



3. Python を使った実践データ解析

鈴木康浩

核融合科学研究所

(原稿受付：2008年1月24日)

3.1 はじめに

この章では、スクリプト言語 Python を使ったデータ解析を紹介します。Python の実行に必要な環境はインターネットから無償で入手できます¹。この章で紹介するスクリプトはすべて Linux (openSUSE10.3, Python2.5), Windows Vista (Python2.5) で動作確認をしています。

Python とは

Python は軽量言語 (Lightweight Language) の一種で、スクリプト言語と呼ばれるプログラミング言語です。Python は、スクリプトと呼ばれるプログラムリストをインタプリタとして実行します²。スクリプト自体はプログラムが記述されたテキストファイルなので、Python の実行環境があれば、マシン環境に依存することなく実行可能です。Python はオブジェクト指向言語とも呼ばれ、多くのツール、ライブラリを利用することが可能です。したがって、適用範囲は広く Web ベースのアプリケーションから本格的な GUI ソフトまで様々です。そのため、他の言語 (Java や C++, Perl や Ruby) とよく比較されます。そのような中、筆者が Python を利用する理由として、以下のようなものがあります。

1. 海外を中心にユーザーが多く、科学技術計算用のライブラリやモジュールが多く開発されています。
2. 必要最小限の機能のみを提供しています。Perl の TIMTOWTDI (there's more than one way to do it) とは対照的なアプローチをとり、誰がスクリプトを書いても同じようなものになるよう言語設計がなされています³。逆をいえば、参考書に書いてある定石をしっかり勉強するだけでよいことになります。
3. プログラミング言語としてだけでなく、いくつかのプログラムをまとめるシェルとして利用することができます。このとき Python の特徴である、ライブラリやモジュールを使うことにより、GUI 環境を持つ洗練されたラッパープログラムを開発することが可能です⁴。

Python は、巷で言われるようにオブジェクト化した Perl のようなものです。文法は C 言語に大変よく似ています。もし、Perl や Ruby など他の言語を習得しているのであれば、簡単にマスターできるでしょう。初心者でも、参考書を 1 冊真剣に読み込めば、簡単なテキスト処理はすぐにできるようになるでしょう [1-3]。

本講座では、Python の詳しい文法解説は割愛します。Python プログラムの書き方は、参考文献に示した参考書を参照してください。

3.2 Python でデータ処理をしてみよう

では、どのように Python でデータ処理をするか、シンプルなデータビューワーを作ることを目標にして、見ていきましょう。リスト 1 は "Vf-i-dia@73116.dat" というファイルを 1 行ずつ読み込んで表示するだけの単純なプログラムです。

リスト 1

```

1 # sample script 1
2
3 import re
4
5 data_id = "Vf-i-dia"
6 shot_id = "73116"
7 extension = ".dat"
8 data_path_r = "./"
9
10 rfile = data_path_r + data_id + "@" + shot +
    _id + extension
11 fr = open(rfile, "r")
12
13 for line in fr:
14     a = re.split(",", line)
15     print a[0], a[1]
16

```

1 <http://www.python.org/>

2 処理系としてはバイトコードに変換して処理するので Java に似ています。

3 他の人が書いたモジュールを見比べても、教科書どおりの書き方であることが多いです。

4 Python をラッパーとして FORTRAN プログラムを呼び出す例は第 7 章で紹介します。

ファイル"Vf-i-dia@73116.dat"の中身はリスト2のとおりです。

リスト2

```

1 Time (ms), data (V)
2 0.110000E+02, -0.374625E-02
3 0.110020E+02, -0.374625E-02
4 0.110040E+02, -0.374625E-02
5 0.110060E+02, -0.374625E-02
6 0.110080E+02, -0.374625E-02
7 0.110100E+02, -0.374625E-02
8 0.110120E+02, -0.374625E-02
9 0.110140E+02, -0.374625E-02
10 0.110160E+02, -0.374625E-02
11 0.110180E+02, -0.374625E-02
12 .
13 .
14 .
15 中略

```

紙面の都合で12行目以降は割愛していますが、実際には15001行あります。これはプローブ信号の時系列データで、1列目は時間(単位はms)、2列目はプローブ電圧(単位はV)になります。このデータをグラフにしたものが図1です。ではリスト1を詳しく見ていきましょう。

