Fermi Contours

Pablo Piskunow

Nov 17, 2023

CONTENTS

CHAPTER

ONE

REFERENCE

Code adapted from

Perez-Piskunow, Pablo M, Bovenzi, Nicandro, Akhmerov, Anton R, & Breitkreiz, Maxim. (2021). Code and data associated with the paper "Chiral Anomaly Trapped in Weyl Metals: Nonequilibrium Valley Polarization at Zero Magnetic Field" (1.0.1). Zenodo. https://doi.org/10.5281/zenodo.4704325

and extended to take into account periodic boundaries and other edge-cases.

CHAPTER

TWO

REQUIREMENTS

- Python >=3.9

CHAPTER

THREE

INSTALLATION

You can install *Fermi Contours* via pip from PyPI:

```
$ pip install fermi-contours
```

CHAPTER

FOUR

USAGE

Please see the *Command-line Reference* for details.

CHAPTER

FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

SIX

LICENSE

Distributed under the terms of the *MIT license*, *Fermi Contours* is free and open source software.

CHAPTER

SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

EIGHT

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

8.1 Python API

class fermi_contours.marching_squares.**MarchingSquares**(*grid_values=None*, *func=None*, *bounds=None*, *res=None*, *open_contours=True*, *periodic=False*)

Bases: `object`

Marching square class.

Parameters

- **grid_values** (ndarray of floats with shape *(n, m)*, optional) – Values of the function on a 2-dimensional coordinates grid.
- **func** (*callable*) – Function that returns floats for each point in a 2-dimensional coordinates space. If *grid_values* is provided, *func* is used to resolve saddle points. If *grid_values* is not provided, then *func*, *bounds* and *res* must be provided. The *grid_values* is obtained using a 2-dimensional coordinates grid.
- **bounds** (*ndarray-like of shape (2, 2)*) – Bounds in the x- and y-axis *((x_min, x_max), (y_min, y_max))*, optional. If not provided, the bounds are assumed to be *((0, n), (0, m))*.
- **res** (*int, optional*) – Number of linear points to subdivide each of the axis intervals. If bounds is not provided, *res* is not used, and the shape of the *grid_values* is used instead.
- **open_contours** (*bool*) – Wheather to allow open contours or raise an error when contours do not close on themselves.
- **periodic** (*bool, default to 'False'*) – If 'True', the 2-dimensional coordinates grid has periodic boundaries. Thus, the point *(n, j)* is equivalent to *(0, j)*, and *(i, m)* equivalent to *(i, 0)*. If bounds are provided, then *x_max* should be mapped to *x_min*, and *y_max* to *y_min*.

__call__(*level=0*)

Calcualte the Fermi contours for a Fermi level.

Sets values for the attributes defined below.

Parameters
level (*float, default to '0'*) – Isolevel.

Returns
contour_paths – Each list has numerical interpolated points along the path.

Return type
list of lists of pairs of floats.

8.2 Tutorial

8.2.1 Computing contours of a custom function

The function defines a 3d surface, for which we will compute its contours.

We start by importing our library and some plotting functions.

[1]:
```
import numpy as np

from fermi_contours import marching_squares as ms

from matplotlib import pyplot as plt
from matplotlib import colors
from mpl_toolkits.axes_grid1 import make_axes_locatable
```

[2]:
```
def plot_contours(squares, levels, contours, cmap=None, background_cmap="Blues"):
    if cmap is None:
        cmap = plt.cm.Oranges

    fig = plt.figure(figsize=plt.figaspect(2))
    ax1 = fig.add_subplot(2, 1, 1)
    ax2 = fig.add_subplot(2, 1, 2, projection='3d', computed_zorder=False)

    # 2d axis

    n = len(levels)
    color_list = cmap(np.linspace(0, 1, n))
    norm = colors.Normalize(vmin=min(levels), vmax=max(levels))

    divider = make_axes_locatable(ax1)
    cax1 = divider.new_horizontal(size="5%", pad=0.2)
    fig.add_axes(cax1)
    fig.colorbar(plt.cm.ScalarMappable(norm=norm, cmap=cmap), cax=cax1)
    X, Y = np.meshgrid(*squares.grid_points)
    ax1.pcolormesh(X, Y, squares.grid_values.T, cmap=background_cmap);



    for i, contours_per_level in enumerate(contours):
        for contour in contours_per_level:

            ax1.plot(*np.array(contour).T, color=color_list[i])



    # 3d axis
    X, Y = np.meshgrid(*squares.grid_points)
```

