

Using Sphinx for generating course content

(Written by Martin Vogelaar, Kapteyn Astronomical Institute, Groningen, The Netherlands)

Introduction

In June 2013 we started an experiment with a document generator to structure and enhance our existing course material. The candidate course is a programming course and the programming language that we use is Python. Therefore our attention was immediately drawn to Sphinx. [Sphinx \(sphinx-doc.org/\)](http://sphinx-doc.org/) is written by Georg Brandl and licensed under the BSD license. Other programming languages are also supported, but we wanted to test whether the tool is also usable for courses that are not related to a programming language but are rich with mathematical formulas. We concluded that Sphinx gives also added value to this category of courses.

Starting point

In the last decade, many teachers and lectures migrated their written course material to the Web. Advantages are the availability of the material and the relative low burden of maintenance. Also the use of HTML is straightforward and formulas could be included as images so material for science and engineering was not restricted to text and could be written and published by teachers themselves. But in the meantime, many new techniques were introduced to enhance these web courses, but they are often missed or not applied because of the extra work involved.

Our Python programming course was an ideal starting point. The course material was entirely in HTML. Spelling mistakes could be corrected very easily and other small changes could be applied immediately, all improving the documents while they were in use. Formulas were build with MimeTeX. For writers who are familiar with LaTeX, this seemed a natural way to build formulas for the Web. But, compared to state of the art documentation on the Web, we saw many options for improvement. The most important were:

1. The structure of the documents and the relation between documents was not optimal.
2. The quality of the formulas was not very good and scaling them does worsen the quality.
3. The formulas could not be numbered and not be referenced.
4. A search facility was not available.
5. We did not have an index file.
6. There was not a straightforward way to display highlighted program code.
7. There was no way to keep a plot and the code that generates it, together.
8. Creating a PDF version was cumbersome.

HOWTO

(Written by Martina Tsvetanova, ASTRON, now at University of Twente, The Netherlands)

Our howto is written for the maintainers of a programming course at the Kapteyn Institute for which we use Sphinx to generate course content. To facilitate the course makers/maintainers, the following questions point to the locations of the corresponding answers in the text.

Basic setup

1. *What should I know before starting?*
2. *What to be prepared for before building documentation?*
3. *How to install Sphinx?*
4. *How to create documentation?*
5. *What is the documentation structure after creation?*
6. *How to add information in the documentation?*

The building process

7. *How to build?*
8. *How to add new documents to the source directory?*
9. *What kind of output can be generated except HTML? (How to create a PDF version?)*

SPHINX syntax (used in PROGNUM)

10. *How to add comment?*

11. *How to create a heading?*
12. *How to exclude section heading from table of contents?*
13. *How to create simple paragraph?*
14. *How to create coloured boxes?*
15. *How to write todo notes?*
16. *How create literal blocks and decorate inline text?*
17. *How to add image or figure?*
18. *How to add lists?*
19. *How to add tables?*
20. *How to control the line break?*
21. *How to determine to what depth the section headings appear in the table of contents?*
22. *How are documents related to each other?*

Configuration

23. *What is important to know for the configuration file `conf.py`?*
24. *How to specify in the configuration file where the custom extensions are?*
25. *Could a user create additional directories to reference files from?*
26. *How to specify the MathJax path in `conf.py`?*
27. *Where are extensions specified in `conf.py`?*
28. *Where can I find the HTML output configuration in `conf.py`?*
29. *How to hide the source of the page through `conf.py`?*
30. *Where can I find the LaTeX output configuration in `conf.py`?*
31. *Are there other output configuration options in `conf.py`?*
32. *What do I need to know about the style sheets?*
33. *What HTML style themes are available?*

Style

34. *How to move the sidebar from left to right under the default theme?*
35. *How to change the sidebar background color under the default theme?*
36. *How to change the text background color under the default theme?*
37. *How to create custom method for coloring text under the default theme?*

Mathematics and plots

38. *How to display and reference mathematics?*
39. *What are the options for mathematics display?*
40. *How to generate plots directly from Python code?*
41. *What do I need to know when using the `plot_directive`?*

Linking

42. *How to link to external page?*
43. *How to link to special place of a page?*
44. *How to link to other rst document (even) outside the toctree?*
45. *How to link to downloadable document?*
46. *How to link to other Sphinx documentation? (Intersphinx linking)*
47. *How to link to Python modules, classes, functions, methods, attributes and statements from other Sphinx documentations? (Intersphinx linking)*
48. *How to create entry in the index table?*
49. *How to create a Python code block?*
50. *How to number the lines of a Python code block?*

Customization

51. *How to start with custom extensions?*
52. *How to use the custom `plot_directive` extension?*
53. *How to use the custom question directive extension?*

Other examples from PROGNUM

54. *How to use directly HTML code?*
55. *How to make links in Sphinx to open in new tab?*
56. *How to create the question container used in PROGNUM?*

- 57. *How to create plots, similar to the PROGNUM examples?*
- 58. *How to create plots, including linking?*

Frequent errors

There are several frequently repeating errors, which can become slightly hard to discover sometimes. Please check these in *PART IV: Frequent errors*.

Reading carefully the error messages would help to discover the error easier.

Part I

1. Introduction

Sphinx is powerful documentation generator. It creates Web pages, requiring almost no effort from the user. The generator is very convenient for programmers who have to display a lot of source code, automatically generated plots, or who need a good referencing system for their Python modules, objects and equations.

On the other hand, Sphinx can be used for ordinary web pages and is really user-friendly, all input from the user is plain text, without too many syntax rules.

This document aims to help the user to start a documentation project with Sphinx and explain basic rules for using the generator, as well as to give advise on particular procedures while designing the desired documentation. For more extended help and tutorials, please visit the following link <http://sphinxsearch.com/docs/current.html>.

2. Information before documentation creation

Before starting, the user should decide what is going to be the directory for the documentation. A lot of files will be constantly generated when updating the documentation, that is why the client has to choose a place with enough free space. He or she has to also decide what the name, author and release of the documentation should be. These titles will appear in several places of the web page so it is nice to prepare them initially.

3. Creating a documentation

To install Sphinx, use the instructions <http://sphinx-doc.org/latest/install.html>. It is required to at least have Python 2.6.

After installation the user can start with the documentation with:

```
$ sphinx-quickstart
```

The documentation will start to generate itself. Several questions will be asked. The user has to decide and answer.

Example can be seen below:

```

1 Welcome to the Sphinx 1.1.3 quickstart utility.
2
3 Please enter values for the following settings (just press Enter to
4 accept a default value, if one is given in brackets).
5
6 Enter the root path for documentation.
7 > Root path for the documentation [.]: tutorialdoc
8
9 You have two options for placing the build directory for Sphinx output.
10 Either, you use a directory "_build" within the root path, or you separate
11 "source" and "build" directories within the root path.
12 > Separate source and build directories (y/N) [n]: y
13
14 Inside the root directory, two more directories will be created; "_templates"
15 for custom HTML templates and "_static" for custom style sheets and other static
16 files. You can enter another prefix (such as ".") to replace the underscore.
17 > Name prefix for templates and static dir [_]:
18
19 The project name will occur in several places in the built documentation.
20 > Project name: tutorial documentation
21 > Author name(s): user
22
23 Sphinx has the notion of a "version" and a "release" for the
24 software. Each version can have multiple releases. For example, for
25 Python the version is something like 2.5 or 3.0, while the release is
26 something like 2.5.1 or 3.0a1. If you don't need this dual structure,
27 just set both to the same value.
28 > Project version: 1.0
29 > Project release [1.0]: 2013
30
31 The file name suffix for source files. Commonly, this is either ".txt"
32 or ".rst". Only files with this suffix are considered documents.
33 > Source file suffix [.rst]:
34
35 One document is special in that it is considered the top node of the
36 "contents tree", that is, it is the root of the hierarchical structure
37 of the documents. Normally, this is "index", but if your "index"
38 document is a custom template, you can also set this to another file name
39 > Name of your master document (without suffix) [index]:
40
41 Sphinx can also add configuration for epub output:
42 > Do you want to use the epub builder (y/N) [n]: y
43
44 Please indicate if you want to use one of the following Sphinx extensions:
45 > autodoc: automatically insert docstrings from modules (y/N) [n]: y
46 > doctest: automatically test code snippets in doctest blocks (y/N) [n]: y
47 > intersphinx: link between Sphinx documentation of different projects (y/N) [n]: y
48 > todo: write "todo" entries that can be shown or hidden on build (y/N) [n]: y
49 > coverage: checks for documentation coverage (y/N) [n]: y
50 > pngmath: include math, rendered as PNG images (y/N) [n]: n
51 > mathjax: include math, rendered in the browser by MathJax (y/N) [n]: y
52 > ifconfig: conditional inclusion of content based on config values (y/N) [n]: y
53 > viewcode: include links to the source code of documented Python objects (y/N) [n]: y
54
55 Makefile and a Windows command file can be generated for you so that you
56 only have to run e.g. `make html` instead of invoking sphinx-build
57 directly.
58 > Create Makefile? (Y/n) [y]: y
59 > Create Windows command file? (Y/n) [y]: y
60
61 Creating file tutorialdoc/source/conf.py.
62 Creating file tutorialdoc/source/index.rst.
63 Creating file tutorialdoc/Makefile.
64 Creating file tutorialdoc/make.bat.
65
66 Finished: An initial directory structure has been created.
67
68 You should now populate your master file tutorialdoc/source/index.rst and create other documentation
69 source files. Use the Makefile to build the docs, like so:
70     make builder
71 where "builder" is one of the supported builders, e.g. HTML, LaTeX or linkcheck.

