Java comments before the method itself, and should have the name of the parameter (the name to show to the user), the equal sign (=) and the type of parameter. The following keywords can be used to describe the type of a given parameter: `raster`, `vector`, `table`, `multiple raster`, `multiple vector`, `boolean`, `number`, `string`, `output raster`, `output vector`, `output table`. Comments used to define parameters should appear in the same order as they appear in the method call.

For the above example, the following comment lines should be added:

```
//dem=raster
//meanslope=output raster
```

The last step is to define the scripts folder. Only script files from that folder will be loaded as algorithms. You will find a *Script* tab in the setting dialog, which is very similar to the *Models* one that you should already know how to use.

Scripts in the scripts folder are automatically executed when the command line is opened, so their corresponding method will be available without having to call them using the `source()`

command. Make sure that those files contain only method definitions; otherwise, the processes they contain will be executed as well each time you start a new command-line session (unless you really want that to happen...).

An easier way of creating scripts and adding them to the toolbox is to use the built-in script editor. In the toolbox, you will find a *Create new script* command. It will open the editor, where you can start writing your script. The save button will allow you to save it. The default folder opened in the dialog when you click the save button is the scripts folder, so if you save your script there, it will be automatically included in the toolbox, with no need to go to the settings dialog as explained above.

The SEXTANTE help folder contains several example scripts. Check them to better understand how this feature works.

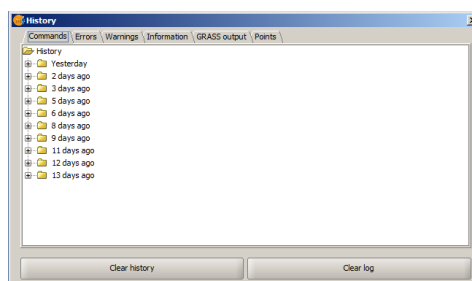
The SEXTANTE history manager

6.1 Introduction

Every time you execute a SEXTANTE algorithm, information about the process is stored in the SEXTANTE history manager. Along with the parameters used, the date and time of the execution are also saved.

This way, it is easy to track the and control all the work that has been developed using SEXTANTE, and easily reproduce it.

The SEXTANTE history manager is a set of registries grouped according to their date of execution, making it easier to find information about an algorithm executed at any particular moment.



Process information is kept as a command-line expression, even if the algorithm was launched from the toolbox. This makes it also useful for those learning how to use the command-line interface, since they can call an algorithm using the toolbox and then check the history manager to see how that same algorithm could be called from the command line.

Apart from browsing the entries in the registry, processes can be re-executed, simply double-clicking on the corresponding entry.

You can also right click on a process (the command-line sentence must start with “runalg”) and select *Open algorithm dialog*. This will show the dialog used to execute the algorithm, already filled with the parameter values corresponding to the selected command.

Configuring algorithm providers

7.1 Introduction

SEXTANTE can be extended using additional applications, calling them from within SEXTANTE. Currently, GRASS, SAGA and R are supported. This chapter will show you how to do it. Once you have configured the system, you will be able to execute external algorithms from any SEXTANTE component like the toolbox or the graphical modeler, just like you do with any other SEXTANTE gealgorithm.

Certain desktop GIS that incorporate SEXTANTE have third-party applications already preconfigured, so you do not have to configure any of them and their algorithms are available in the SEXTANTE toolbox since the first time you start the program. Using the preconfigured settings is always preferred, so you can (and, unless you are a experienced user, you should) skip this chapter if you are using one of those desktop GIS.

7.2 SAGA

SAGA binaries have to be installed in the SEXTANTE folder, in a subfolder named **saga**. The folder structure has to be like the following one:

```
|-[SEXTANTE_folder]
|  |-saga
|     |-description
|     |-dll
|     |-modules
```

The **description** folder contains the description of SAGA algorithms. It can be obtained from the SEXTANTE SVN repository. It is part of the **sextante_gui** project and it can be found under the **alg_descriptions/grass/descriptions**. All SAGA executables should be in the **saga** folder.

