

Pythonで学ぶ フーリエ解析と信号処理

博士(理学) 神永 正博 著

コロナ社

まえがき

本書は、フーリエ解析と信号処理の入門書です。本書を読んで得られる知識は、大きく分けて、フーリエ解析の数学的基礎、信号処理の原理と使い方、Pythonによる科学技術計算の基礎、の三つです。1章では、本書で頻繁に使うPythonのライブラリであるNumPyとMatplotlibについて概略を説明します。2章から7章まではフーリエ級数、フーリエ変換の数学的な説明で、8章、9章では、実際の信号の周波数解析を行います。10章はルベグ積分のユーザーズガイドになっています。ゴールは音声データ（wav形式の子猫の鳴き声のデータ）のスペクトログラムを描くことです。これは音声データの時間ごとの周波数情報を表現するもので、短時間フーリエ変換という技術を使って実現できます。ここまでできれば1次元の信号処理の基本はわかったことになり、これを土台にして、より専門的な信号処理を学んでいけるはずです。

本書において、重要なポイントでは数学的に厳密な議論をしています。厳密な数学的議論の大きな利点は二つあります。第一の利点は、(関数解析を基礎とした)より高度な信号処理の理論を学ぶハードルが大きく下がることです。例えば、ウェーブレット解析などを学ぶには厳密な議論を避けて通ることができませんが、本書で学べばスムーズに学習が進められるでしょう。第二の利点は、ハードウェアでの信号処理まで含めた場合に必要となる、アナログ信号に対するフーリエ解析が理解できるようになることです。本書ではデジタル信号処理を扱いますが、フーリエ解析部分は、アナログ信号処理の基礎にもなっているのです。

二十年近く、工学部でフーリエ解析を教えてきました。その間、さまざまな教科書を使ってきましたが、いずれも偏微分方程式への応用が中心で、信号処理への応用にはあまり言及されていないものでした。しかし近年、電気系、情報系では、偏微分方程式への応用もさることながら、それ以上に信号処理への応用が求められています。信号処理では、時間とともに変化する信号を周波数で見るのが重要になります。しかし、偏微分方程式への応用では「周波数」の物理的意味については副次的にしか扱われません。

信号処理を教えるため、既存の信号処理の教科書に目を通してみると、実践的に書かれたものでは数学的な厳密さが犠牲に、厳密に書かれたものでは実践が犠牲になりがちな傾向にありました。

このようなことが起きるのは、実際の応用では、デジタル信号処理をする前の測定段階で、ハードウェアによって高周波成分がカットされており、そのような帯域制限された信号を離散フーリエ変換すればよいからです。デジタル信号処理で必要になる離散フーリエ変換は有限和ですから、極限操作は自由にでき、連続信号を扱う際に生じる極限操作の問題が前面に出ないからです。しかし、離散フーリエ変換は連続フーリエ変換の近似ですので、連続の問題につ

いても正確な議論をすべきだと思いました。

これらの観点から、講義を数年がかりで信号処理向きに修正し、その際に作成した講義ノートが本書のもとになっています。10章には、ルベグ積分のユーザーズガイドがあります。ルベグ積分は数学科以外ではほとんど教えられていないために、不当に無駄なものだと思われるようですが、ごまかしなく議論するには必要な数学です。使うだけならそう難しいものではありませんので、是非この機会に読んでみてください。本書がより発展的なIT・数学を学ぶための基盤となれば、望外の喜びです。

執筆に際し、京都工芸繊維大学の峯拓矢先生、東北学院大学の鈴木利則先生、同じく東北学院大学の深瀬道晴先生に査読いただきました。記して感謝いたします。

2020年7月

神永 正博

目 次

1. Python と便利なライブラリたち

1.1 本書でよく使うライブラリ	1
1.2 NumPy の n 次元配列 (ndarray)	2
1.3 関数のベクトル化	6
1.4 Matplotlib	7
章 末 問 題	11