1行目はコメント行です。#以降はコメントとして処理されます。

3行目は標準ライブラリ⁵の一つである正規表現モジュール `re` をインポートしています。Pythonにはサードパーティ製を含め様々な機能がライブラリ、モジュールとして提供されています。外部モジュールを利用する場合は(標準で用意されているものを含め)、プログラム先頭で必ず読み込み(インポート)します。インポートしたいモジュールが複数ある場合は、以下の例のようにカンマで区切って記述します。

```
import re, math, pylab
```

この例では、標準ライブラリに含まれる正規表現モジュール `re`、数学関数モジュール `math`、外部ライブラリである `pylab` をインポートしています。

5行目から11行目まではファイル名を指定して、ファイルを開くまでを記述しています。5行目から8行目まではファイル名を文字列を使って定義します。**Pythonは変数の型(タイプ)と名前を事前に定義する必要がありません**⁶。変数に文字列を代入すれば自動的に文字列型に、数値を代入すれば数値型になります。ここでは、ファイル名が

```
"信号名"+"@"+"ショット番号"+"拡張子"
```

によって定義される場合を考えます。文字列は10行目のような文字列演算によって、複数の文字列変数を合わせて新

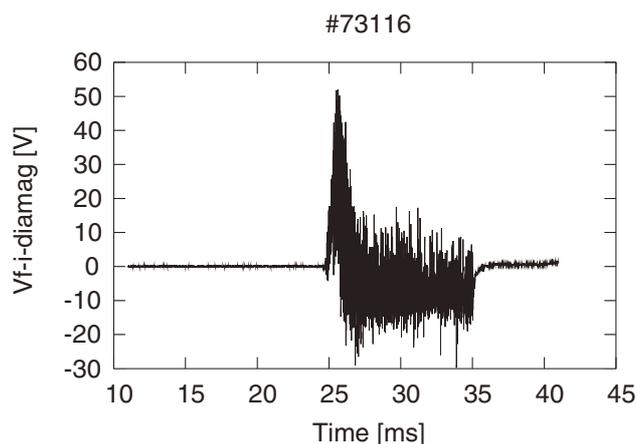


図1 Vf-i-dia@73116.datのグラフ。

しい文字列変数を作成できます。10行目は、ファイルが存在するディレクトリのパス(例題ではカレントディレクトリ)とファイル名を組み合わせて、開くべきファイルを指定します。ディレクトリのパスは絶対パス、相対パスの両方が使用できます。この演算により変数 `rfile` には `"/Vf-i-dia@73116.dat"` が代入されています。

11行目は関数 `open` を用いてファイルオブジェクト `fr` を作成します。Pythonでは、読み込んだファイルも1つのオブジェクトとして扱います。読み込んだファイルに対する操作は、読み書きを含めすべてファイルオブジェクトに対する操作になります。

13行目以降は、読み込んだファイルを1行ずつ表示します。この処理は、繰り返しになるので `for` 文を使用することになります。`for` 文の文法は、ほぼC言語と同じです。決定的に違うのは、`for` 文で繰り返し処理する部分をインデントすることです。C言語の場合は文という概念があります。FORTRAN言語の場合 `DO` と `END DO` で挟まれたブロックが繰り返し処理の対象になります。Pythonの場合は同じインデントの部分が、ブロックとして認識されます。したがって、以下のような場合、

```
for line in fr:
    a = re.split(",", line)
    print a[0], a[1]
```

`print` 文以下は繰り返し処理の対象として認識されません。一般的に、Pythonでは `for` 文や `if` 文のように、まとまった処理を行うブロックは、インデントにより明示します。`for` 文は以下のように、

```
for 繰り返し変数 in シーケンス:
```

繰り返し変数とシーケンスで構成され、行末にコロンを付け `for` 文が終わったことを示します。シーケンスには、通常、リストを用います。リストはC言語やFORTRAN言語で配列と呼ばれます⁷。変数と同じく、あらかじめ型、サイ