SAGA 2.0.7 version is supported. Other versions might be used, but certain algorithms might not work as expected.

7.3 GRASS

GRASS binaries have to be installed in the SEXTANTE folder, in a subfolder named "grass".

The folder structure has to be like the following one:

```
|-[SEXTANTE_folder]
|-grass
|-bin
|-bwidget
|-description
|- ...
|-tools
```

The `description` folder contains the description of GRASS algorithms. It can be obtained from the SEXTANTE SVN repository. It is part of the `sextante_gui` project and it can be found under `alg_descriptions/grass/descriptions`. All the other folders belong to the GRASS distribution.

SAGA 6.4.1 version is supported. Other versions might be used, but certain algorithms might not work as expected.

Windows users also need the `msys` interpreter, which should reside in the SEXTANTE folder, in a subfolder named `msys`.

The folder structure in this case has to be like the following one:

```
|-[SEXTANTE_folder]
|-grass
|-bin
|-bwidget
|-description
|- ... |-tools
|-msys
|-bin
|-etc
|-...
|-var
```

7.3.1 R

R binaries have to be installed in the SEXTANTE folder, in a subfolder named `r`.

The folder structure has to be like the following one:

```
|-[SEXTANTE_folder]
|-r
|-bin
|-doc
|- ...
|-src
```

The SEXTANTE–R integration has been tested with R 2.11.1.

Check section 7.6 for more info about how to create R scripts and use them from SEXTANTE once you have configured the R–SEXTANTE interface.

7.4 Some important notes

Although it is recommended to keep third party applications in their default location (under the SEXTANTE folder), you can change that by entering the settings dialog (click on the button in the lower-right part of the SEXTANTE toolbox) and navigating to the corresponding

settings group in its left-hand side. You will find *GRASS folder*, *R Folder* and *SAGA folder* textboxes where you can change the path to each one of the executables for these applications.

You can activate or de-activate algorithms from each one of these applications, by clicking on the corresponding check-box (you must activate them the first time, since they are deactivated by default, unless your system is already preconfigured). Note that, even if the folder you entered is not correct, when a group of algorithms is activated, you will see them in the toolbox and you will be able to call them. In other words, the descriptions of the algorithms are not a part of the third party software, but are incorporated into SEXTANTE. Once you try to execute the algorithm, you will see an error message in case SEXTANTE could not call the software needed to run the requested algorithm.

7.5 More about the GRASS interface

This section describes some additional configuration items for the SEXTANTE-GRASS interface. It also gives some additional information on the mechanism used by SEXTANTE to integrate GRASS modules, which should be useful for all users, but specially for those familiar with the GRASS command-line interface.

Open the settings dialog and select the *GRASS* menu page. Apart from the folders already explained, there are additional parameters that can be defined.

- The path to a GRASS mapset. The mapset doesn't have to contain any data at all, since data will be imported automatically each time you execute an algorithm. The GRASS interface is totally ignorant of projection settings. When data is processed, no reprojection is performed, and layers are assumed to be in the same projection as the mapset itself. So make sure that you only process data with matching spatial reference systems. Otherwise, accurate results cannot be guaranteed. If you do not have a GRASS mapset, you can ask SEXTANTE to create a temporary one for you. Select the corresponding checkbox and a new one will appear instead of the textbox for selecting the mapset folder. In order to create the temporary mapset, SEXTANTE needs to know whether your data will use geographic coordinates (lat/lon) or projected ones. Check the new box accordingly. The temporary mapset and all data in it will automatically be deleted when the processing is done.
- If you wish to process 3D vector data with GRASS, then you must activate the corresponding setting (3D input data will not automatically be recognized as such).
- You may also choose to import polygons as polylines instead.

7.5.1 Usage notes and limitations

GRASS is a system that consists of hundreds of independent, loosely coupled, programs designed to be run from the command line. There are some complexities in trying to wrap a graphical user interface (GUI) around such an architecture. It can never be done perfectly, but the GRASS-SEXTANTE interface goes to some lengths in order to ensure a smooth user experience.