```

Warning: **pngmath** and **MathJax** cannot be used at the same time for displaying mathematics. While pngmath displays an equation as an image, MathJax is using scalable vector graphics (quality remains the same after zooming).

Also, separation of the build and source folders is recommended, it allows` better insight in the documentation structure. Most of the other functionalities are always nice-to have, so answering with yes is never a mistake.

The user needs to be sure what is expected from the Python documentation before building. Otherwise undesirable work might be needed later to fix the configuration file generated (**conf.py**).

4. Exploring newly created documentation and building

After creating the Sphinx documentation, it is recommended for the user to get acquainted with it. One should visit the directory where it is hosted. At first there are 2 other directories inside- **source** and **build** (if separated in the creation process) and 2 files- **Makefile** and **make.bat**. These are the files that should be never deleted, otherwise the documentation created would not be able to generate the HTML files for the Web pages This is also the working directory where the user has to type the command for building the HTML files:

```
$ sphinx-build -b html sourcedir builddir
```

Or only the following:

```
$ make html
```

Of course, before creating any output, there should be some input. The **source** directory should be visited first. Initially the **build** one is empty. When opened, it contains the master file- index.rst (or other name and **txt** extension if specified during the generation). Other present file is the **conf.py** file, which is responsible for a lot of documentation configurations. It is very well commented and easy to understand.

There are also the **_static** and **_templates** directories which are empty if the user does not fill them in. These contain style-related files.

The start page of a documentation project will be generated from the master file (as named by the user in the documentation creation session).

In the same directory as the master file (**index.rst** if not specified otherwise), the user can create the other documentation files. Every file would correspond to a HTML page being generated after building. To refer to other file (page) in a file's table of contents the toctree directive is used. If files are created in another directory, the path to them should be also added as in the example below.:

```
1  .. toctree::
2     :maxdepth: 2
3
4     document1.rst or /path/to/from/current/directory/document2.rst
5     document3.rst
6     ...
```

After building, the build directory will be written, containing doctrees directory and HTML directory. The HTML directory is the one containing the HTML output. With then toctree as above a hypothetical page will have links generated for the other pages- document1, document#, document2, etc.

While building the documentation, explore how the build and HTML directories change. While there is direct control on the source one, the build directory contains the HTML output so that is why it is important to observe the content. Also, changes will not become visible if the working **rst** (txt) file is not saved. Normally a message is displayed if the files in the source directory are up to date and about which files are exactly build. While building, messages show which files are being processed. It is important to observe the process in order to avoid surprises.

With Sphinx, also LaTeX and PDF output can be generated. The command for generating LaTeX output is `make latex` and for generating PDF of the documentation- `make latexpdf`. Similar to the HTML output a LaTeX output directory will appear in the build directory, containing the generated files.

Building a PDF is not trivial. The process is sensitive to particular problems with the interpretation of LaTeX as it seems. For example, the usage of **eqnarray** in the output causes errors. They can be still *solved* by using LaTeX line breaks (`\\`) and still write the equations on the same line. The building process puts the result in `/source/LaTeX` but then it is not yet available as a link to your HTML documentation, so you should run 'make html' again. Please ignore LaTeX warnings by pressing 'Enter' while building PDF to at least force some usable output. Also check at the end if the PDF was generated. With severe errors, the building process does not create a newer version. The PDF would include documents in the toctree. If the user wants to refer to a rst document without putting it in the toctree and generating PDF of it, he/she could use the *doc role*.

As already visible in the creation phase, TeXinfo and Epub output can be also generated.

For resume of the various builders and output possibilities, the reader can visit the following page with available builders <http://sphinx-doc.org/builders.html>.

5. Basic syntax

Headings and comments

All content to a sphinx page can be added with plain text. There is no hard syntax to learn.

The sections in a file as the one below are considered comments:

```
.. text
```

When a title of a subsection has to be created, the line needs to be underlined with #, =, *, -, ^ or ". There are no heading levels assigned but it is important to keep consistency because the structure is determined from the succession of headings. It is good to experiment at first to find the most comfortable for the user way. Of course, the user is free to use other characters but it is important to realize that there is limitation how deep the nesting of sections can be (limited for LaTeX)

The underlining can be longer than the title, but not shorter:

```
1 Title -valid header
2 =====
3
4 Title -invalid header
5 ===
6
7 =====
8 This is also valid heading
9 =====
```

Every title is being added as node in the table of contents of the page, adding up to the nodes created by the toctree. If the user does not want that, he or she should use:

```
.. rubric:: title
```

Paragraphs and text boxes

Paragraphs start on a new line, separated from previous and next sections with a blank line.

Sphinx works with directives and options. Several directives can be used specifically in order to improve the text arrangement.

For a coloured box with some title the user can apply the topic directive:

```
.. topic:: title

Text
```

It is very important to keep the indentation and the blank line, also a blank line at the end of every logical block in the plain text.

For a warning box the syntax is as follows:

```
.. Warning::

Text
```

For 'seealso' box and note box, the idea is the same:

```

1  .. seealso::
2
3      Text
4
5  .. note::
6
7      Text

```

The results would be respectively:

title

Text

Warning: Text

See also: Text

Note: Text

A lot more options are available.

Todo boxes are also possible:

```

.. todo::

    Text

```

Resulting in:

Todo: Text

Before being able to use the directive, the user needs to add the following extension: **sphinx.ext.todo** (if not present in the array with extensions at **conf.py** and also set the variable **todo_include_todos** to **True**. If there is no line for this in **conf.py**, then one should be added:

```
todo_include_todos= True
```

Warning: The style hereby is custom. All blocks could be given different background and general appearance throughout style sheet changes.

Literal text can be displayed with literal text markup `text` within a line or a whole paragraph- with `”::”` at the end of the previous paragraph:

```

Previous papapagraph. text... text::

    Literal paragraph... indentation should be used.

```

Markup within a paragraph such as `*text*` or `**text**` stands respectively for italics and emphasis. Inline literal blocks could be achieved by: ```text```.

Pictures and figures

To insert a picture, the user needs to add the following structure:

```

1  .. image:: ../images/image.png
2      :width: ??px
3      :align: center
4      :height: ??px
5      :alt: caption

```

To import the image as figure the syntax is rather the same:

```

1  .. figure:: ../images/image.png
2      :width: ??px
3      :align: center
4      :height: ??px
5      :alt: caption
6
7      Could add text here

```

It has to be notified that the specified path is relative to the source directory. This means that if directly the title of the image is given, it is in the same directory as the rst files.

Lists and tables

Creating lists is very easy job. The simplest list is basically a list of bullet points:

- item1
 - item2
 - item3
-
- item1
 - item2
 - item3

Numbered lists just include numbering:

```

1. item1
2. item2
3. item3

```

```

1. item1
2. item2
3. item3

```

Autonumbering is also possible:

```

#. autonumber item

```

```

1. autonumber item

```

In order to make the list looking more clean, the user can apply the following:

```

1  .. hlist::
2      :columns: 3
3
4      * A list of
5      * short items
6      * that should be
7      * displayed
8      * horizontally

```

- | | | |
|---------------|------------------|----------------|
| • A list of | • that should be | • horizontally |
| • short items | • displayed | |

Warning: Depending on the version of the builder that is present on the client machine, the effect can vary.

Tables are also very simple to create. All the user needs is to “draw it”.

Example:

```

1  +-----+-----+-----+-----+
2  | Header   Title      | Title      | Title      | Title      |
3  +-----+-----+-----+-----+
4  | row1      column 1   | column 2   | column 3   | column 4   |
5  +-----+-----+-----+-----+
6  | row 2      ...      | ...      | ...      | ...      |
7  +-----+-----+-----+-----+
8

```

Header Title	Title	Title	Title
row1 column 1	column 2	column 3	column 4
row 2

The “=” underlining is optional. When added it emphasizes and colors the headers.

Other table options are also available

A	B	C
.	.	.
.	.	.
.	.	.

A	B	C
.	.	.
.	.	.
.	.	.

A very simple table format is the **csv-table**. It does not require pre-drawing:

```

1  .. csv-table:: a title
2     :header: "first header", "second header", "third header"
3     :widths: 20, 20, 10
4
5     "item 1", "item 2", 3
6     "item 4", "item 5", 5

```

a title		
first header	second header	third header
item 1	item 2	3
item 4	item 5	5

However, it has to be noted that with this format there is no control over the merging of cells.

Warning: The user should be careful with the spacing in the **csv-table** syntax. The comma needs to be exactly after the quotes and 1 space separated from the next quotes.