2. フーリエ級数展開

2.1 関数を級数展開することで何がわかるのか?	13
2.2 周波数情報を取り出す	14
2.3 Python でフーリエ級数部分和を見てみよう	17
2.4 連続でない関数のフーリエ展開の例	20
2.5 半区間展開	23
章 末 問 題	25

3. 関数の直交性

3.1 関数の世界に内積を導入する	27
3.2 シュヴァルツの不等式と三角不等式	29
3.3 フーリエ係数と内積	32
3.4 ベッセルの不等式・パーセバルの等式	33
3.5 フーリエ係数の最適性とベッセルの不等式	34
3.6 その他の直交関数系	39
章 末 問 題	40

4. ギブス現象と総和法

4.1 Python でギブス現象を見てみよう	42
-------------------------------	----

4.2 ひげが残り続けること	43
4.3 チェザロ総和法	45
章 末 問 題	48

5. 複素フーリエ級数

5.1 実フーリエ級数を見直す	50
5.2 実例を見てみよう	52
5.3 振幅スペクトル・パワースペクトル・位相スペクトル	53
5.4 関数の滑らかさと複素フーリエ係数の関係	55
章 末 問 題	58

6. フーリエ変換

6.1 フーリエ変換の導入	59
6.2 フーリエ変換の基本的な性質	62
6.3 フーリエ逆変換	63
6.4 フーリエ変換・逆変換の例	65
章 末 問 題	67

7. フーリエ変換の諸性質

7.1 L^2 条 件	69
7.2 畳 込 み	72
7.3 相互相関関数・自己相関関数	76
7.4 フーリエ変換の減衰オーダーと滑らかさ	78
章 末 問 題	79

8. Python で FFT

8.1 サンプリング定理	81
8.2 離散フーリエ変換	84
8.3 Python を使って周波数情報を取り出す	87
8.4 FFT のアルゴリズム	93

8.5 あえて Python で FFT を作る	96
章 末 問 題	99

9. Python でスペクトログラム

9.1 窓 関 数	100
9.2 窓関数を掛けた信号の FFT	104
9.3 窓関数の周波数特性の見方	107
9.4 SciPy のカイザー窓を見てみよう	109
9.5 短時間フーリエ変換と音声データの解析	112
9.6 wav ファイルのスペクトログラム	113
章 末 問 題	117

10. ルベグ積分ユーザーズガイド

10.1 本 章 の 方 針	118
10.2 「ほとんどいたるところ」ってどういう意味?	118
10.3 積分の定義と役に立つ極限定理	122
10.4 リーマン=ルベグの補題	129
10.5 積分の順序交換	130
章 末 問 題	131
引用・参考文献	133
章末問題略解	134
索 引	155

Python と便利なライブラリたち

本章では、Python のライブラリのうち、本書で頻繁に利用する NumPy と Matplotlib について基本的なことを解説します。なお、本書では、この二つのライブラリだけでなく、ほかにもいくつかのライブラリを利用しますが、全体に関わるのはこの二つですので、ここでまとめておきます。Python をあまり使ったことがなければ、ここから読み始めるのがよいでしょう。数学的な内容を先に知りたい方は、2章から読み始め、プログラムに関してわからないことがあったら本章に戻ってきてもよいかと思えます。

1.1 本書でよく使うライブラリ

本書で使う Python はプログラミング言語の一つですが、使っている印象では、プログラミング言語というより、他の（一般に高速な）言語で書かれたライブラリを活用するためのインタフェースのように感じられます。Python は科学技術計算のライブラリが豊富で、利用する際に数学と無関係なことを考えなくてよいというところが大きな利点です。ここでは、本書で利用する代表的なライブラリと、その概略を説明します。

本書では Python 3.6（以降）と必要なライブラリがインストールされていることを前提にしています（互換性のない Python 2.x については対応していませんのでご注意ください）。

