君は cmath を知っているか

Hayao Suzuki

PyCon mini Shizuoka 2020

February 29, 2020

Contents

自己紹介

お前誰よ

- **名前** Hayao Suzuki（鈴木　駿）
- **Twitter** @CardinalXaro
- **ブログ** `https://xaro.hatenablog.jp/`
- **専門** 数学 (組合せ論・グラフ理論)
- **学位** 修士（工学）、電気通信大学
- **仕事** 株式会社アイリッジ
  - スマートフォンアプリのバックエンドサーバーの開発

自己紹介

技術書の査読

- 『Effective Python』（オライリージャパン）
- 『エレガントな SciPy』（オライリージャパン）
- 『データサイエンス設計マニュアル』（オライリージャパン）など
- https://xaro.hatenablog.jp/ に一覧あります。

いろんな発表

- 「SymPy による数式処理」（PyCon JP 2018）
- 「Python で楽しむ初等整数論」（PyCon mini Hiroshima 2019）など
- https://xaro.hatenablog.jp/ に一覧あります。

今日の発表

バッテリー同梱哲学（PEP 206 より）

Python ディストリビューション自身が、別途ダウンロードすることなくすぐに利用できる豊富で汎用性の高い標準ライブラリを持つこと。

Python チュートリアルで紹介されている例

- `xmlrpc.client` XML-RPC クライアント
- `xmlrpc.server` XML-RPC サーバー
- `email` 電子メールと MIME 処理のためのパッケージ
- `json` JSON エンコーダおよびデコーダ
- `sqlite3` SQLite データベースに対する DB-API 2.0 インタフェース

今日の発表

cmath とは何者か

1. C 言語で実装された高速な `math` ライブラリ
2. キュウリ (Cucumber) の画像識別のための数学ライブラリ
3. 複素数 (Complex Number) の計算ライブラリ

今日の発表

`cmath` モジュール

- 複素数のための数学関数
- 9V 電池やらニカド電池のような存在に今、スポットを当てる。

今回使うもの

- Python 3.7.x（Python 3.8.x でも楽しめます！）
- Matplotlib（グラフ描画ライブラリ）

今日の発表

君は `cmath` を知っているか

- `cmath` とは何者か
- 複素数とは何か
- 複素数の極座標表記
- 複素指数函数
- 離散 Fourier 変換
- 1 の n 乗根
- まとめ

資料は設計図共有サイトにある！

資料はすべて
`https://github.com/HayaoSuzuki/PyCon-mini-Shizuoka-2020/` に
あります。

複素数とは

複素数、知ってますか？

複素数の定義

定義（複素数）

$i^2 = -1$ であるような基底 $1, i$ を持つ実数体 $\mathbf{R}$ 上の 2 次元ベクトル空間の元を複素数と呼ぶ。また、i を虚数単位と呼ぶ。

Python で複素数を定義する

```
>>> 3 + 5j # Python では虚数単位を j または J とする
(3+5j)
>>> 1J**2 # 虚数単位の自乗は-1 となる。
(-1+0j)
>>> 4 + 5j == (5j + 4) # 実部と虚部がそれぞれ等しい
True
```

複素数と体

体 (Field) == 四則演算ができる集合

複素数は複素数体 $\mathbf{C}$ をなす。

Python における複素数の四則演算

```
>>> 8 - 5j + -5 + 1j # 加法
(3-4j)
>>> (1 + 2j) * (1 - 2J) # 乗法
(5+0j)
>>> (97 + 0j) / (4 + 9j) # 除法
(3.9999999999999996-9j)
```

97 は素数であるが、$97 = (4 + 9i)(4 - 9i)$ となる。

複素数と順序

複素数体は順序体ではない

実数のような全順序関係を定義できない！

Python も複素数体は順序体ではないことを知っている

```
>>> -100 - 100j < 65536 + 256j # 右辺が大きそうに思えるが...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between
instances of 'complex' and 'complex'
```

順序体における平方元は非負である。

複素数の共役

複素数の共役

複素数 $z = x + iy$ に対して $\bar{z} = x - iy$ を z の共役と呼ぶ。

Python における複素数の共役

```
>>> z = 5 - 3j
>>> z.conjugate()  # complex 型のメソッドとして
(5+3j)
>>> z * (z.conjugate() / abs(z)**2)  # 逆元を構成する
(1+5.551115123125783e-17j)
```

共役の説明はどこにある？

組み込み型や cmath ではなく numbers モジュールで説明されている。

複素数の極座標表記

複素数平面

複素数 $z = x + iy$ を 2 次元実数平面 $\mathbf{R}^2$ 上の点 (x, y) とみなすことができる。これを複素数平面という。

複素数の極座標形式

複素数平面上の点 $z = x + iy (x, y \in \mathbf{R})$ を実部 x と虚部 y の組 (x, y) ではなく原点からの距離 r と偏角 θ の組 (r, θ) でも定義できる。これを複素数の極座標形式という。