To control the text line break in a listing without using tables or lists, the user can apply the following topology:

line 1
line 2

line 1
line 2

A lot of other possibilities are available for stylish output. Separate extensions and style methods can also be written in

order to reach the desired output.

6. Page content and documentation structure

Sphinx documentation has a simple structure. It consists of the files that were created and the connections between them are specified within the files toctrees (table of contents).

The paragraph titles from every document are included in the table of contents. Tables of contents from all sub documents are inserted also, with a maximum depth of `maxdepth`. One uses `maxdepth` to avoid too much detail in a table of contents:

```
1 .. toctree::
2     :maxdepth: 2
3
4     document1.rst
5     document2.rst
6     ...
```

A depth of 1 displays only the sub documents the current page. A depth of 2 displays the document names and the titles of their subsections as well. Higher `:maxdepth:` is analogous. In order to remove a single section header from the table of contents, look at the [answer of question 12](#).

7. Sphinx configuration file

The main Sphinx configuration file can be found in the source and also in the HTML output (after building) directories. Hereby we supply an example of a modified configuration file.

Warning: This file has been modified for our documentation project and will differ from yours!

You can download and inspect this configuration file: [conf.py](#)

Lets start with the initial section. It represents the general configuration:

```
1  # -*- coding: utf-8 -*-
2  #
3  # Introduction to Programming and Computational Methods build configuration file, created by
4  # sphinx-quickstart on Tue Aug 27 11:53:19 2013.
5  #
6  # This file is execfile()d with the current directory set to its containing dir.
7  #
8  # Note that not all possible configuration values are present in this
9  # autogenerated file.
10 #
11 # All configuration values have a default; values that are commented out
12 # serve to show the default.
13
14 import sys, os
15 CURDIR = os.path.abspath(os.path.dirname(__file__))
16 # If extensions (or modules to document with autodoc) are in another directory,
17 # add these directories to sys.path here. If the directory is relative to the
18 # documentation root, use os.path.abspath to make it absolute, like shown here.
19 sys.path.append(os.path.join(os.path.abspath(os.path.dirname(__file__)), 'sphinxext'))
20
21 # -- General configuration -----
22
23 # If your documentation needs a minimal Sphinx version, state it here.
24 #needs_sphinx = '1.0'
25
26 # -- plot directive script:
27 #
28 #plot_ext = 'matplotlib.sphinxext.plot_directive' # 'official'
29 plot_ext = 'plot_directive' # local
30
31 plot_figwidth = '60%' # Works with latest plot_directive adapted by Hans Terlouw 12-4-2014
32
33 math_ext = 'sphinx.ext.mathjax'
34 mathjax_path = '/MathJax/MathJax.js?config=TeX-AMS-MML_HTMLorMML'
35
36 # Add any Sphinx extension module names here, as strings. They can be extensions
```

```

37 # coming with Sphinx (named 'sphinx.ext.*') or your custom ones.
38 extensions = ['sphinx.ext.autodoc', 'sphinx.ext.doctest', 'sphinx.ext.intersphinx', 'sphinx.ext.todo',
39              'sphinx.ext.coverage', math_ext, plot_ext, 'sphinx.ext.ifconfig', 'sphinx.ext.viewcode',
40              'figtable', 'question']
41
42 todo_include_todos=True #To allow todo notes and lists
43
44 # Add any paths that contain templates here, relative to this directory.
45 templates_path = ['_templates']
46
47 # The suffix of source filfile names
48 source_suffix = '.rst'
49
50 # The encoding of source files.
51 #source_encoding = 'utf-8-sig'
52
53 # The master toctree document.
54 master_doc = 'index'
55
56 # General information about the project.
57 project = u'Introduction to Programming and Computational Methods'
58 copyright = u'2014, Martin Vogelaar'
59
60 # The version info for the project you're documenting, acts as replacement for
61 # |version| and |release|, also used in various other places throughout the
62 # built documents.
63 #
64 # The short X.Y version.
65 version = '1.0'
66 # The full version, including alpha/beta/rc tags.
67 release = '2014'
68
69 # The language for content autogenerated by Sphinx. Refer to documentation
70 # for a list of supported languages.
71 #language = None
72
73 # There are two options for replacing |today|: either, you set today to some
74 # non-false value, then it is used:
75 #today = ''
76 # Else, today_fmt is used as the format for a strftime call.
77 #today_fmt = '%B %d, %Y'
78
79 # List of patterns, relative to source directory, that match files and
80 # directories to ignore when looking for source files.
81 exclude_patterns = []
82
83 # The reST default role (used for this markup: `text`) to use for all documents.
84 #default_role = None
85
86 # If true, '()' will be appended to :func: etc. cross-reference text.
87 #add_function_parentheses = True
88
89 # If true, the current module name will be prepended to all description
90 # unit titles (such as .. function::).
91 #add_module_names = True
92
93 # If true, sectionauthor and moduleauthor directives will be shown in the
94 # output. They are ignored by default.
95 #show_authors = False
96
97 # The name of the Pygments (syntax highlighting) style to use.
98 pygments_style = 'sphinx'
99
100 # A list of ignored prefixes for module index sorting.
101 #modindex_common_prefix = []

```

The comments help a lot to understand the functionality of an option. This example is extracted from a file which was customized by us. If the user opens the original file, it becomes obvious what has been changed.

The lines:

```

import sys, os
CURDIR = os.path.abspath(os.path.dirname(__file__))

```

help specifying the directory that contains the **conf.py** file.

By the following:

```
sys.path.append(os.path.join(os.path.abspath(os.path.dirname(__file__)), 'sphinxext'))
```

A user can specify a directory which includes the extensions location. In the current example, a directory called **sphinxext** was created in the source directory which was filled in with any **custom** extensions.

The user can add also other directories in the source directory which could be filled in with files for referencing. For example, the user could add a directory for text files, pictures and other. This is not going to change the documentation structure. These directoried could be also outside the source directory, but when referencing their pat, it should be known relative to the source location. The locations do not need to be added in the configuration file.

The lines:

```
1  # -- plot directive script:
2  #
3  #plot_ext = 'matplotlib.sphinxext.plot_directive' # 'official'
4  plot_ext = 'plot_directive'                     # local
5
6  plot_figwidth = '60%' # Works with latest plot_directive adapted by Hans Terlouw 12-4-2014
```

show that the user had created a custom extension **plot_directive** (positioned in the created sphinxext directory). That is why the official extension declaration is commented.

The last line indicates that this new extension is having options that can be configured in the current file. Later on in the page [custom extension section](#) the reader can find the code of this custom extension, illustrating the possibility Sphinx provides for customisation.

Continuing further, we see the specification of the path for **MathJax** (in case this is the math option sselected):

```
math_ext = 'sphinx.ext.mathjax'
mathjax_path = '/MathJax/MathJax.js?config=TeX-AMS-MML_HTMLorMML'
```

The official extension is used and its location on the server is specified. In case that the reader uses Sphinx on a personal machine, then the path to where MathJax is located might be different of course. The path can be absolute or relative. If it is relative, it is relative to the `_static` directory of the built docs.

The next part of the general configuration continues with the extensions that are used in the documentation. A list of strings that are module names of Sphinx extensions. These can be extensions coming with Sphinx (named `sphinx.ext.*`) or custom ones:

```
extensions = ['sphinx.ext.autodoc', 'sphinx.ext.doctest',
             'sphinx.ext.intersphinx', 'sphinx.ext.todo', 'sphinx.ext.coverage',
             math_ext, plot_ext, 'sphinx.ext.ifconfig', 'sphinx.ext.viewcode',
             'figtable', 'question']
```

In the current example, the user can find several custom extensions next to the other ones, already provided by Sphinx. The remaining part of the general configuration is self explanatory. Some options and paths have been already specified in the documentation creation phase as well.

