Project Documentation

Author

March 2022

TABLE OF CONTENTS

INTRODUCTION

The following is a demonstration of what you can achieve by using Sphinx and reStructuredText to document your projects.

Say that you want to create documentation from the docstrings in your code. **Two conditions**:

- Your project must be a Python **package**
- Your **package** must have a `__version__` global variable

Say that you have a package: it has a main script, and a single `lonely_function`.

Now before we go,

Some tips on writing proper docstrings:

- Use raw strings.
- Write reStructuredText.

CHAPTER

ONE

THIS IS A PACKAGE

It's usually convenient to write a short introduction in your `__init__.py`.

1.1 Main Module

`lonely_function`()

You can add **bold** and *italic* text, text with *math typesetting* and `source code`. You can even define custom markup using CSS/LaTeX to highlight text in unimaginable ways. Leaving that as an exercise for the curious (check *roles*).

- Bullet points work as well

and so does LaTeX math, as you can see in (1.1)!

$$f(n) = \begin{cases} n/2, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases} \tag{1.1}$$

Table syntax is rather weird but manageable.

Table 1.1: Example table.

Header 1	Header 2	Header 3	Header 4
A	B	C	D

You can add pictures if you want too. Importantly, when you are generating documentation from your code:

- Place your figures in the **docs/source/figures directory**.
- Include them with their path relative to the source. That is, **figures/<your figure>.<ext>**.

Unfortunately the figure doesn't nicely fit in this page. Fortunately, we can break page to try and have our discourse flow anyway.

Check it:

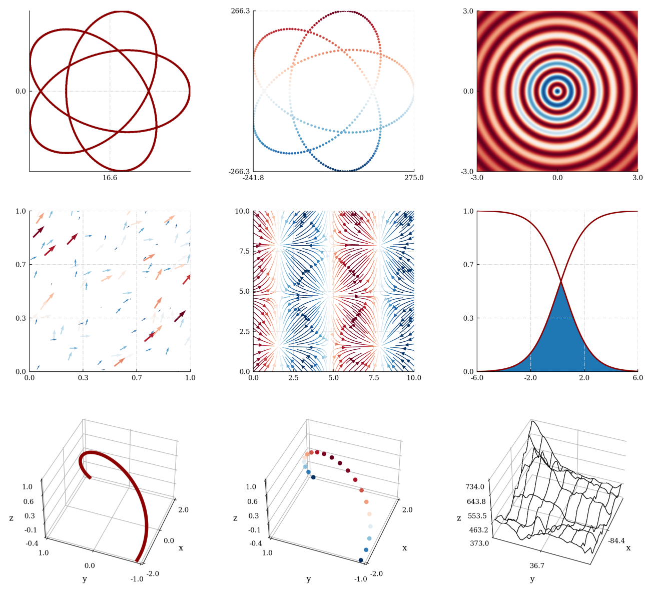

Figure 1.1: You're free to set alignment, figure width relative to the text's, etc. Don't trust Sphinx to go too far. If you need something complex use proper LaTeX code inside a `.. raw:: latex` block.

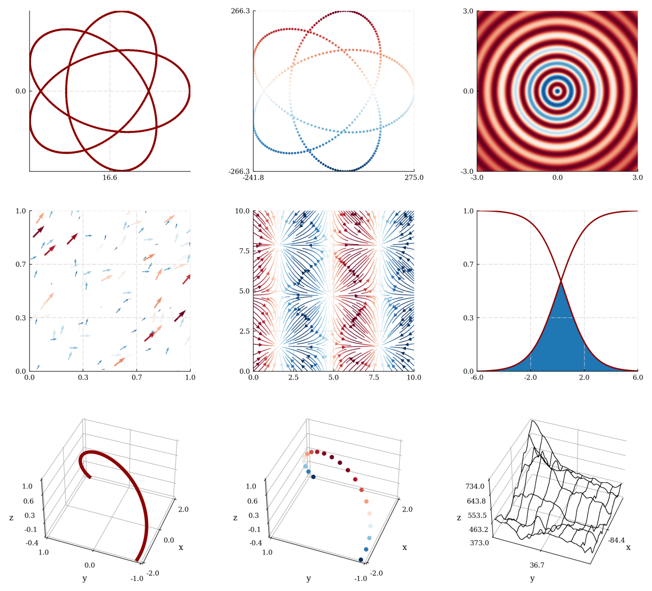

Figure 1.2: Adding a cross-reference to Table 1.1 or Figure 1.1 works much as it would in LaTeX.

1.2 A Subpackage

As before, this package's `__init__.py` is the home of this introduction.

1.2.1 A Module Within

`h``()`
A function which is part of one of the modules of a subpackage of your project.

`g``()`
A second function in the module.

`f``()`
The three functions in this module are stowed in this order:

1. h
2. g
3. f

As you can see, functions are rendered in the order in which you've got them written down. Nice.

1.3 `__init__.py`

`some_API_function``()`
You might wanna keep commonly used utilities or high level interfaces to your package here for ease of import. A way to achieve the same effect is by importing functions defined in your package from this file.

PYTHON MODULE INDEX

INDEX