There are several different versions of GRASS available. Currently, the GRASS-SEXTANTE interface has been tested and designed to run with GRASS 6.4. Other GRASS versions may or may not work.

If you notice anything wrong with a particular GRASS module, please post a message to the SEXTANTE users mailing list, notifying us of your concern. We will try to fix it for the next release.

The current version of the SEXTANTE-GRASS interface offers good support for most of the GRASS raster and vector processing modules. Most significantly, it does not support the imagery (i.*) and voxel (3D raster; r3.*) processing modules. Users who want access to the full power and flexibility of GRASS GIS are advised to install GRASS on an operating system with good POSIX compatibility (such as Linux or Mac OS X) and learn to use it from the command line.

Not all GRASS algorithms are available from SEXTANTE. Some of them are not compatible with the architecture of SEXTANTE and its algorithm-definition semantics, while others do not make much sense in the context of SEXTANTE (like, for instance, those used to digitize and create new vector layers). Unsuitable algorithms are automatically removed and will not appear in any SEXTANTE component.

Message output from GRASS modules

Many GRASS modules produce verbose and important output as part of their processing. This can be reviewed after a GRASS module has run, by opening the *GRASS output* page of the SEXTANTE History. This is always a good idea, especially when unexpected results occur.

Some modules do not output error messages in a standard way, so that errors can be detected and a message displayed by SEXTANTE. If a module produces an empty or no result, check the full GRASS messages transcript in the SEXTANT log browser.

Graphical interface

Those GRASS modules that can produce a multitude of optional outputs will be split up into "sibling" algorithm, one for each optional output. Siblings share the same name with the "parent" algorithm (the one with the full set of parameters) but have an additional specifier in "()".

Sometimes, a certain option is impossible or pointless to replicate in the GUI and will thus be skipped, leading to discrepancies with the official GRASS module documentation. A prime example is the "layer=" option which many GRASS vector modules employ to let the user switch between different attribute tables connected to the same "layer" (which is actually called a "map" in GRASS lingo).

Some modules upload data into existing or new attribute tables for an existing input vector dataset. Such modification will be lost after the GRASS command finishes. We have tried to encapsulate the most important ones using a postprocessing function which will export the new attribute table fields together with a copy of the original input dataset as a new vector layer. In this way, modules such as v.distance become fully functional. However, this is not a universal solutions and some modules that modify the attribute table structure of an existing input vector dataset are still likely to lose these changes.

Vector data exchange

The GRASS interface currently uses ESRI Shapefiles as a kind of lowest common denominator to exchange data between SEXTANTE and GRASS. Shapefiles have severe limitations, which may also be felt when processing vector data with the SEXTANTE-GRASS interface. These limitations do not exist in the native GRASS vector models but are caused by having to rely on the much simpler Shapefiles for data exchange.

The most obvious limitation is the fact that Shapefiles can only store one type of geometric primitive each (point, line or polygon). The output of GRASS modules that produce multi-type geometries will automatically split into separate files for the primitives.

In addition, since Shapefiles use DBase files for attribute data, all limitations associated with that file format also apply.

Output vector maps will have a “cat” column or (if that already exists) a “_cat” column, which are the internal primary keys used by GRASS to link vector objects with attribute table fields. Apart from being a waste of bytes in the output file, GRASS modules will fail to run on input vector maps that already have both “cat” and “_cat” field. So it is a good idea to delete them manually from the attribute table. Unfortunately, the current official version of GRASS does not yet offer a safe way of doing this automatically.

(Please also make sure to read the notes on topology below)

Raster data exchange

There are no severe limitations for raster data processing via the SEXTANTE–GRASS interface.

However, there is no simple support for setting the GRASS raster MASK yet. If you need one, then you must create a GRASS mapset externally and then create a mask in there. Then use SEXTANTE to connect to that mapset. The mask will now be active for all raster operations carried out through the SEXTANTE GRASS interface.