(continues on next page)

(continued from previous page)

```
    ax2.plot_surface(X, Y, squares.grid_values.T, alpha=1, cmap=background_cmap)
    ax2.view_init(azim=-80, elev=20)


    for i, (level, contours_per_level) in enumerate(zip(levels, contours)):
        for contour in contours_per_level:
            if len(contour) > 0:
                _x, _y = np.array(contour)[None, :].T
                _z = np.ones(len(_x))[:, None] * level
                ax2.plot(_x, _y, _z, color=color_list[i])

    return fig, ax1, ax2
```

The custom function can be anything that takes two floats and returns another float. If the function is expensive to compute in a serial way, consider computing the `grid_values` elsewhere, and provide the values instead of the function.

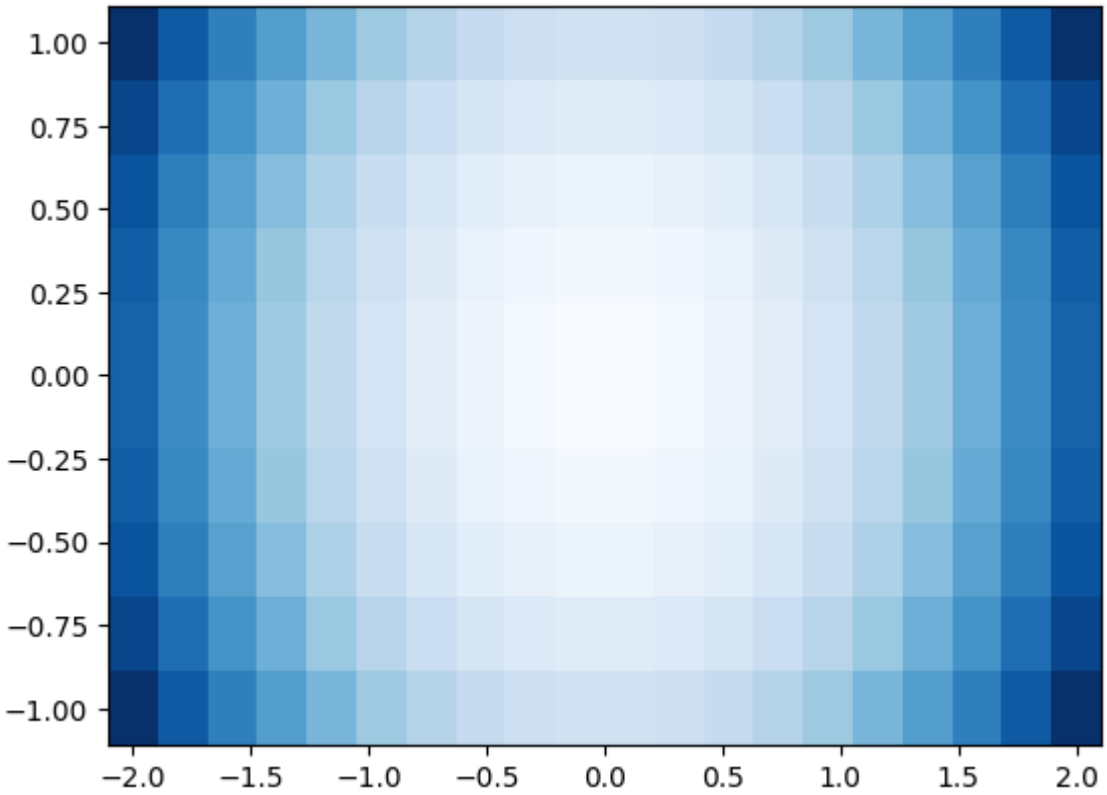
[3]:
```
def surface(x, y):
    return x ** 2 + y ** 2
```

To instantiate our `MarchingSquares` class, we need to provide the bounds of the grid. These bounds will be used to scale the position of the contours.