The next section in the configuration file is the HTML output section:

```
1  # -- Options for HTML output -----
2
3  # The theme to use for HTML and HTML Help pages. See the documentation for
4  # a list of builtin themes.
5  html_theme = 'default'
6
7  # Theme options are theme-specific and customize the look and feel of a theme
8  # further. For a list of options available for each theme, see the
9  # documentation.
10 #
```

```

11 #html_theme_options
12
13 # Add any paths that contain custom themes here, relative to this directory.
14 #html_theme_path = []
15
16 # The name for this set of Sphinx documents.  If None, it defaults to
17 # "<project> v<release> documentation".
18 #html_title = None
19 html_title = "Introduction to Programming and Computational Methods"
20
21 # A shorter title for the navigation bar.  Default is the same as html_title.
22 #html_short_title = None
23 html_short_title = "Prognum"
24
25
26 # The name of an image file (relative to this directory) to place at the top
27 # of the sidebar.
28 #html_logo = None
29
30 # The name of an image file (within the static path) to use as favicon of the
31 # docs.  This file should be a Windows icon file (.ico) being 16x16 or 32x32
32 # pixels large.
33 #html_favicon = None
34
35 # Add any paths that contain custom static files (such as style sheets) here,
36 # relative to this directory. They are copied after the builtin static files,
37 # so a file named "default.css" will overwrite the builtin "default.css".
38 html_static_path = ['_static']
39
40 # If not '', a 'Last updated on:' timestamp is inserted at every page bottom,
41 # using the given strftime format.
42 #html_last_updated_fmt = '%b %d, %Y'
43
44 # If true, SmartyPants will be used to convert quotes and dashes to
45 # typographically correct entities.
46 #html_use_smartypants = True
47
48 # Custom sidebar templates, maps document names to template names.
49 #html_sidebars = {}
50
51 # Additional templates that should be rendered to pages, maps page names to
52 # template names.
53 #html_additional_pages = {}
54
55 # If false, no module index is generated.
56 html_domain_indices = True
57
58 # If false, no index is generated.
59 html_use_index = True
60
61 # If true, the index is split into individual pages for each letter.
62 #html_split_index = False
63
64 # If true, links to the reST sources are added to the pages.
65 html_show_sourcelink = False
66
67 # If true, "Created using Sphinx" is shown in the HTML footer. Default is True.
68 #html_show_sphinx = True
69
70 # If true, "(C) Copyright ..." is shown in the HTML footer. Default is True.
71 html_show_copyright = True
72
73 # If true, an OpenSearch description file will be output, and all pages will
74 # contain a <link> tag referring to it.  The value of this option must be the
75 # base URL from which the finished HTML is served.
76 #html_use_opensearch = ''
77
78 # This is the file name suffix for HTML files (e.g. ".xhtml").
79 #html_file_suffix = None
80
81 # Output file base name for HTML help builder.
82 htmlhelp_basename = 'IntroductiontoProgrammingandComputationalMethodsdoc'

```

The explanation in the comments is crucial for the understanding the functionality. Later on the usage of themes is shortly reviewed. Hereby the **default** HTML theme provided by Sphinx is used. The path to the style sheets specified to be `html_static_path = ['_static']`, which is the **_static** directory in the source of the documentation. Also, the

HTML is provided with copyright for the example above. Index generation is allowed.

(depends if the user wants to expose the rst code or not), the page source could be hidden by `html_show_sourcelink = False` in `conf.py` as above.

After the HTML output section, the configuration file contains similar section for the LaTeX output:

```

1  # -- Options for LaTeX output -----
2
3  latex_elements = {
4      # The paper size ('letterpaper' or 'a4paper').
5      #'papersize': 'letterpaper',
6
7      # The font size ('10pt', '11pt' or '12pt').
8      #'pointsize': '10pt',
9
10     # Additional stuff for the LaTeX preamble.
11     #'preamble': '',
12 }
13
14 # Grouping the document tree into LaTeX files. List of tuples
15 # (source start file, target name, title, author, documentclass [howto/manual]).
16 latex_documents = [
17     ('index', 'IntroductiontoProgrammingandComputationalMethods.tex',
18      u'Introduction to Programming and Computational Methods',
19      u'Martin Vogelaar', 'manual'),
20 ]
21
22 # The name of an image file (relative to this directory) to place at the top of
23 # the title page.
24 #latex_logo = None
25
26 # For "manual" documents, if this is true, then toplevel headings are parts,
27 # not chapters.
28 #latex_use_parts = False
29
30 # If true, show page references after internal links.
31 #latex_show_pagerefs = False
32
33 # If true, show URL addresses after external links.
34 latex_show_urls = False
35
36 # Documents to append as an appendix to all manuals.
37 #latex_appendices = []
38
39 # If false, no module index is generated.
40 latex_domain_indices = True

```

Hereby the reader can find the options for the paper size, appendices, font size, chapter numbering etc. The LaTeX documents section is specified already in the documentation creation section.

The `conf.py` file then contains the following sections, analogous to the LaTeX output section, dealing with the options for manual page, Texinfo and Epub output:

```

1  # -- Options for manual page output -----
2
3  # One entry per manual page. List of tuples
4  # (source start file, name, description, authors, manual section).
5  man_pages = [
6      ('index', 'introductiontoprogrammingandcomputationalmethods',
7       u'Introduction to Programming and Computational Methods',
8       [u'Martin Vogelaar'], 1)
9  ]
10
11 # If true, show URL addresses after external links.
12 #man_show_urls = False
13
14
15 # -- Options for Texinfo output -----
16
17 # Grouping the document tree into Texinfo files. List of tuples
18 # (source start file, target name, title, author,
19 #  dir menu entry, description, category)
20 texinfo_documents = [

```

```

20 texinfo_documents = [
21     ('index', 'IntroductiontoProgrammingandComputationalMethods',
22      u'Introduction to Programming and Computational Methods',
23      u'Martin Vogelaar', 'IntroductiontoProgrammingandComputationalMethods',
24      'One line description of project.',
25      'Miscellaneous'),
26 ]
27
28 # Documents to append as an appendix to all manuals.
29 #texinfo_appendices = []
30
31 # If false, no module index is generated.
32 #texinfo_domain_indices = True
33
34 # How to display URL addresses: 'footnote', 'no', or 'inline'.
35 #texinfo_show_urls = 'footnote'
36
37
38 # -- Options for Epub output -----
39
40 # Bibliographic Dublin Core info.
41 epub_title = u'Introduction to Programming and Computational Methods'
42 epub_author = u'Martin Vogelaar'
43 epub_publisher = u'Martin Vogelaar'
44 epub_copyright = u'2014, Martin Vogelaar'
45
46 # The language of the text. It defaults to the language option
47 # or en if the language is not set.
48 #epub_language = ''
49
50 # The scheme of the identifier. Typical schemes are ISBN or URL.
51 #epub_scheme = ''
52
53 # The unique identifier of the text. This can be a ISBN number
54 # or the project homepage.
55 #epub_identifier = ''
56
57 # A unique identification for the text.
58 #epub_uid = ''
59
60 # A tuple containing the cover image and cover page html template filename
61 #epub_cover = ()
62
63 # HTML files that should be inserted before the pages created by sphinx.
64 # The format is a list of tuples containing the path and title.
65 #epub_pre_files = []
66
67 # HTML files that should be inserted after the pages created by sphinx.
68 # The format is a list of tuples containing the path and title.
69 #epub_post_files = []
70
71 # A list of files that should not be packed into the epub file.
72 #epub_exclude_files = []
73
74 # The depth of the table of contents in toc.ncx.
75 #epub_tocdepth = 3
76
77 # Allow duplicate toc entries.
78 #epub_tocdup = True

```

At the end of this section the intersphinx mapping configuration is discussed. In order to be able to reference parts of other Sphinx documentations, the intersphinx extension (`sphinx.ext.intersphinx`) should be added in the extensions section. It can generate automatic linking to objects in other Sphinx documentation projects. The user has to specify in the configuration file which are the documentation projects of interest:

```

1  # -- Intersphinx linking -----
2
3  intersphinx_cache_limit = 10      # days to keep the cached inventories
4  _python_doc_base = 'http://docs.python.org/2.7'
5  _numpy_doc_base = 'http://scikit-learn.org/stable'
6  intersphinx_mapping = {
7      _python_doc_base: None,
8      'http://docs.scipy.org/doc/numpy': None,
9      'http://docs.scipy.org/doc/scipy/reference': None,
10     _numpy_doc_base: None,
11     'matplotlib': ('http://matplotlib.sourceforge.net', None),
12     'pyfits': ('http://pythonhosted.org/pyfits/', None)
13 }

```

The example above illustrates that there are 2 ways of specifying the link to the documentation- through direct citing or with a variable. Also, the last 2 lines give another alternative syntax which also uses identifier before the tuple. All of them are correct and can be applied.

The code above works simple. The target (the base URI of external documentation or local path) is mapped to its inventory (inventory file). The key word “None” indicates that the objects inventory file (**obj.inv**) can be found at the same location. If not, another URI should be supplied.

In order to be able to reference an object from other documentation, this object should be part of the inventory file.

The rst syntax for referencing is discussed later in the second part of the howto [Intersphinx mapping](#).

8. Page basic style understanding

style sheets and configuration

There are 3 files responsible for the sphinx page appearance. These are the **basic.css**, **default.css** and **pygments.css**, positioned in the `_static` directory under `build/html`. This is in case the **default** HTML theme has been selected in **conf.py** - [HTML output configuration](#).

Sphinx allows changing of the style of the HTML via themes. Theme option is present in the Sphinx configuration file **conf.py**. There are several options, explained on this page: <http://sphinx-doc.org/theming.html>.

An example of setting up the default theme is given below. In this example the sidebar is positioned at the right:

```

html_theme = "default"
html_theme_options = {
    "rightsidebar": "true"
}

```

The page <http://sphinx-doc.org/theming.html> also explains the options that can be altered in the **conf.py** file. However, altering the configuration file might not bring all the desired output. Especially for other themes (different than the default one), options might not be available. In such cases the user can alter the style sheets themselves. He/She needs to copy the style sheets from the `_static` directory under `build/html` to the `_static` directory under **source**. In that case, whatever is changed in the style sheets under the source location will be remembered after building the documentation

basic examples (default theme used)

In this section there are only basic examples.

In case options are not used in **conf.py**, to change the position of the sidebar and align it from left to the right, the **default.css** and the **basic.css** files need to be altered.