5 Pythonの配布セットにあらかじめ含まれているライブラリです。

6 PerlやRubyを含め軽量言語に共通する仕様です。

7 Pythonでは、同じく配列のように働くタプルと辞書があるので注意が必要です。詳しくは参考書を見てください。

ズを定義する必要はありません。リストを作る場合は次のようにします。

```
list = (1,2,3,4,5,6)
```

自動的にサイズが6のリストに整数が格納されます。ところでC言語やFORTRAN言語の繰り返し処理の場合、繰り返し変数は数値でインクリメントなどを行うわけですが、Pythonの場合は繰り返し変数が数値とは限らず文字列なども扱えます。次のような文字列、

```
a = "abcde"
```

を考えます。文字列 a をシーケンスにした for ループを考えてみましょう。

```
for i in a:
    print i
```

このループでは繰り返し変数 i はどうなるのでしょうか？このループの出力結果は次のようになります。

```
a
b
c
d
e
```

文字列そのものが繰り返し変数となります。このように、文字列やリストの要素をシーケンスとして扱える機能はデータ解析を行う場合に強力です。この例題で、読み込むファイル名を "Vf-i-dia@ショット番号.dat" としたのは理由があります。ショット番号が格納されたリストを

```
shot_id = ("73116", "73117", "73118", "\
73119", "73120")
```

のように考えます。するとリスト shot_id をシーケンスとした for ループ

```
for id in shot_id:
    rfile = data_path_r + data_id + "@" + \
    id + extension
```

リスト 3

```
1  #!/usr/bin/env python
2
3  import re,  numpy
4
5  data_id = "Vf-i-dia"
6  shot_id = "73116"
7  extension = ".dat"
8  data_path_r = "./"
9
10 tstart = 20.0 # start time
```

```
fr = open(rfile, "r")
```

を作れば、リスト shot_id の要素を次々と呼び出しながらファイルを開き、複数の実験データを処理することができます。一方、リスト 1 ではファイルオブジェクト fr を繰り返し処理のシーケンスとして利用しています。例題の処理では、ファイルオブジェクト fr が 1 行ずつ繰り返し変数 line に代入されます。したがって、

```
for line in fr:
    print line
```

と書けば、1 行ずつ読み込んだファイルをそのまま表示するプログラムになります。では、C 言語や FORTRAN 言語のようにループ回数を指定したい場合はどうすればいいのでしょうか？ そのような場合は、以下のように記述します。

```
for i in range(10):
    print i
```

この場合、0 から 9 までの整数が出力されます。関数 range は、例のように引数 10 を与えると 0 から 9 まで 1 ずつインクリメントするリストを返します。

説明を続けましょう。14 行目はモジュール re 中の関数 split により、区切り文字をカンマ"," として行を要素ごとに分解し、リスト a に代入します。モジュール中の関数を使用する場合は、

モジュール名.関数名(ある場合は引数)

の形で使用します。例題の 14 行目ではリスト a が自動的に定義され、カンマで区切られていた文字列がそれぞれ 1 要素としてリストに格納されます。16 行目では、リスト内のインデックス 0 と 1 の要素⁸を print 文で表示しています。

ファイルを開いて読み込むところまでは理解できましたか？では、読み込んだデータを実際に加工してみましょう。図 1 をみると、波形に随分リップルが乗っています。そこで、ある間隔で平均化されたデータを計算してみましょう。リスト 3 は、リスト 1 のスクリプトを改良して、ある間隔で平均化されたデータを計算するプログラムです。

8 リストのインデックスは 0 から始まり、括弧 [] で指定します。

リスト 3 (続き)

```
11 tend = 40.0 # end time
12 dtint = 1.0 # interval
13
14 rfile = data_path_r + data_id + "@" + shot_id + extension
15
16 n_data = len(open(rfile, "rU").readlines())
17
18 xx = numpy.zeros(n_data-1)
19 yy = numpy.zeros(n_data-1)
20 xx2 = numpy.zeros(n_data-1)
21 yy2 = numpy.zeros(n_data-1)
22
23 fr = open(rfile, "r")
24
25 a = fr.readline()
26
27 for line in range(n_data-1):
28     a = fr.readline()
29     b = re.split(",", a)
30     xx[line] = float(b[0])
31     yy[line] = float(b[1])
32
33 fr.close()
34
35 cnt = 0
36 mcnt = 0
37 ncnt = 0
38 Vf_ave = 0.0
39 tave_s = tstart - 0.5 * dtint
40 tave_e = tstart + 0.5 * dtint
41 while cnt <= n_data-2:
42     if (xx[cnt] >= tstart and xx[cnt] <= tend) :
43         if (xx[cnt] >= tave_s and xx[cnt] < tave_e) :
44             Vf_ave = Vf_ave + yy[cnt]
45             ncnt += 1
46         elif (xx[cnt] >= tave_e and ncnt > 1) :
47             Vf_ave = Vf_ave / ncnt
48             xx2[mcnt] = 0.5 * (tave_s + tave_e)
49             yy2[mcnt] = Vf_ave
50             mcnt = mcnt + 1
51             ncnt = 0
52             Vf_ave = 0.0
53             tave_s = tave_e
54             tave_e = tave_s + dtint
55         cnt += 1
56
57 for line in range(mcnt):
58     print line, xx2[line], yy2[line]
```