[4]:
```
squares = ms.MarchingSquares(
    func=surface,
    res=[20, 10],
    bounds=[[-2,2], [-1, 1]],
)
```

At this point, the `squares` instance has extracted the `grid_points` and their `grid_values`. If these are not provided, the `grid_points` are computed them using the `res` and `bounds` parameters, and the `grid_values` are the evaluations of `func` on those points.

[5]:
```
X, Y = np.meshgrid(*squares.grid_points)
plt.pcolormesh(X, Y, squares.grid_values.T, cmap="Blues");
```



Computing contours

Now, we just call our instance with the values of the levels for which we want to compute the contour.

[6]:
```
levels = np.linspace(0, 1, 21)
levels
```

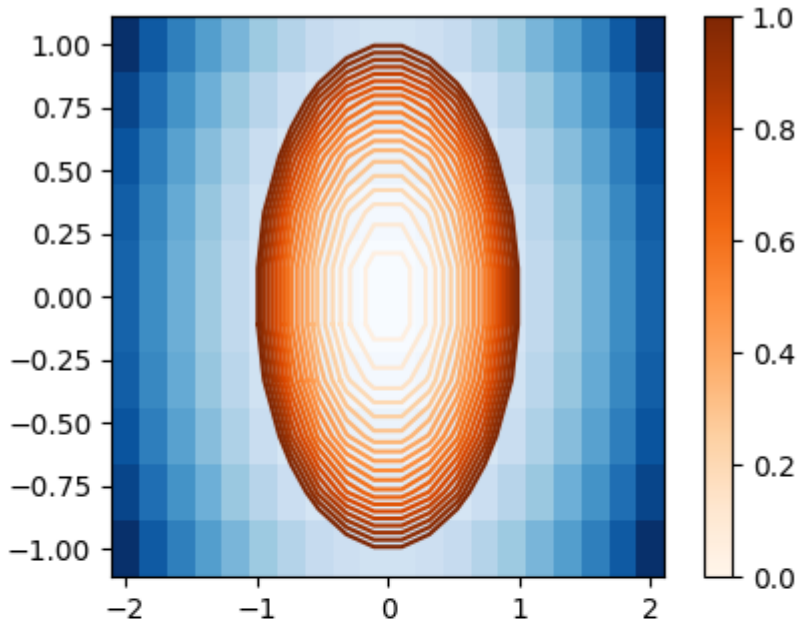
[6]:
```
array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
       0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1.  ])
```

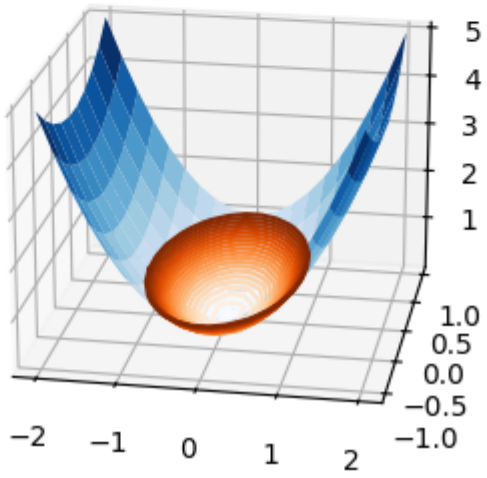
[7]:
```
contours = [squares(l) for l in levels]
```

To visualize the contours, we can plot the `grid_values` in blue, and the set of contours in orange.