Topology

GRASS is one of the few GIS that insist on keeping a strict topological model for all vector data that goes through it. This ensures reliable operation and correct output, but means that topologically unclean data may be a challenge to process without first cleaning it (“garbage in, garbage out”).

One common source of problems are overlapping polygons in one input file. The latter are not allowed in the 2D topology model that GRASS uses. GRASS will employ an automated cleaning process on such data which will most likely result in some of the polygons being discarded.

Note that the GRASS vector model currently has no valid topological representation for arbitrary 3D polygons (as opposed to simple 3D triangles, so called “faces”, which make up meshes such as TINs). Getting such data past the (unfortunately) 2D topological cleaning mechanism of GRASS without having it “butchered” can be a challenge. In those cases where only the geometry information (not the attribute data) is of interest, setting the GRASS interface options to import polygons as polylines may provide a solution.

The GRASS region

GRASS GIS offers many ways of setting the computation region’s extent and resolution on-the-fly. Doing this is only mandatory for modules with raster output (except raster import modules: `r.in.*`). But may also be important for some others (such as `v.voronoi`), whose result depends on the extent of the region, nonetheless. So the region settings are always available on the *Region* tab of each GRASS module’s GUI.

7.5.2 Windows notes

Due to its design, GRASS does not run as smoothly on Windows as it does on other operating systems, since the latter lacks some POSIX features for inter-process communication. For the user, the most significant effect of this is that SEXTANTE cannot display an accurate progress bar for GRASS commands running on Windows.

There is also no support for mapset locking on the Windows platform. So the user must take care not to use a mapset for processing which might be in use by another person at the same time.

7.5.3 Notes on specific modules

These are some usage hints for some interesting GRASS modules, which may not be obvious to GRASS novices. They also serve to illustrate common principles of GRASS usage via the SEXTANTE-GRASS interface.

r.colors(.stddev)

GRASS provides some beautiful, automatically adjusted color schemes for raster data. You can use “r.colors” to pick a scheme, but the new color scheme can only be applied to the result if the GIS that you run SEXTANTE under can handle external color map definitions in the format which the SEXTANTE-GRASS interface uses. At the moment, this is only true for gvSIG.

Note also that the result will be returned as a new layer, as the SEXTANTE-GRASS interface cannot directly manipulate the input layer.

r.in/out.gdal

Raster data in a variety of formats can be imported and exporting using r.in.gdal and r.out.gdal, respectively. These modules use the geodata drivers provided by the GDAL project¹. See the project’s web page for details about the level of support for the different formats.

r.mapcalculator

This is a GRASS script that wraps the powerful r.mapcalc tool, which is a commandline-only tool for raster map algebra in GRASS. If you want to get an idea of all its capabilities, find the HTML manual page for r.mapcalc in your local GRASS installation or on the web.

How to use r.mapcalculator: Specify up to six input layers to be used and then reference them in the “formula=” field. You can A,B,C etc . or amap,bmap,cmap etc. Don’t worry about putting in quotation marks (“). That will be done automatically. Here is an example of an expression that shows how to use the null() function and the if() conditional function: “if(A=>500,A,null())”. This will filter out all cells of a DEM (input as map A) that lie below 500 m.

r.null

You can very easily set a (range of) cell value(s) to “no data” (NULL) using this module. Note that the result will be returned as a new layer, as the SEXTANTE-GRASS interface cannot directly manipulate the input layer.

v.in/out.ogr

You can import and export several vector data formats using the v.in.ogr and v.out.ogr commands. The OGR drivers cater for a number of different vector data sources, so the interface semantics have been built around “dsn=” (data source) and “layer=” (layer within a data source) specifiers. The SEXTANTE-GRASS interface will allow you to simply select a file using the file selector behind the “dsn=” parameter.

¹<http://www.gdal.org>

For exporting data with `v.out.ogr`, make sure to select the right data format. If you skip the extension, the right one will automatically be added to the output file. For most formats, you can simply enter a path and file name into the “olayer” option field. Please consult the GDAL/OGR documentation for individual format details (e.g. set the “lsco” option value to “format=mif” if you want to create MapInfo ASCII vector output).

