Deconvoluted Documentation

Release 0.1.1

Martin Roelfs

Jun 05, 2019

Contents:

CHAPTER 1

Deconvoluted

Deconvoluted makes performing numerical integral transforms simple and pythonic!

- Free software: MIT license
- Documentation: https://deconvoluted.readthedocs.io.

1.1 Features

1.1.1 Fourier Transforms

As a first example, let's perform a Fourier transform:

```
t = np.linspace(0, 10, 201)
f = np.sin(3 * 2 * np.pi * t)
F, nu = fourier_transform(f, t)
```

By default, Fourier transforms use Fourier coefficients $a = 0$, $b = -2\pi$. Using another convention is simple:

```
F, omega = fourier_transform(f, t, convention=(-1, 1))
```

As a physicist myself, I therefore switch the labelling of the output from ν for frequency, to ω for angular frequency.

Performing multidimensional transforms is just as easy. For example:

```
F_pq, p, q = fourier_transform(f_xy, x, y)
```

transforms both x and y at the same time. Transforming only one of the two variables can be done simply by setting those that shouldn't transform to `None`:

```
F_py, p = fourier_transform(f_xy, x, None)
F_xq, q = fourier_transform(f_xy, None, y)
```

See the documentation for more examples!

CHAPTER 2

Installation

2.1 Stable release

To install Deconvoluted, run this command in your terminal:

```
$ pip install deconvoluted
```

This is the preferred method to install Deconvoluted, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Deconvoluted can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/tbuli/deconvoluted
```

Or download the tarball:

```
$ curl  -OL https://github.com/tbuli/deconvoluted/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Deconvoluted in a project:

```
import deconvoluted
```

CHAPTER 4

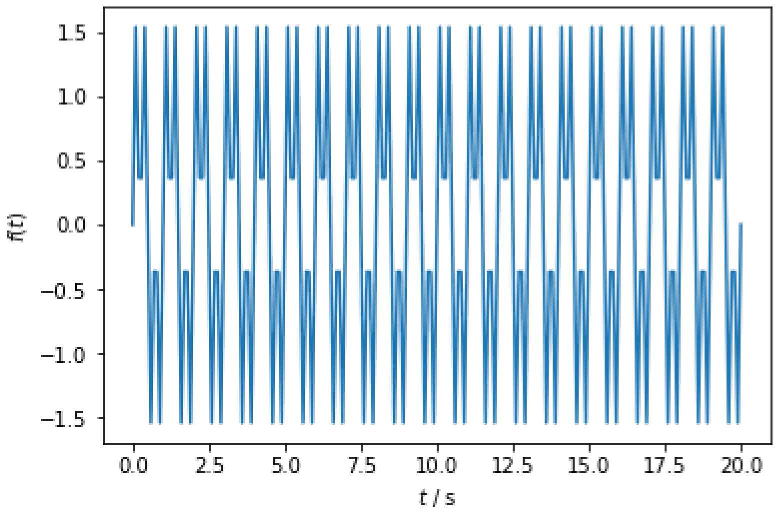
Deconvoluted Examples

4.1 1D Fourier transform

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from deconvoluted import fourier_transform
```

Suppose we want to compute the 1D fourier transform $F(\nu)$ of a function $f(t)$. Let us generate a signal which is a superposition of a signal with $\nu_1 = 1$ Hz and $\nu_2 = 3$ Hz:

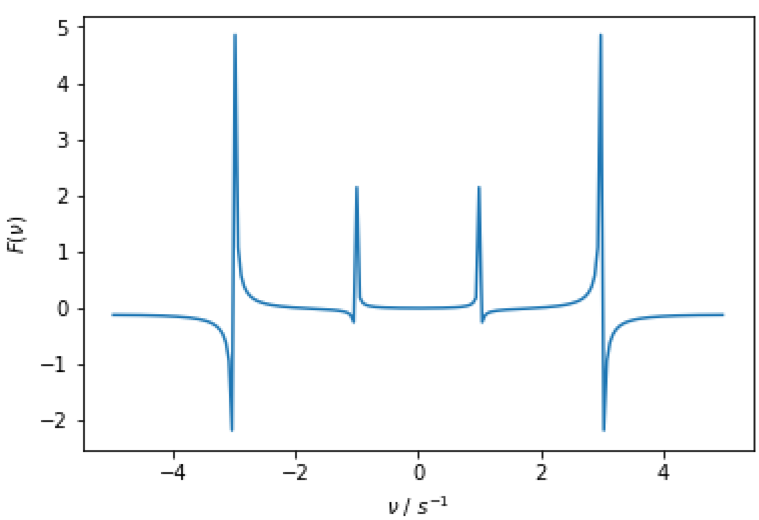
[9]:
```
t = np.linspace(0, 20, 201)  # 20 seconds
nu_1 = 1
nu_2 = 3
f_t = np.sin(nu_1 * 2 * np.pi * t) + np.sin(nu_2 * 2 * np.pi * t)