[8]:
```
plot_contours(squares, levels=levels, contours=contours)
```

[8]:
```
(<Figure size 400x800 with 3 Axes>, <Axes: >, <Axes3D: >)
```





8.2.2 Special cases

Open contours

It may happen that our contour is not closed, or at least not within the bounds that we want. In this case, we can set the parameter `open_contours=True` to allow this behaviour. This can also lead to more than one contour per level

[9]:
```
levels_open = np.linspace(0, 5, 21)
levels_open
```

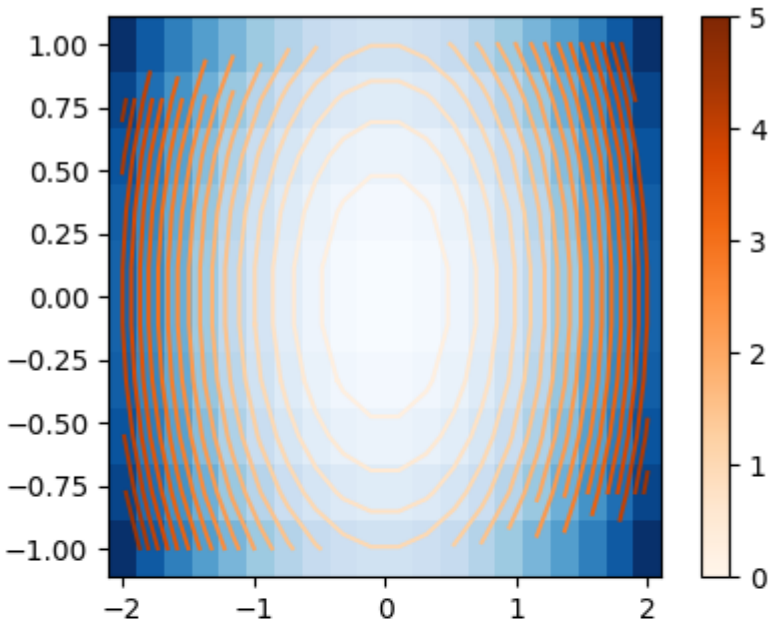
[9]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  , 2.25, 2.5 ,
       2.75, 3.  , 3.25, 3.5 , 3.75, 4.  , 4.25, 4.5 , 4.75, 5.  ])
```

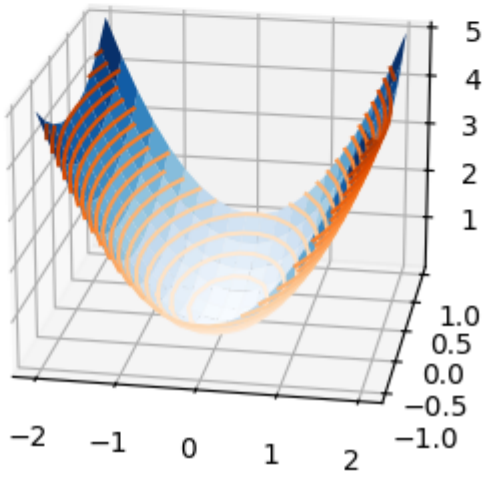
[10]:
```
contours_open = [squares(l) for l in levels_open]
```

To visualize the contours, we can plot the `grid_values` in blue, and the set of contours in orange.

[11]:
```
plot_contours(squares, levels=levels_open, contours=contours_open)
```

[11]:
```
(<Figure size 400x800 with 3 Axes>, <Axes: >, <Axes3D: >)
```





Periodic boundaries

If we want that contours wrap around the edges of our grid, we can set the parameter `periodic=True`.

To make use of this functionality, let's define a periodic function and plot the contours.

[12]:
```
def surface_periodic(x, y):
    period = np.pi
    return np.sin(period * x / 2) + np.sin(period * y )
```

[13]:
```
squares_periodic = ms.MarchingSquares(
    func=surface_periodic,
    res=[20, 10],
    bounds=[[-2,2], [-1, 1]],
    periodic=False,
)
```

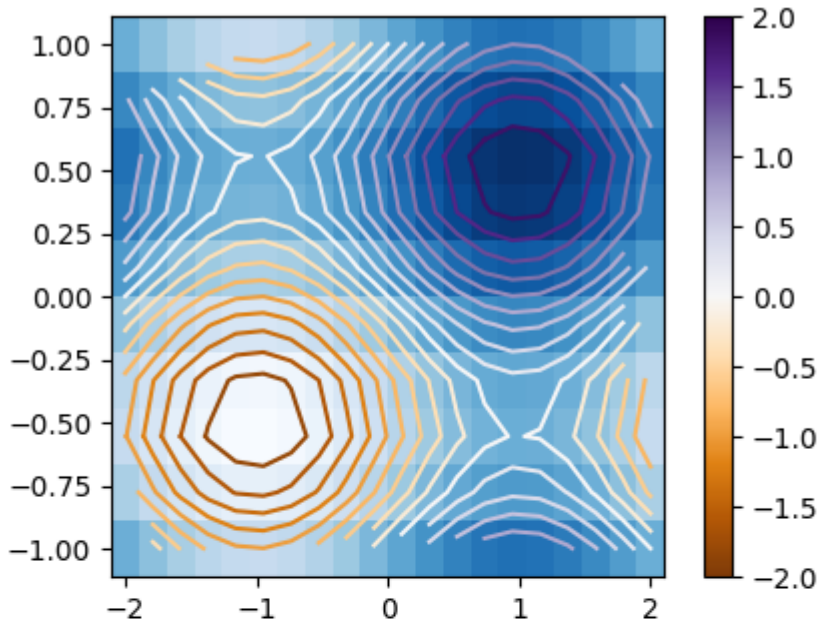
[14]:
```
levels_periodic = np.linspace(-2, 2, 21)
levels_periodic
```