In **basic.css**:

```

1  div.sphinxsidebar {
2      float: left--->right; -change from left to right
3      width: 230px;
4      margin-left: -100%;
5      font-size: 90%;
6  }

```


In **default.css**

```
div.bodywrapper {
    margin: 0 0 0 230px---> 0 230px 0 0;
}
```

Under the default theme the sidebar location can be changed via **conf.py** as well. Look at the [HTML theming support](#)

For changing the background of the sidebar (the page background), the **default.css** file needs to be altered. The location in the file is:

```
div.document {
    background-color: #1c4e63;
}
```

To change the text background the location is:

```
1  div.body {
2      background-color: #ffffff;
3      color: #000000;
4      padding: 0 20px 30px 20px;
5  }
```

Also look at the [HTML theming support](#)

custom style methods

In the style sheet, the user can define custom methods and later on use them in the rst files after defining a role with the method name.

For example, if a user would like to have a method for inline text color, then the following method can be created in one of the style sheets **default.css** or **basic.css** or other style sheets:

```
1  .red {
2
3      color: red;
4
5  }
```

Then the following should appear at the beginning of the rst file where the method should be used:

```
.. role:: red
```

From then on the user could use `:red:`text`` in order to have the text in **red**. When using the container directive, a whole paragraph can be given particular style:

```
.. container:: red
    Here the full block is red.
```

Here the full block is red.

Part II

1. Displaying mathematics and generating plots

Inline mathematics, equation arrays and referencing

The current page uses **MathJax** for displaying the mathematics.

The following procedure illustrates the use of inline mathematics:

```
Text :math:`a^2+ b^2= c^2` text.
```

Text $a^2 + b^2 = c^2$ | text.

To display single equation on a new row the following syntax is used:

```
.. math:: a^2+ b^2= c^2
   :label: some_label1
```

or:

```
.. math::
   :label: some_label2

   a^2+ b^2= c^2
```

$$a^2 + b^2 = c^2 \quad (1)$$

$$a^2 + b^2 = c^2 \quad (2)$$

In the example above `some_label` refers to the equation name and how the user can reference it. The reader needs to use `:eq: `some_label`` to refer to the equation. Example: (1), (2).

The signs are traditional for the TeX format: “+” is used for plus, “-” for minus, “*” for multiplication, “`frac{ }{ }`” for division, “^” for powers, etc.

To display arrays of mathematical equations, the following option is very suitable, because later on the equations can be referenced under the same label:

```
1 .. math::
2   :label: some_label3
3
4   \begin{eqnarray}
5     a^2+ b^2= c^2\\
6     a^2+ b^2= c^2\\
7     a^2+ b^2= c^2\\
8   \end{eqnarray}
```

or:

```
.. math:: a^2+ b^2= c^2\\ a^2+ b^2= c^2\\ a^2+ b^2= c^2\\
   :label: some_label4
```

$$\begin{array}{l} a^2 + b^2 = c^2 \\ a^2 + b^2 = c^2 \\ a^2 + b^2 = c^2 \end{array} \quad (3)$$

$$\begin{array}{l} a^2 + b^2 = c^2 \\ a^2 + b^2 = c^2 \\ a^2 + b^2 = c^2 \end{array} \quad (4)$$

Display options: PNGmath and MathJax

There are 2 options to display mathematics in Sphinx generated document. One of them is using PNGmath- showing the equations as an image and the other is using MathJax. MathJax is a cross-browser JavaScript library that displays mathematical notation, using MathML, LaTeX and ASCII MathML markup.

During installation the user has a option to choose between the 2 options. In the case MathJax is used it also needs to be installed. For speed and reliability we installed MathJax on a local server.

The installation is explained on (in case installation is performed on local machine): <http://docs.mathjax.org/en/v1.1-latest/installation.html>.

Following the suggested tests in the link is advisable. It is nice to have the MathJax directory within the documentation directory.

If the mathematics is still displayed as image, the user can check the `conf.py` file (within the source and build directory). The extensions section may still be ordered in a way that the MathJax extension is overwritten.

This is an example of the extensions section:

```
extensions = ['sphinx.ext.autodoc', 'sphinx.ext.doctest', 'sphinx.ext.intersphinx',
             'sphinx.ext.todo', 'sphinx.ext.coverage',
             'sphinx.ext.mathjax', 'sphinx.ext.ifconfig', 'sphinx.ext.viewcode']
```

Only one extension should be kept for the mathematics output.

The font size of the block mathematics can be altered from the **default.css** style sheet or other style sheet being used):

```
1  div.math {
2      text-align: center;
3      margin: 1em 0em;
4      position: relative;
5      display: block;
6      width: 100%;
7      font-size: 90%;
8  }
```

For the inline mathematics the procedure would include a more complicated approach and was not done within our PROGNUM documentation. The MathJax configuration file has to be altered.

Generating plots automatically

It is possible to generate plots automatically using Python code. The tool that makes this possible is the so called plot directive:

```
.. plot:: pyplots/code.py
```

From above, `pyplots` corresponds to the new directory the user needs to create within the source, and this new directory should contain the Python code `code.py`. As a result, the new “plot_directive” will be created within the build directory and also the plots will appear as images within the `_images` directory within `build/html`.

On the website this will result in a good quality output, available for downloading. The user can choose to display or hide the code which created the plot. In order to display it, the “include-source” option should be added:

```
.. plot:: pyplots/code.py
   :include-source:
```

In the examples below the plot is generated from external file and `:include-source:` is used for the first graph while for the second one not:

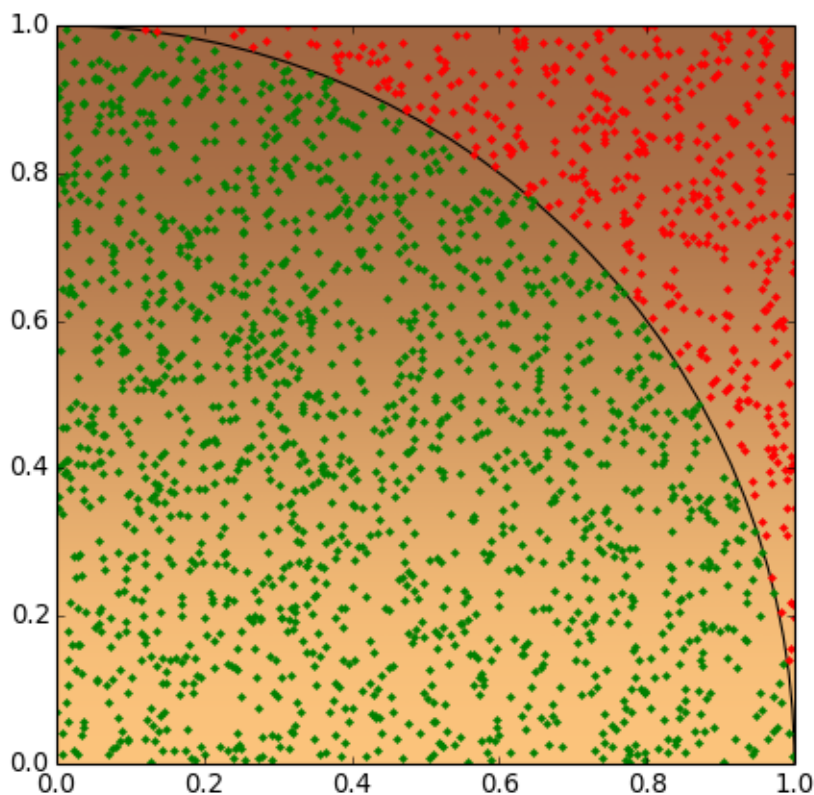
```
.. plot:: examples/circleMC.py
   :include-source: - not for second plot
```

```

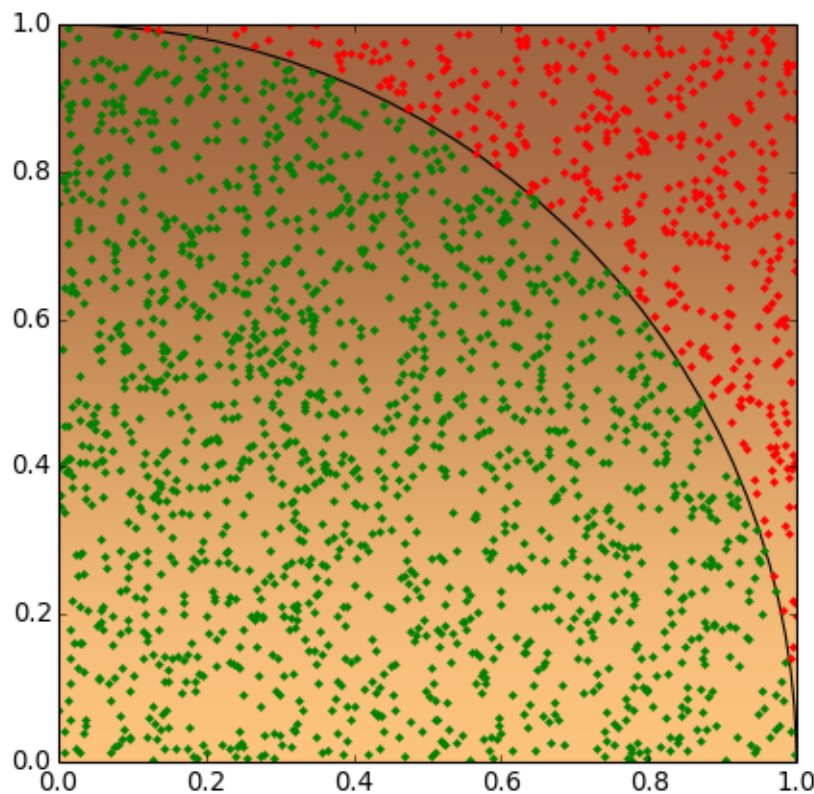
1  import numpy as np
2  from matplotlib.pyplot import figure, show
3  from matplotlib.patches import Arc
4  from matplotlib.cm import copper
5
6  fig = figure()
7  frame = fig.add_subplot(1,1,1)
8  n = 2000
9  x = np.random.random_sample(n)
10 y = np.random.random_sample(n)
11 radius = x**2 + y**2
12 In = (radius <= 1.0)
13 Out = (radius > 1.0)
14 frame.plot(x[In], y[In], 'g.')
15 frame.plot(x[Out], y[Out], 'r.')
16 arc = Arc((0,0), 2,2, 0,0, 90)
17 frame.add_patch(arc)
18 frame.imshow([[0, 0],[1,1]], interpolation='bicubic', cmap=copper,
19             vmin=-0.5, vmax=0.5,
20             extent=(frame.get_xlim()[0], frame.get_xlim()[1],
21                   frame.get_ylim()[0], frame.get_ylim()[1]),
22             alpha=1)
23
24 frame.set_xlim(0,1)
25 frame.set_ylim(0,1)
26 frame.set_aspect(1.0)
27 show()