ここでは、20 ms から 40 ms まで 1 ms 刻みで平均化したデータを計算することを考えます。ではリスト 3 を見ていきましょう。1 行目に

```
#!/usr/bin/env python
```

と書かれています。スクリプト先頭の#!で始まる行は特別で、スクリプトファイルに実行属性を付ければ直接起動できます⁹。このあたりは通常のシェルスクリプトと同様です。Windows 環境の場合、拡張子を".py"としたファイルを作成し、ダブルクリックすれば直接起動できます。ただし、Windows 環境の場合はスクリプトが終了するとウィンドウがすぐに閉じてしまい、結果の確認には不便です。そのような場合、スクリプトの最後に raw_input() と命令を加えておけば、リターンキーが押されるまでウィンドウが開いたままになります。

3 行目では標準ライブラリに含まれる正規表現モジュール re のほかにサードパーティー製の外部モジュールである numpy¹⁰ をインポートしています。numpy は、FORTRAN 言語のような多次元配列オブジェクトや、基本的な線形代数、高速フーリエ変換、特殊関数などの機能を提供するモジュールです¹¹。Python のリストは C 言語や FORTRAN 言語の配列とは違います。次元やサイズをあらかじめ定義する必要はありません。ところが、実際のデータ解析の場面ではあらかじめサイズを指定した配列を用意してデータを格納し、その配列を用いて計算することが便利な場合が多くあります。numpy は FORTRAN 言語のように配列を扱えるようにするものです。

numpy で配列を定義するには次のようにします。リスト 3 では、16 行目でファイルの行数を数えています¹²。18 行目から 21 行目でサイズが n_data-1¹³ の配列 xx, yy, xx2, yy2 を定義します。zeros が引数のサイズで配列を定義し、0 を代入して初期化する関数です。インポートしたモジュール numpy に含まれる関数 zeros を使うので、

```
xx = numpy.zeros(n_data-1)
```

のように記述します。

ところで、インポートしたモジュールに含まれる関数を"モジュール名. 関数名"のようにいちいち記述するのは面倒な場合があります。たとえば標準ライブラリに含まれる数学関数モジュール math には円周率があらかじめ定義されています。次の例を見てください。

```
In [1]: import math
```

```
In [2]: math.pi
```

```
Out [2]: 3.1415926535897931
```

円周率を用いた計算をしたい場合は、math.pi とすればよいわけです。ただしこれを math.cos(math.pi) などと書いていると面倒になってくる場合があります。pi, あるいは cos(pi) と書きたい場合は、次のようにモジュールをインポートします。

```
from math import *
```

* はワイルドカードです。このようにモジュールをインポートすると、関数を利用するときにモジュール名を省略できます。ただし、from 文を使ったモジュールのインポートは注意が必要です。たとえばモジュール numpy と後で説明するモジュール scipy には同じ名前の関数 fft が含まれています。関数名は多くの場合で抽象的な名前が付けられるので、異なるモジュールに同じ名前の関数が含まれることは想像に難くありません¹⁴。したがって、上の例のようにワイルドカード*を使ったモジュールのインポートは、名前空間が重ならないよう注意が必要です。本講座では対象としませんが、オブジェクト指向を陽にプログラミングする場合はモジュール内に階層構造をもった関数が現れる場合があるので、筆者の流儀として、面倒でも from 文は使わないでインポートしています。ただし、リスト 3 程度のスクリプトでは問題ないでしょう。あるいは、以下の例

```
from math import pi, cos
```