Python のインストールについては、いくつかの方法がありますが、ウェブ上に豊富な情報がありますので、そちらを見ていただいたほうがよいでしょう。本書では、Anaconda をインストールして標準で使える Spyder を利用しています（Python 3.7.5, Spyder 4.0.0 で動作を確認しています）。Anaconda をインストールした場合は、すでに以下の三つのライブラリがインストールされているはずです。もちろん、独力でインストールしてもかまいません。本書で使用されているサンプルコードは、<https://www.coronasha.co.jp/np/isbn/9784339009378/> からダウンロードできます。

本書で頻繁に利用するライブラリは以下の三つです。

- **NumPy**：本書で中心となるライブラリです。線形代数（行列やベクトルの計算）、 n 次元配列（`ndarray`）に対する数学関数の計算、高速フーリエ変換など広範な科学技術計算を提供してくれます。
- **SciPy**：本書では、信号処理や最適化計算のために必要なライブラリです。NumPy ベー

スの科学技術計算ライブラリで、使いやすいインタフェースにラッピングされています。

- **Matplotlib** : グラフを描画するための関数を提供してくれるライブラリです。

NumPy と Matplotlib は、本書で頻繁に使われますので、ここで少し立ち入って説明しておくことにしましょう。SciPy については使うときに適宜説明することにします。まずは、NumPy から説明しましょう。

1.2 NumPy の n 次元配列 (ndarray)

Python で配列を扱う場合、リストにする方法と ndarray にする方法の二つがあります。リスト化は手軽な方法ですが、処理が遅く、高速な処理が必要な科学技術計算には向いていません。そのため、Python で科学技術計算をする場合、ndarray というデータ型にして、NumPy という高速なライブラリを利用して処理を行うことが多いのです。NumPy は Python より高速な言語で記述されているため、Python 自体で計算するよりもはるかに高速な処理が可能です。

ndarray は単に配列と呼ばれることもあります。以下、単に配列といえば ndarray のことを指します。ndarray は、 n -dimensional array (n 次元配列) という意味です。1次元の配列はリストと大体同じようなものですが、リストでは異なる型が混ざっていてもよいのに対し、ndarray では許されません。ndarray では、すべての要素が同じ型である必要があるのです。IPython で少し様子を見てみましょう。IPython は、対話型の Python インタフェースで、プログラムを組むほどでないちょっとしたことを試すのに便利です。例えば、 $\sin \frac{\pi}{4}$, $\sin \frac{\pi}{3}$, $\sin \frac{\pi}{6}$ を一度に計算させることを考えます。NumPy を使う場合は、このようにします。

```
In [1]: import numpy as np
In [2]: PI = np.pi
In [3]: x = np.array([PI/4,PI/3,PI/6])
In [4]: np.sin(x)
Out[4]: array([0.70710678, 0.8660254 , 0.5      ])
```

1行目で `numpy` を `np` という名前をつけてインポートしています。これは広く使われている略記法ですので、本書でもこれになります。2行目で `numpy` 用の π に `PI` という名前をつけました。3行目では、 $(\pi/4, \pi/3, \pi/6)$ という1次元配列 (ndarray 型のデータ) を作っています。4行目では、この `x=(x[0], x[1], x[2])` という配列に対して、`[sin(x[0]), sin(x[1]), sin(x[2])]` の値を計算しています。配列の要素ごとに正弦の値を求めて並べたものが出力されていることがわかるでしょう。続けて、`x` の要素を一気に3倍してみましょう。つぎのように書くだけです。ベクトルと同じです。

```
In [5]: 3*x
Out[5]: array([2.35619449, 3.14159265, 1.57079633])
```

同じ型の配列なら、ベクトルのように足すこともできます。

```
In [6]: y = np.array([1, 2, 3])
```



```
In [7]: z = np.array([3, 8, 9])
In [8]: y+z
Out[8]: array([ 4, 10, 12])
```

乗算やべき乗の計算もできます。例えば、 t を `ndarray` として $t^2 + 2t + 3$ とするとつぎのようになります。Python では、 a^b は、`a**b` で表します。

```
In [9]: t = np.array([1, 2, 3, 4])
In [10]: t**2+2*t+3
Out[10]: array([ 6, 11, 18, 27])
```