OGR is a subproject of GDAL, so details about the different formats can be found on the same project page. As with GDAL, the drivers supported will depend on your local version of the GDAL library.

Note that due to the use of Shapefiles for data exchange, multiple-geometry-type formats (such as MapInfo) are supported, but they will be split into single-geometry files after import.

v.surf.bspline

GRASS has some very flexible modules for spline curves based interpolation. The tricky part of `v.surf.bspline` is that you have to set the “layer” option to 0 if you want to interpolate the the Z coordinates of the input points directly. If you want to interpolate based on an attribute table field, set “layer=1” and then enter the name of the field as “column”.

v.surf.idw

This is a very capable, but also complex spline-based interpolation module. Getting high-quality output requires some knowledge about the many different parameters. Note that, due to the SEXTANTE interface semantics, at least one raster layer must be present in your project before the module becomes available.

v.to.3d

This module offers a convenient way to cast 2D vector data to 3D (or the other way around). The interface was tweaked a little so that it can run under SEXTANTE. Enter a constant height value into “height”, or leave height at the default setting and additionally enter an attribute field name into “column”. But “height” must always be set (at least to some dummy value).

Make sure to check the 3D data processing setting in the GRASS settings!

7.5.4 Technical details

If you are a GRASS user, it might be useful for you to know how SEXTANTE calls GRASS algorithms and communicates with the GRASS interface. This can be summarized in the following steps:

- Importing data. The layers that have been selected as input are imported into the (temporary or existing) GRASS mapset. Not every layer that you can open in your GIS can be used to execute a GRASS algorithm. It must be a file-based layer and it must have a format compatible with the capabilities of the GRASS modules that import external data. Refer to the GRASS help files for further information.
- Processing. The selected GRASS algorithm is executed.
- Exporting results. Resulting data are exported to the filenames selected by the user in the input parameters dialog. Only layers (whether raster or vector) are exported. Also, take into account that exporting data also has its limitations, and not all the information generated might be available. For instance, topological information will be lost, since data are exported to Shapefile format, which cannot store it. for further details, check the help file associated with GRASS export modules.

All this steps are stored in a batch file that is executed using the `GRASS_BATCH_JOB` variable. When `SEXTANTE` invokes `GRASS`, the commands in the batch file are executed and `GRASS` closes up automatically after that.

7.6 Creating R scripts

R integration in `SEXTANTE` is different from that of `GRASS` and `SAGA` in that there is not a predefined set of algorithms you can run (except for a few examples). Instead, you should write your scripts and call R commands, much like you would do from R. This chapter shows you the syntax to use to call those R commands from `SEXTANTE` and how to use `SEXTANTE` objects (layers, tables) in them.

To add a new algorithm that calls an R function (or a more complex R script that you have developed and you would like to have available from `SEXTANTE`), you have to create a script file that tells `SEXTANTE` how to perform that operation and the corresponding R commands to do so.

Script files have the extension `rsx` and creating them is pretty easy if you just have a basic knowledge of R syntax and R scripting. They should be stored in the R scripts folder. You can set this folder in the R settings window (available from the `SEXTANTE` settings dialog), just like you do with the folder for regular `SEXTANTE` scripts. You will also find a *Load scripts* button to load scripts in that folder. Check the command-line interface chapter to know more about how this works.

Let's have a look at a very simple file script file, which calls the R method `spsample` to create a random grid within the boundary of the polygons in a given polygon layer. This method belongs to the `maptools` package. Since almost all the algorithms that you might like to incorporate into `SEXTANTE` will use or generate spatial data, knowledge of spatial packages like `maptools` and, specially, `sp`, is mandatory.

```
//polyg=vector
//numpoints=number
//output=output vector
//sp=group
pts=spsample(polyg,numpoints,type="random")
output=SpatialPointsDataFrame(pts, as.data.frame(pts))
```

The first lines, which start with a java comment sign (`//`), tell `SEXTANTE` the input of the algorithm described in the file and the outputs that it will generate. They all have the following syntax: `[name of the input/output parameter]=[type of parameter]`

Supported types for input parameters are

- **vector**: a vector layer.
- **raster**: a raster layer.
- **table**: a raster layer.
- **string**: a string.
- **number**: a numerical value.
- **field**: a field selected from a vector layer or table. The name of the parent vector layer or table parameter has to be written following the **field** command.