のように必要な関数のみインポートすることをお勧めします。繰り返しになりますが、関数名が重なり合わないよう、名前空間には注意してプログラミングしてください。

説明を続けます。27 行目から 31 行目でファイル中の数値データを配列に格納します。データは配列に格納されているので、33 行目でファイルを閉じます。今後の計算はすべてメモリ上で行われます。

35 行目から 55 行目は、20 ms から 40 ms までの間で 1 ms ずつ条件判定でデータを取り出しながら、平均化処理をしています。このブロックは while 文と if 文で入れ子状の階層構造になっています。同じブロックで処理したい命令は必ず、同じインデントをとります。while 文と if 文は特に説明はいらぬいですね。それぞれ条件判定の命令文です。

最後に、ある間隔で平均化され、配列 xx2, yy2 に格納されていたデータを出力します。結果を図 1 と同様にプロットしたものが図 2 です。時間間隔で平均化した結果、信号の傾向が見やすくなりました。

9 Linux などの UNIX 環境上用です。

10 <http://numpy.scipy.org/>

11 Numeric, numarray と 2 つ存在した拡張をマージしたものです。

12 Python クックブックに載っている典型的な手法です。

13 1 行目がヘッダ行なので。

14 いくつもあります。orz...

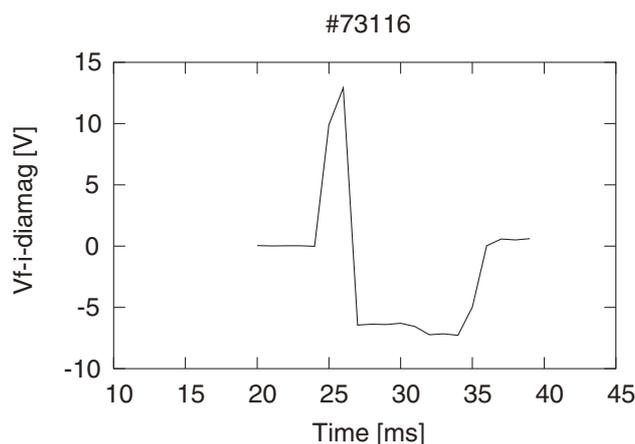


図2 Vf-i-dia@73116.dat を例題2のスキプトに通して平均化した結果。

3.3 科学技術計算用ツールSciPyと周辺ツールの紹介

簡単なスクリプトを例に、Pythonを使ったデータ解析例を紹介しました。しかし、この程度の処理であれば、必ずPythonで処理する理由にはなりません。筆者がPythonをおすすめする理由の一つが、SciPyの存在です。SciPy¹⁵は、統計、最適化、積分、線形代数、フーリエ変換、信号・イメージ処理、遺伝的アルゴリズム、ODE（常微分方程式）solver、特別な関数、その他のモジュールを提供します¹⁶。線形代数やフーリエ変換に関しては、有名なFORTRAN言語用数値計算ライブラリであるBLAS¹⁷、LAPACK¹⁸やFFTW¹⁹のラッパープログラムとして働くので高速に動作します。SciPyは前節で紹介したNumPyを使用しますので、あらかじめサイズを指定した数値配列を利用することもできます。SciPyで提供される機能は、FORTRAN言語やC言語ではよく知られた枯れたライブラリで提供されているものと同等です。したがって、既存の言語環境でもこれらのライブラリを利用すれば、本質的には同等です。それでもPythonをおすすめする理由は、スクリプト言語としての簡易性、充実した文字列処理、優れた外部ライブラリとの接続性になります。FORTRAN言語やC言語でもできますが、ファイル内のデータを加工して、計算して、さらに表示するという考えると、簡便性と開発の容易さでPythonが勝ると筆者は考えています²⁰。

では、実際にSciPyを使ってみましょう。前節で用いたリスト3を改良して、データをフーリエ変換してみましょ

リスト4

```

1 xx3 = numpy.zeros (mcnt)
2 yy3 = numpy.zeros (mcnt)
3
4 yy3 = scipy.fftpack.fft (yy2 [0:mcnt-1]) / xx2 [0:mcnt-1].size
5