このように成分ごとに計算してくれるわけです。

`ndarray` では複素数を扱うこともできます。フーリエ解析ではオイラーの公式 ($e^{i\theta} = \cos \theta + i \sin \theta$) を経由して複素数成分を持つ配列を扱いますのでここで見ておきましょう。Python では虚数単位を `j` で表現します。これは電気工学などで一般的に使われる記法です。電気工学では、電流を i で表現するため、 i を虚数単位としては使うのが都合が悪かったということのようです。本書において数学的記述では、数学の慣習に従って虚数単位を i で表しますが、Python での表現は `j` になるのでご注意ください。

例えば、 $z = (-1 + 2i, 2.3 + 3.5i, -3.7 + 0.1i)$ というベクトル (配列) はつぎのように表現します[†]。

```
In [11]: z = np.array([-1.0+2.0j, 2.3+3.5j, -3.7+0.1j])
```

もちろん、複素数演算もできます。例えば、 $(-2.1 + 3.3i)z$ と、 $\exp(z) = (\exp(z[0]), \exp(z[1]), \exp(z[2]))$ は、つぎのように計算できます。もちろん、複素数の指数関数は、 $z = x + iy$ (x, y は実数) に対し、 $\exp(z) = \exp(x) \exp(iy) = e^x (\cos y + i \sin y)$ と解釈されます。

```
In [12]: (-2.1+3.3j)*z
Out[12]: array([-4.5 -7.5j, -16.38 +0.24j, 7.44-12.42j])
In [13]: np.exp(z)
Out[13]: array([-0.15309187+3.34511829e-01j, -9.34038986-3.49877592e+00j,
0.02460001+2.46823412e-03j])
```

本書では、`np.linspace` 関数をよく使います。`np.linspace` は、等差数列 (配列) を生成する関数です。本書で扱う `np.linspace` 関数の引数は、以下の `start`, `stop`, `num` の三つです (引数はこのほかにもありますが、本書では触れません)。

```
numpy.linspace(start, stop, num = 50)
```

ここにおける、引数 `start` は始点 (初項) で、`stop` は数列の終点 (末項) です。いずれも `int` 型 (整数型) または `float` 型 (浮動小数点型) です。`num` という引数は、生成する配列 (`ndarray`) の要素の数で、デフォルト値は 50 に設定されています。例えば

```
t = np.linspace(-PI, PI, 10000)
```

とすると、`t` は、 $-\pi$ を始点 ($t[0] = -\pi$) とし、 π を終点 ($t[9999] = \pi$) とした等差数列

$$t[j] = -\pi + \frac{2\pi j}{9999} \quad (j = 0, 1, \dots, 9999)$$

[†] `dtype` メソッドを使う (`z.dtype` とする) とその型 (プラットフォームに依存) が表示されます。筆者の環境では、`dtype('complex128')` と出ます。

になります (もちろん, 要素の数は 10000 です)。一般に, 公差は

$$\frac{\text{stop} - \text{start}}{\text{num} - 1}$$

になります。分母が `num - 1` になっているのは, デフォルトで, `endpoint = True` になっているためです。これは終点が含まれるという意味です。もし, 終点を含まないようにしたければ, `endpoint = False` とします。このとき, 公差は

$$\frac{\text{stop} - \text{start}}{\text{num}}$$

になります。本書ではデフォルトのまま使いますので, 以下, `endpoint` については気にしないことにします。

紛らわしいのですが, NumPy で等差数列を作る方法がもう一つあります。`arange` 関数を使う方法です。`linspace` 関数とほとんど同じですが, 微妙な違いがあります。`arange` 関数では, (デフォルトでということですが) 終点の値を含みません。`linspace` 関数では (デフォルトでは) 終点の値を含みます。また, `arange` 関数では, 初項を指定しなくてもよい場合があります。例えば

```
In [14]: np.arange(10)
Out[14]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

のように, 初項は 0 で, 公差 1, 終点の値を含まない 10 項からなる等差数列が得られます。`linspace` 関数と同様に, 始点と終点, 公差を変更することもできます。例えば, 始点 (初項) 2 で公差 0.2, 3 未満の等差数列を作るには, つぎのようにします。`linspace` 関数ではいくつに刻むかを指定するのに対し, `arange` 関数では, 公差を指定する点に違いがあります。ただし, `arange` 関数では浮動小数点演算で終点を含んだり含まなかったりすることが報告されています[†]。