plt.plot(t, f_t)
plt.xlabel(r'$t$ / s')
plt.ylabel(r'$f(t)$')
plt.show()
```



Taking the transform is now simply a matter of calling `fourier_transform`:

[10]:
```
F_nu, nu = fourier_transform(f_t, t)
```

[11]:
```
plt.plot(nu, F_nu)
plt.xlabel(r'$\nu$ / $s^{-1}$')
plt.ylabel(r'$F(\nu)$')
plt.show()
```

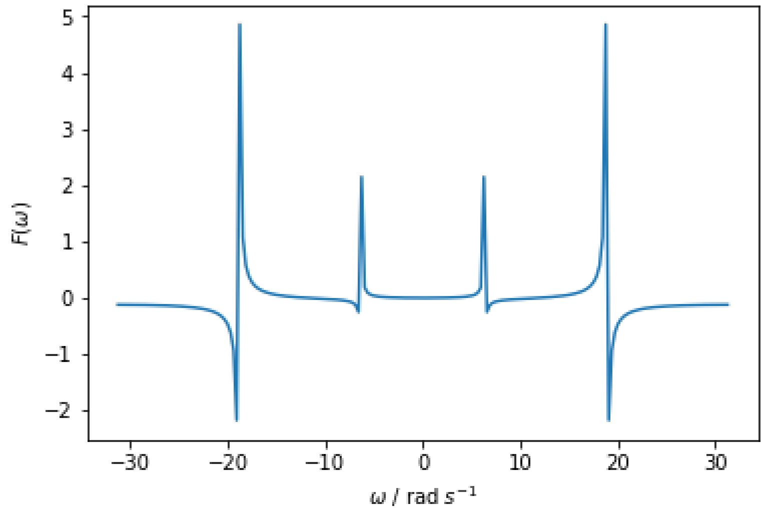


As expected, we find resonances at $\nu_1 = 1$ Hz and $\nu_2 = 3$ Hz.

We could also perform the transform using angular frequency instead:

[12]:
```
F_omega, omega = fourier_transform(f_t, t, convention=(1, -1))
```

[13]:
```
plt.plot(omega, F_omega)
plt.xlabel(r'$\omega$ / rad $s^{-1}$')
plt.ylabel(r'$F(\omega)$')
plt.show()
```



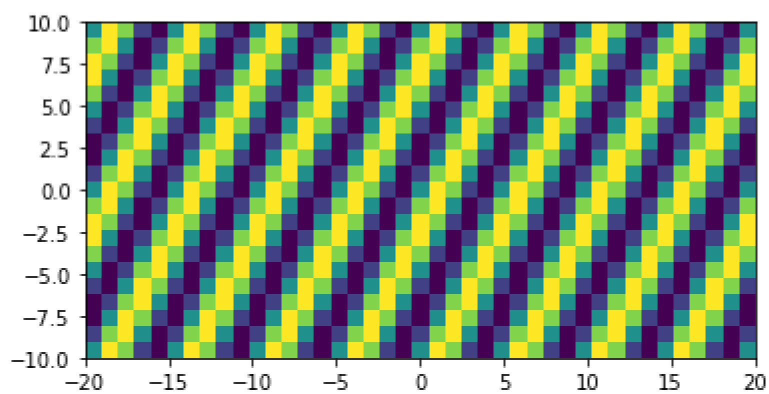
Now our resonances are at $\omega_1 = 2\pi$ and $\omega_2 = 6\pi$ instead.

4.2 2D Fourier transform

[10]:
```
import numpy as np
import matplotlib.pyplot as plt