```

15 <http://www.scipy.org>

16 <http://en.wikipedia.org/wiki/SciPy>

17 <http://www.netlib.org/blas/>

18 <http://www.netlib.org/lapack/>

19 <http://www.fftw.org/>

20 Rubyを同様の理由で選択することも出来ます。どちらがよいかは趣味の問題でしょう。

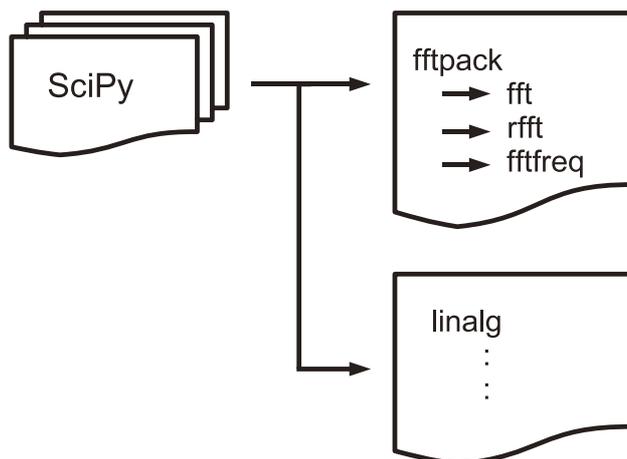


図3 モジュールSciPyの階層構造。

う。SciPyを使うにはモジュール `scipy` をインポートするのですが、注意点が一つあります。前に述べたよう、SciPyは広範な機能を持つモジュールライブラリです。そこで、広範な機能を効率よく管理・利用できるようにモジュールは階層構造を持っています。SciPyの階層構造を図3に示します。1番上の階層として `scipy` があり、2番目の階層として各機能の高速フーリエ変換 `fftpack`、線形代数 `linalg`、補間 `interpolate`、特殊関数 `special` などがあります。ここでは高速フーリエ変換を用いたので、`fftpack` をインポートします。`fftpack` のインポートは、次のように行います。

```
import scipy.fftpack
```

階層構造を持つモジュールで下階層のモジュールを利用したい場合には、

モジュール名 . 機能名

のようにドット"."でモジュール名と機能名を繋ぎます。モジュール `fftpack` の中で複素フーリエ変換を行う関数は `fft` です。この関数を用いたいときは、前に述べたルールに従って、

```
scipy.fftpack.fft(x)
```

と呼び出します。リスト3に追加する部分をリスト4に示します。

```

6 dt = 1.0 / (tend - tstart)
7 for i in range (mcnt/2) :
8     xx3 [i] = dt * i
9     print xx3 [i], abs (yy3 [i])**2

```

リスト3の最後にリスト4を追加するだけで信号のフーリエ解析ができます。fftは複素フーリエ変換の関数で、フーリエ変換された信号の配列を返します。6行目で周波数の単位がkHzになるようにしています。関数fftが返すフーリエ振幅は N で規格化されているので、例題では $1/N$ をかけています²¹。10行目では関数absを用いて複素共役の自乗を取ってパワースペクトルを求めています。

SciPyの中に含まれているわけではありませんが、SciPyと組み合わせて使えるすぐれたグラフィックライブラリがあります。その中で、本講座ではmatplotlib²²を紹介しま

リスト5

```

1 pylab.plot (xx3 [0:mcnt/2], abs (yy3 [0:mcnt/2])**2)
2 pylab.xlabel ('Frequency [kHz]')
3 pylab.ylabel ('Vf-i-dia [V$^2$/Hz]')
4 pylab.title ('#73116')
5 pylab.show ()

```

リスト5を追加したスクリプトを実行して得られた結果が図4です。たったこれだけで、結果を図にすることができます。matplotlibを使ってグラフを描く基本は関数plotで、以下のように使います。

```
plot (x, y)
```

x と y は、それぞれX軸とY軸が収められている1次元の配列です。xlabel, ylabel, titleは、それぞれX軸のラベル、Y軸のラベル、グラフのタイトルです。最後に関数showで画面を表示します。リスト5の3行目で

```
pylab.ylabel ('Vf-i-dia [V$^2$/Hz]')
```

と書いていますが、図4を見ると $\$^2\$$ の部分が上付き文字になっています。matplotlibでは、TeXと同じような文法で数式やギリシャ文字をグラフ中に書き込めます。この機能がまさに、論文品質の図が書ける所以なのです。matplotlibのプロジェクトページには様々形式のグラフのスクリーンショットとサンプルスクリプトがあるので参考にしてください²⁵。図4の画面を見ると、図下部に様々なボタン類が配置されています。これらを使うことで、ズームや画像を保存することができます。関数savefigを使えば

```
savefig ('test.png')
```