```

([Source code](#), [png](#), [hires.png](#), [pdf](#))



([Source code](#), [png](#), [hires.png](#), [pdf](#))



Warning: Before being able to use the standard `plot_directive`, the user needs to include the following extension in `conf.py`: `extensions = ['matplotlib.sphinxext.plot_directive' ...]`. The extension itself is written for Matplotlib and the code is available at: https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/sphinxext/plot_directive.py

A `plot_directive` directory will appear in the build directory, including the images and PDF's produced for the plotting and showing of the source code.

Later in the howto there is an explanation on customizing the `plot_directive` extension [Customizing the plot_directive](#).

2. Links

External links

External links are very easy to display. The user needs to apply the following structure: ``Link text <http://link.com/>`_`

This means that in order to connect to google.com for example, we need to write: ``Google <http://google.com/>`_`

Internal links

If the user intends to put internal links, the procedure is as follows. First of all a reference label needs to be put at the place which is going to be invoked. The label is put, using:

```
.. _label:
```

To put a link to the specified place, the user just adds in the text: `:ref:`label``. When put above section header, the label reference will obtain the name of the section. When the label is not put above a section header, it needs to be given an additional name when referenced: `:ref:`A link name <label>``

If the user wants to refer to other rst document inside the documentation, then he/she could do it with `:doc:`name <document_name>``. In that case the document does not need to be in the toctree. When building, there will be an error that states that there is an rst document in the documentation which is not included in any toctree. The user needs to add `:orphan:` at the beginning of this rst document and the error will disappear

To refer to any downloadable file the syntax is:

```
:download:`name <path_to_document_relative_to_source>`.
```

Intersphinx linking

If an intersphinx link needs to be created, then first an intersphinx mapping should be present in the configuration file “conf.py”. After that the rst syntax is straightforward. Below there are several examples given:

```
1 :ref:`Link name <intersphinx target>` or :ref:`intersphinx target`
2 :meth:`Link name <intersphinx target>` or :meth:`intersphinx target`
3 :func:`Link name <intersphinx target>` or :func:`intersphinx target`
4 :class:`Link name <intersphinx target>` or :class:`intersphinx target`
5 :mod:`Link name <intersphinx target>` or :mod:`intersphinx target`
6 :attr:`Link name <intersphinx target>` or :attr:`intersphinx target`
```

As the reader can see, intersphinx can be used for referencing keywords, functions, methods, classes, modules, attributes, etc.

Below there are several examples:

- Scipy's function `odeint()`

```
:func:`odeint() <scipy.integrate.odeint>` odeint()
:func:`scipy.integrate.odeint` scipy.integrate.odeint()
```
- Scipy's function `quad()`

```
:func:`quad() <scipy.integrate.quad>` quad()
:func:`scipy.integrate.quad` scipy.integrate.quad()
```
- Matplotlib's method `hist`

```
:meth:`hist() <matplotlib.axes.Axes.hist>` hist()
:meth:`matplotlib.axes.Axes.hist` matplotlib.axes.Axes.hist()
```
- Numpy module

```
:mod:`numpy <numpy>` numpy
:mod:`numpy` numpy
```
- Numpy's class `matrix`

```
:class:`matrix <numpy.matrix>` matrix
:class:`numpy.matrix.` numpy.matrix
```
- Numpy's matrix attribute `shape`

```
:attr:`shape() <numpy.ndarray.shape>` shape()
:attr:`numpy.ndarray.shape` numpy.ndarray.shape
```
- Python if statement

```
:ref:`if <if>` if
:ref:`if` The if statement
```

The user has to know that these links will only work if there is a reference label in the original documentation. For example such a label is not available for numpy's universal functions **ufunc**.

The following link gives an example for such a function.

<http://docs.scipy.org/doc/numpy/reference/generated/numpy.sin.html#numpy.sin>

The following thus do not work:

```
:func:`numpy.sin` numpy.sin()
```

Indexing

The index is the upper right link on every page.

If a user wants to refer to a particular location in the documentation from the index table, it is enough to use simply `:index:`text_in_paragraph``.

There is also a directive `index` which does the same at a whole paragraph level. That is why it needs to be positioned above the paragraph with the location of interest. At the following link there are several <http://sphinx-doc.org/markup/misc.html>.

3. Code blocks

To create a Python code-block the user can type simply type:

```
.. code-block:: python

    code here
```

In order to number the lines, the user can type the `:linenos:` option. For example, consider the code-block below:

```
1  .. code-block:: python
2  :linenos:
3
4  from __future__ import division
5  import numpy
6
7  def volume(height, radius):
8      pi = 3.14
9      vol = (1.0/3.0) * height * pi * pow(radius,2)
10     return vol
11
12     vol = volume(2.0, 10)
13     print vol, "(m^3)"
```

```
1  from __future__ import division
2  import numpy
3
4  def volume(height, radius):
5      pi = 3.14
6      vol = (1.0/3.0) * height * pi * pow(radius,2)
7      return vol
8
9  vol = volume(2.0, 10)
10  print vol, "(m^3)"
```

Long literal blocks (code or other, in LaTeX verbatim) could be also included via a file, so they do not need to be written in the rst code. Then the user can apply the **literalinclude** directive:

```
.. literalinclude:: file_including_code
```

Example: - *Examine the complex plane with phase plots*

```

1  import numpy
2  from matplotlib.pyplot import figure, show
3
4  xlo = -2; xhi = 2
5  ylo = xlo; yhi = xhi
6  x = numpy.linspace(xlo,xhi,100)
7  y = numpy.linspace(ylo,yhi,100)
8  xx,yy = numpy.meshgrid(x,y)
9  z = xx + yy*1j
10 zi = (z**2-1j)/(2*z**2+2j)
11 zi = numpy.angle(zi)
12
13 fig = figure()
14 frame = fig.add_subplot(1,1,1)
15 frame.imshow(zi, origin='lower', extent=[xlo,xhi, ylo,yhi])
16 show()

```

The code that the reader sees has been numbered without adding an option for that in the directive. Automatic numbering of the code blocks (the literal blocks starting with `::` are also code blocks) can be initiated by the highlights directive:

```

.. highlight:: python
   :linenothreshold: 5

```

The code above positioned at the beginning of the rst file would number all Python code blocks with lines more than 5.

Warning: The **Pygments** style sheet is responsible for this. It needs to be present. The example for the complex numbers was numbered exactly because in the current page the highlight directive is used.

The highlight directive could be set for other languages such as C for example. Every highlight is active until the next usage of the directive. Additionally, the individual code-blocks could be used with the `guess` option in order to detect the right language.

More on the usage of highlights and code-blocks can be found at:

<http://sphinx-doc.org/markup/code.html>

4. Usage of customized extensions

As it was mentioned already, custom extensions can be written in order to supply new desired functionality when the provided Sphinx extensions are not able to provide it. In this section there are 2 examples of custom extensions. The custom extensions should be mentioned in the configuration file and also supplied in an extension directory whose path is also given in **conf.py**. Check again the [General configuration section](#)

Custom plot_directive

Sphinx provides extensions for including Matplotlib plots in a Sphinx document. The plot is generated from Python code while building the documentation. This directive can be altered in a way to provide numerous options for the user for better control.