from deconvoluted import fourier_transform
```

Suppose we want to compute the 2D fourier transform $F(p, q)$ of a function $f(x, y)$. Let us generate some data which has a frequency of 0.2 Hz in the x direction, and 0.1 Hz in the y direction:

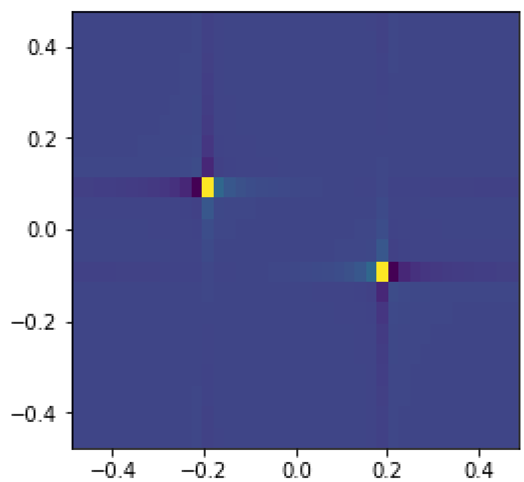
[11]:
```
x = np.linspace(-20, 20, 41)
y = np.linspace(-10, 10, 21)
X, Y = np.meshgrid(x, y)
f_xy = np.sin(0.2 * 2 * np.pi * X + 0.1 * 2 * np.pi * Y)

plt.imshow(f_xy, extent=(x.min(), x.max(), y.min(), y.max()))
plt.show()
```



Taking the transform is now simply a matter of calling `fourier_transform`:

[12]:
```
F_pq, p, q = fourier_transform(f_xy, x, y)
```

[13]:
```
plt.imshow(F_pq.real, extent=(p.min(), p.max(), q.min(), q.max()))
plt.show()
```



We see two resonances, exactly where we would expect them!

CHAPTER 5

Module Documentation

This page contains documentation to every `Deconvoluted` tool.

5.1 deconvoluted.tranforms

`deconvoluted.transforms.`**`determine_axes`**(*f*, **vars*)

Determine the axes along which the FT should be performed.

`deconvoluted.transforms.`**`determine_norm`**(*convention*)

Determine the normalization constant for this `convention`.

Parameters **`convention`** – tuple representing (a, b).

Returns normalization constant.

`deconvoluted.transforms.`**`fourier_transform`**(*f*, **vars*, *convention=Convention(a=0, b=-6.283185307179586)*)

Performs the multidimensional Fourier transform of $f(x_1, \ldots, x_n)$ with respect to any number of variables x_i.

Examples:

```
# 1D transform
F, k = fourier_transform(f, x)

# 2D transform
F_pq, p, q = fourier_transform(f_xy, x, y)

# 2D function, transform only 1 axis
F_py, p = fourier_transform(f_xy, x, None)
```

Parameters

- **`f`** – array representing a function $f(x_1, \ldots, x_n)$

- **vars** – list of x_i w.r.t. which the Fourier transform has to be computed. In case of multi-dimensional functions f the number of `vars` has to match the dimension of `f`. Any axis that should be ignored should be provided as `None`:

```
F_py, p = fourier_transform(f_xy, x, None)
```

- **convention** – The Fourier convention to be used. $a = 0$ and $b = -2\pi$ by default, which is the signal processing standard.

Returns $F(k_1, \ldots, k_n)$, the Fourier transform of $f(x_1, \ldots, x_n)$.

`deconvoluted.transforms.`**`inverse_fourier_transform`**(*F*, **vars*, *convention=Convention(a=0, b=-6.283185307179586)*)

Perform an inverse Fourier transform. See `deconvoluted.transforms.fourier_transform()` for more info.

Parameters

- **F** – Fourier transform $F(k_1, \ldots, k_n)$ of $f(x_1, \ldots, x_n)$.
- **vars** – Any number of k variables or `None`.
- **convention** – The Fourier convention to be used. $a = 0$ and $b = -2\pi$ by default, which is the signal processing standard.

Returns $f(x_1, \ldots, x_n)$, the inverse fourier transform of $F(k_1, \ldots, k_n)$

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/tbuli/deconvoluted/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

6.1.4 Write Documentation

Deconvoluted could always use more documentation, whether as part of the official Deconvoluted docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tbuli/deconvoluted/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *deconvoluted* for local development.

1. Fork the *deconvoluted* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/deconvoluted.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv deconvoluted
$ cd deconvoluted/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 deconvoluted tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6 and 3.7, and for PyPy. Check https://travis-ci.org/tbuli/deconvoluted/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_deconvoluted
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 7

Credits

7.1 Development Lead

- Martin Roelfs <martin.roelfs@kuleuven.be>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 0.1.1 (2019-06-05)

- Implemented support for different FT conventions.

8.2 0.1.0 (2019-06-03)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

D

F

I