21 ライブラリ FFTW の仕様です。

22 <http://matplotlib.sourceforge.net/>

23 <http://ipython.scipy.org/>

24 実際、ユーザーインターフェースと命令文が似ています。

25 <http://matplotlib.sourceforge.net/screenshots.html>

26 <http://www.inkscape.org/>

27 Adobe Illustrator でも SVG を扱えます。

す。matplotlibは、原著論文に載せるに足る品質の図を作ることができる2次元グラフィックライブラリです。また、PythonのインタラクティブシェルIPython²³と組み合わせることでMATLABと同等のことができるようになります²⁴。ここでは、MATLAB互換としての環境はさておき、結果を手取り早くプロットするものとして説明します。

リスト4の結果をmatplotlibを用いてグラフにしてみましょう。matplotlibを使うには、モジュールpylabをインポートします。次にリスト5を追加してください。

のようにスクリプト中で画像フォーマットにより保存することを命令できます。デフォルトの画像フォーマットはPNG (Portable Network Graphics)です。ほかに、PS, PDF, SVGなどを選べます。SVG (Scalable Vector Graphics)形式で保存すれば、あとでInkscape^{26,27}のようなベクターソフトウェアで、さらに画像(グラフ)を編集できるので非

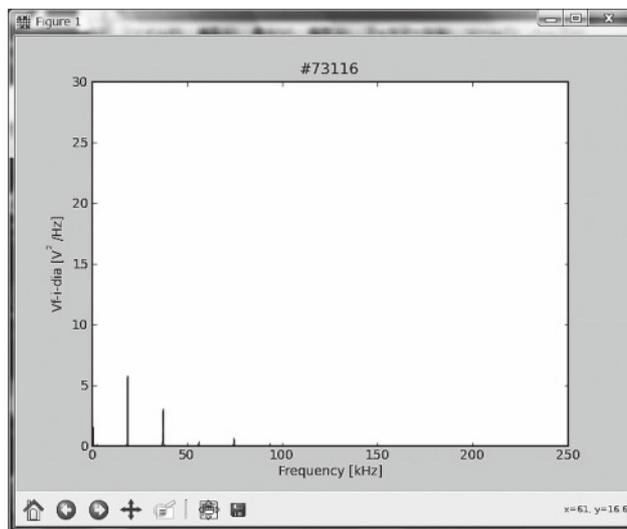


図4 リスト3+リスト4+リスト5を実行した結果。

常に便利です。

matplotlibの関数を使えば次のようなこともできます。
リスト6を例題2に追加してください。

リスト6

```

1 pylab.subplot(211)
2 pylab.plot(xx,yy)
3 pylab.ylabel('Vf-i-dia [V]')
4 pylab.grid(True)
5 pylab.title('#73116')
6 xmin, xmax = pylab.xlim()
7 ymin, ymax = pylab.ylim()
8
9 pylab.subplot(212)
10 pylab.plot(xx2[0:mcnt-1],yy2[0:mcnt-1])
11 pylab.xlabel('Time [ms]')
12 pylab.ylabel('VF-i-dia [V]')
13 pylab.grid(True)
14 pylab.title('averaged')
15 pylab.xlim(xmin,xmax)
16 pylab.ylim(ymin,ymax)
17
18 pylab.show()

```

リスト6を追加したリスト3を実行した結果が図5です。元の時系列データとリスト3の結果により20msから45msまで1ms刻みで平均化されたデータを重ねてプロットしてあります。リスト6で新しく出てきた関数のうち、関数subplotが図を重ねて表示するためのものです。関数subplotは、

```
subplot (numrows, numcols, axesnum)
```

リスト7

```

1 #!/usr/bin/env python
2
3 import re, numpy, scipy.fftpack, pylab, sys
4
5 data_id = "Vf-i-dia"
6 shot_id = sys.argv[1]
7 extension = ".dat"
8 data_path_r = "./"
9
10