- `multiple raster` or `multiple vector`: a set of several raster or vector layers.

Supported types for output parameters are

- `output vector`: a vector layer.
- `output raster`: a vector layer.
- `output table`: a vector layer.

you can also use the `group` tag to define the group in the toolbox where this algorithm should be shown. Usually, this should match the package name, for instance `//maptools=group`

When you declare an input parameter, SEXTANTE uses that information for two things: creating the user interface to ask the user for the value of that parameter and creating a corresponding R variable that can be later used as input for R commands

In the above example, we are declaring an input of type `vector polygon` named `polyg`. When executing the algorithm, SEXTANTE will open in R the layer selected by the user and store it in a variable also named `polyg`. So the name of a parameter is also the name of the variable that we can use in R for accessing the value of that parameter (thus, you should avoid using reserved R words as parameter names).

Spatial elements such as vector and raster layers are read using the `readOGR()` and `readGDAL()` commands (you do not have to worry about adding those commands to your description file, SEXTANTE will do it) and stored as `Spatial*DataFrame` objects. Fields are stored as numbers, representing the 1-based index of the selected field.

Knowing that, we can now understand the first line of our example script (the first line not starting with a java comment).

```
pts=spsample(polyg,numpoints,type="random")
```

The variable `polygon` already contains a `SpatialPolygonsDataFrame` object, so it can be used to call the `spsample` method, just like the `numpoints` one, which indicates the number of points to add to the created sample grid.

Since we have declared an output of type vector named `out`, we have to create a variable named `out` and store a `Spatial*DataFrame` object in it (in this case, a `SpatialPointsDataFrame`). You can use any name for your intermediate variables. Just make sure that the variable storing your final result has the same name that you used to declare it, and contains a suitable value.

In this case, the result obtained from the `spsample` method is not itself a `SpatialPointsDataFrame` object, but an object of class `ppp`, so we have to convert it explicitly.

If your algorithm does not generate any layer, but a text result in the console instead, you have to tell SEXTANTE that you want the console to be shown once the execution is finished. To do so, just start the command lines that produce the results you want to print with the `>` sign. the output of all other lines will not be shown. For instance, here is the description file of an algorithm that performs a normality test on a given field (column) of the attributes of a vector layer:

```
//layer=vector
//field=field layer
//nortest=group
library(nortest)
>lillie.test(layer[[field]])
```

The output of the last line is printed, but the output of the first is not (and neither are the outputs from other command lines added automatically by SEXTANTE).

If your algorithm creates any kind of graphics (using the `plot()` method), add the following line:

```
//showplots
```

This will cause SEXTANTE to redirect all R graphical outputs to a temporary file, which will be later opened once R execution has finished

Both graphics and console results will be shown in the SEXTANTE results manager.

For all kinds of analysis, SEXTANTE will ask the user to enter a bounding box and an output cellsize to be used by those algorithms that need them (the default option is to take those values from input layers if possible, so there is no need for the user to explicitly set them). These are available also for R algorithms. The bounding box is stored in a variable named `boundingBox` and is of class `matrix`, as returned by the `bbox()` method. The cellsize is stored in a variable named `cellsize`.

For more information, please check the script files provided with SEXTANTE. Most of them are rather simple and will greatly help you understand how to create your own ones.

A note about libraries: `rgdal` and `maptools` libraries are loaded by default so you do not have to add the corresponding `library()` commands. However, other additional libraries that you might need have to be explicitly loaded. Just add the necessary commands at the beginning of your script. You also have to make sure that the corresponding packages are installed in the R distribution used by SEXTANTE (the one under the SEXTANTE folder).