```
In [15]: np.arange(2, 3, 0.2)
Out[15]: array([2. , 2.2, 2.4, 2.6, 2.8])
```

本書ではあまり使いませんが (まったく使わないわけではありません), 一般の n 次元配列に対して, 行列を扱うこともできます。1 次元配列を 3 次元配列 (行列) に変換するには, `reshape` メソッドを使います。このようになります。

```
In [16]: a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
In [17]: A = a.reshape(3,3)
In [18]: A
Out[18]:
array([[1, 2, 3],
```

[†] teratail の質問 (投稿: 2017/12/17 14:53, 編集 2017/12/17 14:59) 「python numpy における arange 関数のエラー」 (<https://teratail.com/questions/105198>) で, `print(np.arange(0.01,0.08,0.01))` とすると, `[0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08]` というように終点が含まれることが報告されています。これは, $(0.08-0.01)/0.01$ が浮動小数点計算の精度不足により 7 にならず, `7.00000000000000088817841970012523233890533447265625` となってしまう, これを天井関数で繰り返して 8 になってしまうことによります (mkgrei 氏のコメント (2017/12/17 15:49))。

```
[4, 5, 6],
 [7, 8, 9]])
```

ndarray のデータの型 (各次元ごとの要素の数) を知るには, `shape` (インスタンス変数というものですが, ここでは深い意味を知る必要はありません) を使います。例えば, いま定義した, `a` と `A` の場合は, このようになります。

```
In [19]: a.shape
Out[19]: (9,)
In [20]: A.shape
Out[20]: (3, 3)
```

おのおのの型が表示されていることがわかるでしょう。`a.shape[0]` は長さ 9 のベクトルにあたり, `A.shape[0]` は行の数 3, `A.shape[1]` は列の数です (同じですが)。行と列の値が異なる場合を見てみましょう。例えば, このようになります。

```
In [21]: b = np.array([1, 2, 3, 4, 5, 6])
In [22]: B = b.reshape(2,3)
In [23]: B
Out[23]:
array([[1, 2, 3],
       [4, 5, 6]])
In [24]: B.shape
Out[24]: (2, 3)
```

単に `b` の要素数を求めるときに, `b.shape[0]` のような書き方をすることも多いです (本書でも一部で使っています)。

行列 (とベクトル) の計算もできます。例えば

$$A = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 3 & 5 \end{pmatrix}, \quad B = \begin{pmatrix} -2 & 3 \\ 1 & -1 \\ 2 & -3 \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix}$$

に対し, $C = AB$, $\mathbf{v} = A\mathbf{u}$ を計算するには, リスト 1.1 のようにすれば問題ありません。行列の積 ST が定義できるためには, S の列数と T の行数が一致していなければならないことに注意しましょう。

リスト 1.1 (matrixprod.py)