```

これまでのスクリプトはすべて、ショット番号やファイル名などスクリプト中に埋め込んでいました。Pythonのようなスクリプト言語は、実行するためにいちいちコンパイル作業はいらないうえに、短い作業であればスクリプトそのものを編集してしまっても手間にはなりません。ただし、大量のデータをまとめて作業するのであれば、コマンドラインからショット番号やファイル名を渡せるようにしたほうが便利です。6行目で

28 インデックス0はスクリプト自体です。

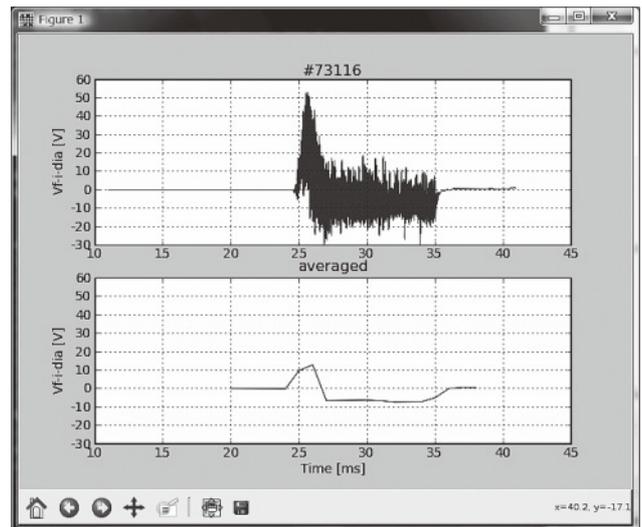


図5 リスト3+リスト6を実行した結果。

のように使います。リスト6の例では、関数subplotの引数に1行目では211、9行目では212を与えています。それぞれ、2行1列のグラフの1つ目と2つ目を指定します。関数xlim,ylimはそれぞれ、X軸とY軸の範囲を

```
xlim(xmin, xmax)
```

のように指定します。

最後に、スクリプトにパラメータをコマンドライン引数から与えられるようにして終わりにしましょう。コマンドラインからスクリプトに引数を渡すには、モジュールsysの関数argvを使います。リスト7に、関数argvの使い方の1例を示します。

```
shot_id = sys.argv[1]
```

として、shot_idにsys.argv[1]を代入しています。関数argvはコマンドラインからの引数のリストを返します²⁸。

```
./example.py 73116
```

のように実行すればshot_idにショット番号73116が代入

されます。これを応用すれば、リスト3の `tstart`, `tend` の値も引数から与えられます。簡単に改造できるので、試してみてください。

3.4 おわりに

この章では、スクリプト言語 Python を使って、簡単なデータビューワーを作成しながら使い方の基本を説明しました。

はじめに述べたように、Python はオブジェクト指向言語です。オブジェクト指向を陽に書くのであれば、汎用的な機能はクラスを定義すべきです。しかし、本講座はとにかく、データ解析にまつわる作業を簡単にすることを目的としたので、オブジェクト指向どころか、自前の関数を定義することはしていません。本章では浮動電位のデータを例にスクリプトを作成しましたが、1種類の物理データに対して作業するだけならば十分だからです。ただし、1つのスクリプトで複数の物理データを処理する汎用的なプログラムをめざすならば、必要な機能を関数にまとめ、クラスを定義したほうが作業効率は上がるでしょう。オブジェクト指向の考え方も、参考文献で紹介した参考書を1冊読

みこめばわかるでしょう。本講座終了後に開設予定のサポートページでは、本章で紹介したスクリプト全文と、オブジェクト指向を用いた汎用プログラムも紹介する予定です。

謝 辞

本章では、東北大学ヘリアック装置の実験データ解析を行うためのスクリプトプログラムを流用する形で執筆しました。快くデータを提供していただいた、東北大学工学研究科の北島純男先生と東北大学ヘリアックグループの皆様感謝いたします。また、九州大学応用力学研究所稲垣滋氏との議論が、大変参考になりました。ここに感謝いたします。

参考文献

- [1] 柴田 淳：みんなのPython (ソフトバンククリエイティブ, 2006).
- [2] M. Lutz and D. Ascher：はじめてのPython 第2版 (オライリー・ジャパン, 2004).
- [3] A. Martelli, A.M. Ravenscroft and D. Ascher: Python クックブック 第2版 (オライリー・ジャパン, 2007).