The code below represents a plot directive extension, adapted by Hans Terlouw. While the original `plot_directive` has limited control (caused by a bug) the modified extension makes it possible to control the size and centering of the plots. Additionally, line numbers are supplied for the Python code (in case the user decides to show it). The modifications are very well explained at the beginning of the “MODIFIED” section.

Custom_plot_directive

Below there are 2 examples, shown again with the same code as when explaining the standard plot directive. Here the user can discover that the customized directive gives better control:

```

.. plot:: examples/circleMC.py
   :align: center
   :height: 300
   :include-source:

```

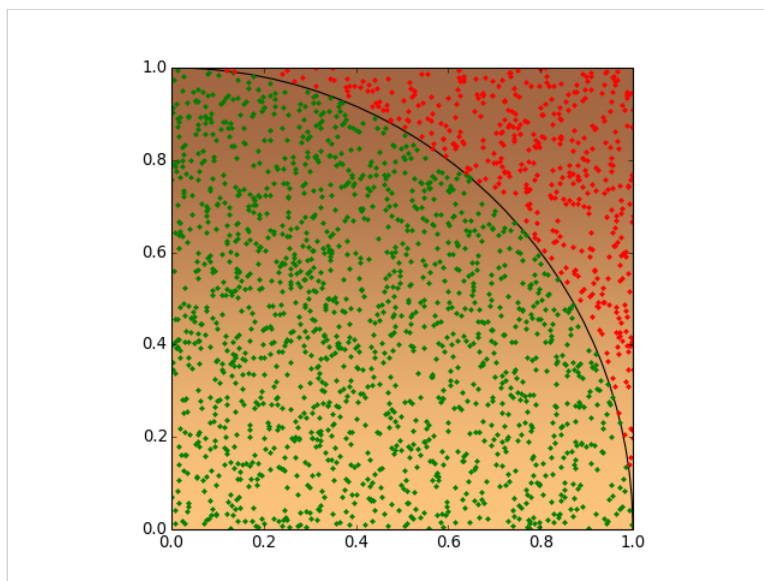


```

1  import numpy as np
2  from matplotlib.pyplot import figure, show
3  from matplotlib.patches import Arc
4  from matplotlib.cm import copper
5
6  fig = figure()
7  frame = fig.add_subplot(1,1,1)
8  n = 2000
9  x = np.random.random_sample(n)
10 y = np.random.random_sample(n)
11 radius = x**2 + y**2
12 In = (radius <= 1.0)
13 Out = (radius > 1.0)
14 frame.plot(x[In], y[In], 'g.')
15 frame.plot(x[Out], y[Out], 'r.')
16 arc = Arc((0,0), 2,2, 0,0, 90)
17 frame.add_patch(arc)
18 frame.imshow([[0, 0],[1,1]], interpolation='bicubic', cmap=copper,
19             vmin=-0.5, vmax=0.5,
20             extent=(frame.get_xlim()[0], frame.get_xlim()[1],
21                   frame.get_ylim()[0], frame.get_ylim()[1]),
22             alpha=1)
23
24 frame.set_xlim(0,1)
25 frame.set_ylim(0,1)
26 frame.set_aspect(1.0)
27 show()

```

([Source code](#), [png](#), [hires.png](#), [pdf](#))



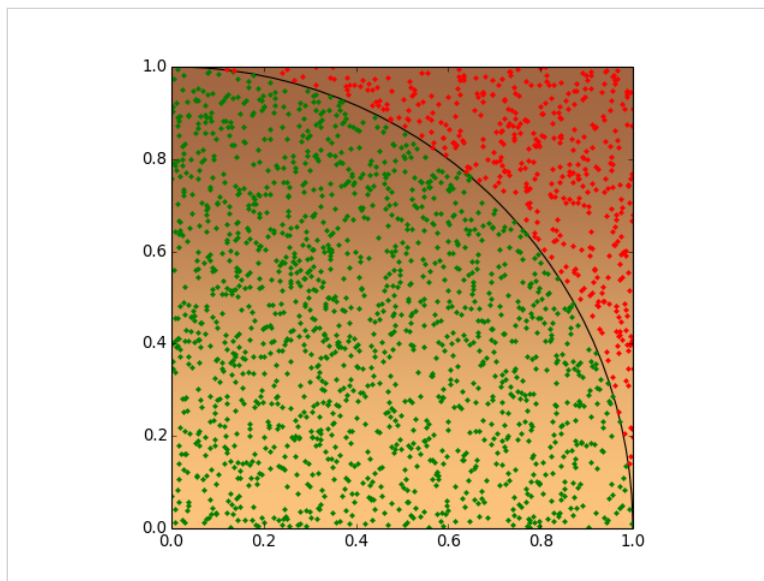
-Ratio of area of a circle and square with a Monte Carlo simulation-

```

.. plot:: examples/circleMC.py
   :align: center
   :height: 300
   :hide-source:

```

([png](#), [hires.png](#), [pdf](#))



-Ratio of area of a circle and square with a Monte Carlo simulation-

The user can see that with the customization of the directive, the link to the source of the plot is suppressed if you use `:hide-source`. Also, centering, height and width are working in combination with each other also.

Custom Question extension

This example extension is used in a Sphinx generated Python course. The role of it is to consequently number the questions in the course by putting the question under the directive:

Question 1:

Question 2:

In every new chapter of the course, the counting could be restarted via the simple option:

```
.. question::
   :number: 1
```

Question 1:

The extension was created by Hans Terlouw and is available at:

[Custom_question_extension](#)

PART III: PROGNUM examples and problems solved

In this section the reader can find examples from the Python programming course which was registered under the name PROGNUM.

1. Raw HTML

One of the first encountered problems was how to migrate very big pieces of HTML code from old documentation- such as a big table or another piece of documentation in HTML that would require a lot of effort.

The directive **raw** can be used then. The syntax is the following:

```
.. raw:: html
   html code
```

Unfortunately, the HTML output would become only visible for the web page version of the documentation and not in the generated PDF. That is why the usage of **raw** was restricted only for linking. Links in Sphinx open in the same page. This is not always what you want. With raw HTML it was possible to open pages in another tab. For this the link needs to be defined before usage. Let's look at the following example:

```

1  .. |timetable| raw:: html
2
3      <a href='http://www.rug.nl/fwn/roosters/2013/vakken/stpce5' target='_blank'>FWN Roster</a>
4
5  |timetable|
6

```

This would result in a link to a Web page with a roster that opens in a new tab: [FWN Roster](http://www.rug.nl/fwn/roosters/2013/vakken/stpce5)

2. Question container

The following output could be produced really easy. It was important to create blocks visible enough to get the attention of the student who wants an overview of the questions in a task. Note that we used option `number` to set the question number!


Question
5:



The text of the question.

It could also be indented:

Question
5:



The text of the question.

There are several tasks to perform before being able to produce the same output:

1. Download the [Custom_question_extension](#)
2. Add the extension to the custom extensions folder (most likely you called it **sphinxext**).
3. Choose a proper image, such as [question01.png](#) and add it to a directory with the pictures you use. In the example above this is a directory called **pictures** under the **source** directory.
4. Create the following style methods in **default.css** or **basic.css**. They are responsible for the appearance:

```

1  .questiontext{
2      display: table-cell;
3      vertical-align: top;
4      color: blue;
5      padding: 1px;
6      width:85%;
7  }
8  .questionimage{
9      display: table-cell;
10     padding: 10px;
11     width: 30px;
12     vertical-align: top;
13 }

```

As the reader can find out, the output is a combination of css and Sphinx output. In order to use the roles in the style sheet called *questionimage* and *questiontext*, we apply the **container** directive. The indentation in the example comes from the fact that we are nesting blocks. The option `:number:` at the beginning is to indicate the initial number from which the counting starts. To prove this, the next example should have number 6:

```

1  .. question::
2
3      .. container:: questionimage
4
5          .. image:: pictures/question01.png
6
7      .. container:: questiontext
8
9          The text of the question.

```

Question 6:



The text of the question.