複素数の極座標表記

百聞は一見に然り

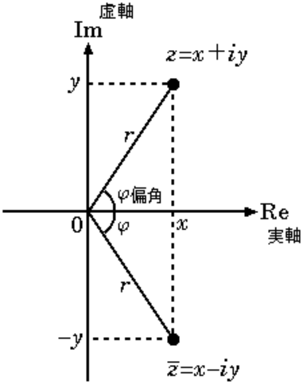

Figure: 複素数平面（Wikipedia から引用）

Python における複素数の極座標表記

Python における複素数の四則演算

```
>>> import cmath  # 真打登場
>>> z = 1 + 2j  # 直交座標から極座標に変換する
>>> r, phi = cmath.polar(z)
>>> r, phi  # r = abs(z), phi = cmath.phase(z)
(2.23606797749979, 1.1071487177940904)
>>> w = cmath.rect(r, phi)  # 極座標から直交座標に変換する
>>> w
(1.0000000000000002+2j)
>>> cmath.isclose(z, w)  # == ではなく isclose を使う
True
```

指数函数

指数函数 cmath.exp(x)（公式ドキュメントより）

e を自然対数の底として、e の x 乗を返します。

自然対数の底の複素数乗って何？？？

- 自然数乗 $\longrightarrow$ わかる
- 整数乗 $\longrightarrow$ わかる
- 有理数乗 $\longrightarrow$ まだわかる
- 実数乗 $\longrightarrow$ まだこれなら...
- 複素数乗 $\longrightarrow$ これもうわかんねぇな

極限で定義すればいいんだ！

$$e^z := \lim_{n\to\infty}\left(1+\frac{z}{n}\right)^n.$$

無限級数で定義すればいいんだ！

$$e^z := \sum_{n=0}^{\infty}\frac{z^n}{n!}.$$

実函数で定義すればいいんだ！

$$e^z := e^x(\cos y + i\sin y)$$

where $z = x + iy$.

指数函数を巡る冒険

極限値による定義

```python
def exp_by_limit(z: complex, n: int = 10) -> complex:
    N = 10 ** n
    return pow(1 + z / N, N)
```

無限級数による定義

```python
def exp_by_series(z: complex, n: int = 30) -> complex:
    return sum(pow(z, i) / factorial(i) for i in range(n))
```

実函数による定義

```python
def exp_by_real_func(z: complex) -> complex:
    x, y, i = z.real, z.imag, (0 + 1j)
    return math.exp(x) * (math.cos(y) + math.sin(y) * i)
```

見せてもらおうか、cmath.exp の威力とやらを

自作の関数で Euler の等式 $e^{i\pi} = -1$ を計算してみる

```
>>> z = cmath.pi * 1j
>>> exp_by_limit(z)
(-1.000493601977009+1.0340526558763341e-07j)
>>> exp_by_series(z)
(-1.0000000000000002+3.461777852236587e-16j)
>>> exp_by_real_func(z)
(-1+1.2246467991473532e-16j)
```

cmath.exp で Euler の等式 $e^{i\pi} = -1$ を計算してみる

```
>>> cmath.exp(z)
(-1+1.2246467991473532e-16j)
```

離散 Fourier 変換

離散 Fourier 変換

複素函数 $f(x)$ の離散 Fourier 変換 $F(t)$ は

$$F(t) = \sum_{n=0}^{N-1} f(n) e^{-i\frac{2\pi nt}{N}}$$

で与えられる。

離散 Fourier 変換の応用分野は 2 つ

- 信号解析
- データ圧縮
- その他諸々

離散 Fourier 変換

例：信号解析

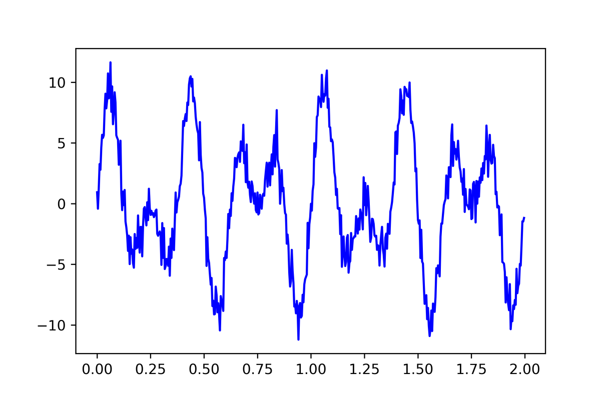

Figure: 謎の信号

例：信号のサンプリング

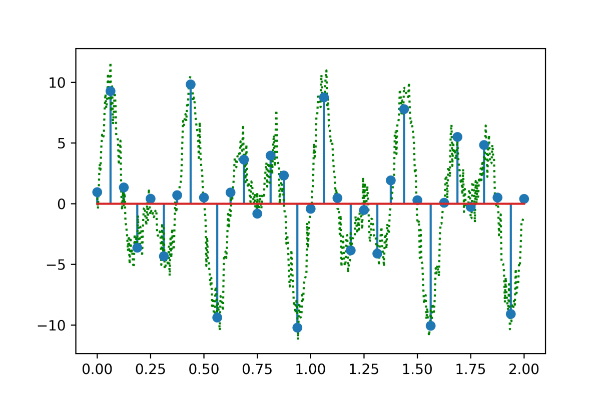

Figure: 謎の信号をサンプリングする

離散 Fourier 変換

離散 Fourier 変換（高速 Fourier 変換ではない！）

```
inverse_Fs = [
    sum(
        (1 / sampling_freq)
        * Fs[n]
        * cmath.exp(2j * cmath.pi * k * n / sampling_freq)
        for n in range(sampling_freq)
    )
    for k in range(sampling_freq)
]
```