```
1 import numpy as np
2
3 A = np.array([[1, 2, 3], [-1, 3, 5]])
4 B = np.array([[ -2, 3], [1, -1], [2, -3]])
5 u = np.array([1, 2, 5]).reshape(3,1)
6
7 C = np.dot(A, B)
8 v = np.dot(A, u)
9 print(C)
10 print(v)
```

索引

【い, え, お】	【し】	
位相スペクトル 53, 61	自己相関関数 76	半区間展開 23
エネルギースペクトル 53	シュヴァルツの不等式 29	半値幅 108
エリアシング 90	周波数 14	ハン窓 104
オーバーシュート 42	振幅スペクトル 53, 61	【ひ】
【か】	振幅スペクトル密度 61	標本化定理 81
カイザー窓 109	【せ, そ】	【ふ】
外測度 120	正規直交基底 34	ファトゥーの補題 125
可算無限個 119	相互相関関数 76	フェイエール核 48
可積分 60, 124	【た】	フーリエ級数 17
可聴周波数 15	帯域制限 81	フーリエ級数部分和 17
カーディナル・サイン関数 60	高々可算個 120	フーリエ係数 17
関数空間 29	畳込み 72	フーリエ正弦展開 23
完全正規直交系 34	単関数 123	フーリエ展開 17
カントールの三進集合 119	短時間フーリエ変換 112	フーリエ余弦展開 23
完備性 127	単調収束定理 125	【へ】
【き】	単調性 121	ベイリー＝ウィーナーの定理 58
ギブス現象 43	【ち, て】	ベッセルの不等式 34
基本角周波数 14	チェザロ総和法 45	ヘルツ 14
基本周波数 14	チェビシェフ多項式 41	【ま, め】
極化恒等式 70	定義関数 123	窓関数 101
【く, こ】	ディリクレ核 75	窓を掛ける 101
矩形窓 101	【な】	メインローブ 107
区分求積法 28	ナイキスト周波数 82	【ゆ】
合成積 72	【は】	優収束定理 125
コーシー列 128	パーセバルの等式 34	【ら, り, る, れ】
【さ】	バタフライ演算 95	ラジアン毎秒 14
サイドローブ 107	バナッハ＝タルスキーの パラドックス 122	ラムダ式 19
サイドローブレベル 107	ハニング窓 104	リップル 42
三角不等式 29	ハミング窓 104	リーマン＝ルベーグの補題 22
サンプリング周波数 82	パワースペクトル 53	ルベーグ測度 121
サンプリング定理 81	パワースペクトル密度 61	ルベーグの収束定理 125
【その他】		零集合 119
L^1 条件 60, 124	L^2 空間 29	劣加法性 121
	L^2 条件 29	
	L^2 ノルム 27	
		3dB 帯域幅 108

—— 著者略歴 ——

1991年 東京理科大学理学部数学科卒業
1993年 京都大学大学院理学研究科修士課程修了（数学専攻）
1994年 京都大学大学院理学研究科博士課程中退（数学専攻）
1994年 東京電機大学助手
1998年 株式会社日立製作所勤務
2003年 博士（理学）（大阪大学）
2004年 東北学院大学講師
2005年 東北学院大学助教授
2007年 東北学院大学准教授
2011年 東北学院大学教授
現在に至る

Python で学ぶフーリエ解析と信号処理

Fourier Analysis and Signal Processing with Python © Masahiro Kaminaga 2020

2020年9月28日 初版第1刷発行

★

検印省略

著者 神 永 正 博
発行者 株式会社 コロナ社
代表者 牛来真也
印刷所 三美印刷株式会社
製本所 有限会社 愛千製本所

112-0011 東京都文京区千石 4-46-10

発行所 株式会社 コロナ社
CORONA PUBLISHING CO., LTD.

Tokyo Japan

振替 00140-8-14844 · 電話 (03) 3941-3131(代)

ホームページ <https://www.coronasha.co.jp>

ISBN 978-4-339-00937-8 C3055 Printed in Japan

(齋藤)



＜出版者著作権管理機構 委託出版物＞

本書の無断複製は著作権法上での例外を除き禁じられています。複製される場合は、そのつど事前に、出版者著作権管理機構（電話 03-5244-5088, FAX 03-5244-5089, e-mail: info@jcopy.or.jp）の許諾を得てください。

本書のコピー、スキャン、デジタル化等の無断複製・転載は著作権法上での例外を除き禁じられています。購入者以外の第三者による本書の電子データ化及び電子書籍化は、いかなる場合も認めていません。落丁・乱丁はお取替えいたします。