3. Plotting

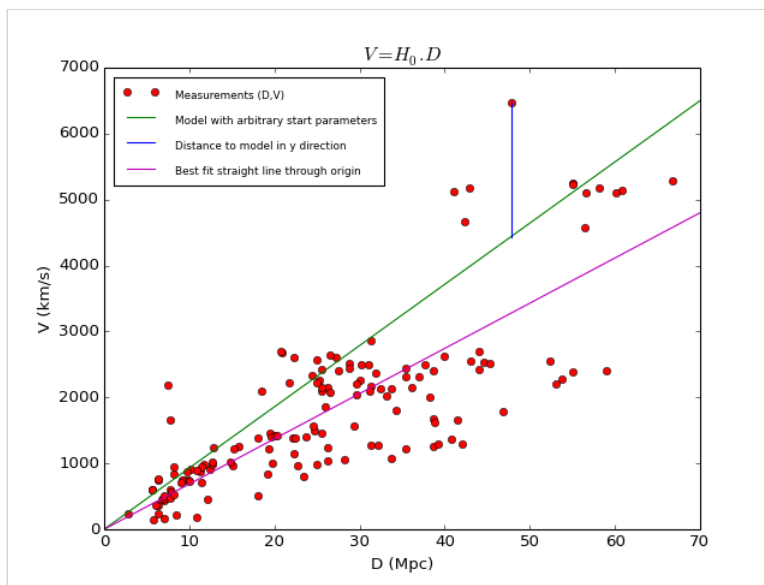
Below, the reader can find several examples of the usage of the plot directive in the PROGNUM course. It is important to note that PROGNUM uses the new custom plot directive ([Custom_plot_directive](#)). Also, all required code should be available in the examples directory:

```

1  .. plot:: examples/hubbleplot.py
2      :align: center
3      :height: 300px
4
5      *- Hubble relation data -*

```

([Source code](#), [png](#), [hires.png](#), [pdf](#))



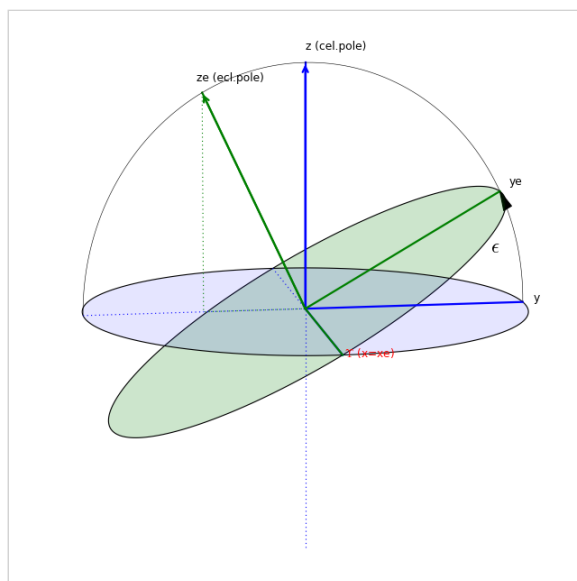
- Hubble relation data -

```

1  .. plot:: examples/rotmatrix_ecliptic.py
2      :height: 300px
3      :align: center
4
5      *- The blue plane is the equatorial system. The green plane is the
6      ecliptic plane after rotation with the transpose of  $E = E_x(23.43)$ . Note that
7      the x axis of the ecliptic also points to the vernal equinox. Note also the
8      difference in orientation with the previous plot.*

```

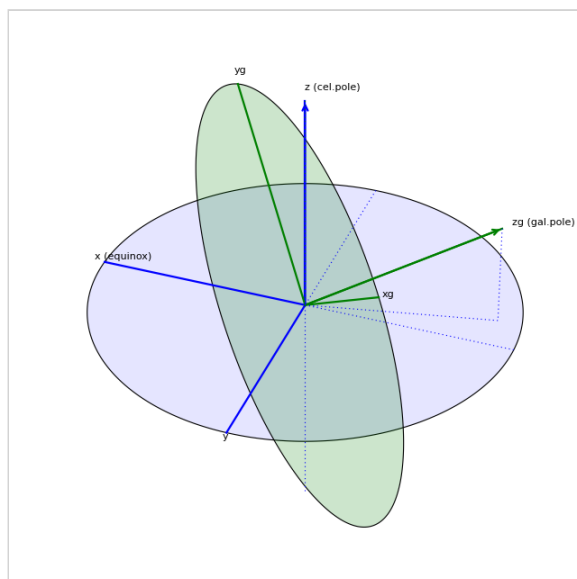
([Source code](#), [png](#), [hires.png](#), [pdf](#))



- The blue plane is the equatorial system. The green plane is the ecliptic plane after rotation with the transpose of $E = E_x(23.43)$. Note that the x axis of the ecliptic also points to the vernal equinox. Note also the difference in orientation with the previous plot.

```
1 .. plot:: examples/rotmatrix_galsys.py
2   :height: 300px
3   :align: center
4
5   *- The blue plane is the equatorial system. The green plane is
6   the galactic plane after rotation with the transpose of
7    $E = E_z(180-123).E_y(90-27.4).E_z(192.25)$ . The galactic pole is
8   at (R.A., Dec) = (192.25, 27.4).*
```

([Source code](#), [png](#), [hires.png](#), [pdf](#))



- The blue plane is the equatorial system. The green plane is the galactic plane after rotation with the transpose of $E = E_z(180-123).E_y(90-27.4).E_z(192.25)$. The galactic pole is at (R.A., Dec) = (192.25, 27.4).

4. Tables

When using tables in Sphinx, it is still possible to have inline markup and linking, as in the following PROGNUM example: Please find the code for the table in the `csv_table.txt` file.

- NumPy functions related to complex numbers -

Function	Description	Example
<code>numpy.array()</code>	Create array with complex numbers. Use dtype 'F', 'G' or 'D'	<code>z = numpy.array([3,2,1,0], dtype='complex64'), z = numpy.array([3,2,1,0], dtype='F')</code>
<code>numpy.ndarray.astype()</code>	Convert float array to complex	<code>z = x.astype('F'), z = x*1j</code>
<code>numpy.complex()</code>	Create a complex number from a real part and an optional imaginary part	<code>z = numpy.complex(3.2)</code>
<code>numpy.real()</code> , <code>numpy.imag()</code>	Create new arrays with only the real or the imaginary part of the complex number(s) z. Note that these new arrays are views on the original array. If you change an element in the original array, then it will also change in the new array with only the real or the imaginary part. If you don't want this behaviour, make an explicit copy.	<code>numpy.real(z), numpy.imag(z), x = numpy.real(z).copy()</code>
<code>numpy.absolute()</code> - <code>numpy.absolute()</code> function (<code>numpy.abs(z)</code>)	Length of z	<code>r = numpy.abs(z)</code>
<code>numpy.angle()</code>	Return the angle of the complex argument. If <code>deg=True</code> , then return angle in degrees	<code>angles = numpy.angle(z,deg=True)</code>
<code>numpy.conj()</code> - <code>conj()</code> function	Return the complex conjugate, element-wise. The complex conjugate of a complex number is obtained by changing the sign of its imaginary part.	<code>zc = z.conjugate() # Creates a copy with changed contents</code>
<code>numpy.iscomplex()</code>	Returns an array with Booleans. True for every complex number.	<code>print numpy.iscomplex(z)</code>

PART IV: Frequent errors

1. Not working external link

The following link is not working: ``Google search<http://www.google.com>`_` because there is no spacing before the `<` character.

The following link is not working: `` Google search <http://www.google.com>`_` because there is spacing after the first ``` character.

The right syntax is: ``Google search <http://www.google.com>`_`

[Google search](#)

The following link will also not work: ``Google search <google.com>`_`. It is crucial to understand that without `http://www.` Sphinx would try to link to `build/html/google.com` which does not exist.

2. Not working csv table

Sometimes it is hard to debug a wrong csv table. The most common error is the following:

```

1  .. csv-table:: a title
2     :header: "first header", "second header", "third header"
3     :widths: 20, 20, 10
4
5     "item 1" , "item 2", 3
6     "item 4", "item 5", 5

```

After "item 1" there is space before the comma. This will cause the error. The right syntax is:

```

1  .. csv-table:: a title
2     :header: "first header", "second header", "third header"
3     :widths: 20, 20, 10
4
5     "item 1", "item 2", 3
6     "item 4", "item 5", 5

```

a title

first header	second header	third header
item 1	item 2	3
item 4	item 5	5

3. Not working math (or other) directive with options

When using the **math** directive or any other directive using options, it is important to look for simple errors. For example the following are common errors:

```

1 .. math:: a^2+b^2= c^2
2   :label : 1
3
4 .. math:: a^2+b^2= c^2
5   :label:1

```

The right syntax is:

```

.. math:: a^2+b^2= c^2
   :label: 1

```

$$a^2 + b^2 = c^2 \quad (5)$$

4. Block that does not contain all the intended text

When using the question or other block and having very long content, it is important to check if every new paragraph starts with the right indentation. The following example would contain only the first paragraph in the question block:: Note that we used option `number` to set the question number!

Question 25:  Paragraph 1. Text Text Text Text Text Text Text Text Text Text ...

Paragraph 2. Text Text Text Text Text Text Text Text

The right indentation is

```

1 .. question::
2
3   .. container:: questionimage
4
5       .. image:: pictures/question01.png
6
7   .. container:: questiontext
8
9       Paragraph 1. Text Text Text Text Text Text Text Text Text Text ...
10
11      Paragraph 2. Text Text Text Text Text Text Text Text

```

Question 26:  Paragraph 1. Text Text Text Text Text Text Text Text Text Text ...
Paragraph 2. Text Text Text Text Text Text Text Text

5. Some inline text errors

When intending to make a piece of text bolded, we use **`**Text**`** to have the following effect: **Text**. The next rows are not correct:

```
** Text**  
**Text **  
** Text **
```

The same stands for the literal blocks which use the ```` markup and look like this: `Text`. The following are incorrect:

```
`` Text``  
``Text``  
`` Text``
```

The only correct syntax is ```Text```.

The problem is analogous for the italic text, using `*`.