離散 Fourier 変換

信号の周波数スペクトル

```
plt.stem(
    range(-sampling_freq // 2, sampling_freq // 2),
    list(map(abs, Fs)),
    use_line_collection=True,
)
```

離散 Fourier 変換

例：信号の周波数スペクトル

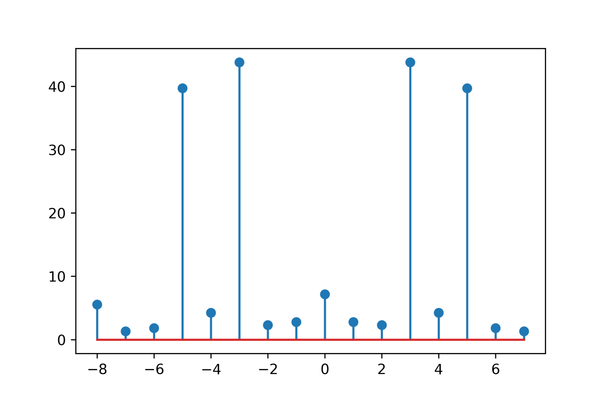

Figure: 謎の信号の周波数スペクトル

離散 Fourier 変換

離散 Fourier 変換

逆離散 Fourier 変換は

$$f(n) = \frac{1}{N}\sum_{n=0}^{N-1} F(t)e^{i\frac{2\pi nt}{N}}$$

で与えられる。

離散 Fourier 変換

逆離散 Fourier 変換

```
# 逆離散 Fourier 変換
inverse_Fs = [
    sum(
        (1 / sampling_freq)
        * Fs[n]
        * cmath.exp(2j * cmath.pi * k * n / sampling_freq)
        for n in range(sampling_freq)
    )
    for k in range(sampling_freq)
]
# 実部だけ取り出す
real_attr = operator.attrgetter("real")
inverse_Fs_real = list(map(real_attr, inverse_Fs))
```

離散 Fourier 変換

例：信号を復元する

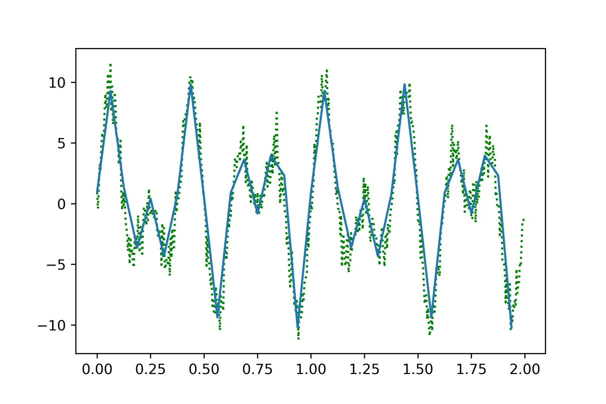

Figure: 逆離散 Fourier 変換で元の信号を復元する

1 の n 乗根

1 の n 乗根

1 の n 乗根は

$$e^{i\frac{2\pi k}{n}}(k = 0, \ldots, n-1)$$

で与えられる。

1 の n 乗根

1 の n 乗根で正 n 角形を作図する

```
N = 7
roots_of_one = [
    cmath.exp(((2 * cmath.pi * k) / N) * 1j)
    for k in range(N + 1)
]
angles = list(map(cmath.phase, roots_of_one))
length = list(map(abs, roots_of_one))
plt.polar(angles, length)
```

1 の n 乗根

1 の 7 乗根による正七角形の作図

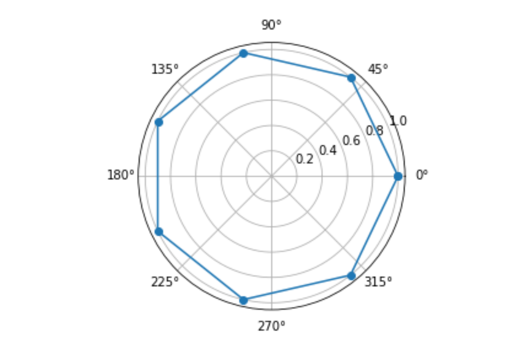

Figure: 正七角形

正七角形は定規とコンパスによって作図ができない。

まとめ

まとめ

- `cmath` は複素数のための数学ライブラリである。
- 標準ライブラリの範囲でも離散 Fourier 変換はできる。
- 君だけの最強の `cmath` アプリケーションを実装しよう！