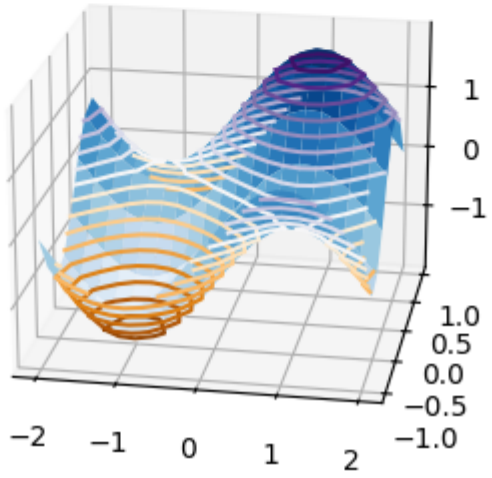
[14]:
```
array([-2. , -1.8, -1.6, -1.4, -1.2, -1. , -0.8, -0.6, -0.4, -0.2,  0. ,
        0.2,  0.4,  0.6,  0.8,  1. ,  1.2,  1.4,  1.6,  1.8,  2. ])
```

[15]:
```
contours_periodic = [squares_periodic(l) for l in levels_periodic]
```

[16]:
```
plot_contours(squares_periodic, levels=levels_periodic, contours=contours_periodic,␣
↪cmap=plt.cm.PuOr)
```

[16]:
```
(<Figure size 400x800 with 3 Axes>, <Axes: >, <Axes3D: >)
```





8.3 Usage

8.3.1 Command line interface

fermi-contours

Fermi Contours.

```
fermi-contours [OPTIONS]
```

Options

`--version`
: Show the version and exit.

8.4 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the [MIT license] and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

8.4.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.4.2 How to request a feature

Request features on the Issue Tracker.

8.4.3 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run fermi-contours
```

8.4.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

8.4.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.5 Contributor Covenant Code of Conduct

8.5.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.5.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.5.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.5.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.5.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at pablo@piskunow.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.5.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.5.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.6 License

INDEX

Symbols

F

M