J. Plasma Fusion Res. Vol.84, No.6 (2008) 376-382

講座 オープンソースソフトウェアを使った実践データ解析

7．軽量言語時代の重量言語

稲垣　滋
九州大学応用力学研究所

（原稿受付：2008年4月7日）

7.1　はじめに

これまでの章ではpython，ruby，octave等のいわゆる軽量言語で科学データ解析を行うことが紹介されてきました．軽量言語は数学や可視化のライブラリーが充実しており，データ解析ツールの開発時間を短縮できるという特徴があります．加えて，最近の軽量言語は汎用に開発されているものが多く，オブジェクト指向やネットワークプログラミング等の機能も備えており，今後のデータ解析環境の変化にも十分対応可能なだけの柔軟性を備えていると言えます．

従来，科学技術に関連した分野ではFORTRANやCが使われてきました．これらの言語は変数が強く型付けされており，実行にはコンパイルという作業が必要となります．このためプログラムの変更から実行までに時間がかかることやソースコードが長くなることから重量言語と呼ばれます．重量言語最大の特徴は，完成したアプリケーションの実行性能の高さです．しかし，最近は実行時間よりも，開発時間およびそのアプリケーションの使用頻度を考慮した全体の時間を最小化することが重要視されています．せいぜい2－3回しか使わないようなツールなら，重量言語で1週間かけて実行時間1時間のものを作るより，軽量言語で1日で作った実行時間10時間のものの方が良い，という考え方です．科学データ解析のように解析方法を試行錯誤するような場面では軽量言語が力を発揮することが多くなるかもしれません．

それでは科学データ解析には重量言語はもう必要ないのでしょうか？いや，やはり実行速度そのものが必要になるという場面はまだまだあります．例えば最近使われる実験データ時系列解析のひとつにバイスペクトル解析がありますが，通常のスペクトル解析以上の計算量がある上に統計精度向上のために多くのアンサンブルを必要とするため実行時間が非常に長くなります．このような解析ツールの開発は重量言語の出番です．

重量言語で開発されたアプリケーションは実行性能は高いのですが，ユーザインターフェィス等は軽量言語アプリケーションより明らかに見劣りがします．例えばFORTRANで開発されたアプリケーションの多くは入力ファイルをname listで読み込み，結果を常に同じ名前でファイルに保存します．入力ファイルを置く場所が決まっていたり，それを一度エディタで編集したり，と近代的なアプリケーションとは趣きが異なります．使いやすさというのもアプリケーションの重要な要素の一つです．この章では，餅は餅屋ということで，pythonを使ったアプリケーションインターフェイス作成を紹介します．まず最初に，FORTRANやCのプログラムにはいっさい手を加えずグラフィカルユーザインターフェィス（GUI）のみをpythonで作ります．次に，pythonメインプログラムから，Fortranで書かれたルーチンを呼び出します．

7.2　Pythonで GUI

PythonやRubyではいくつかのGUIツールが使えます．その多くはプラットフォーム非依存のものです．Pythonではpygtk，pyQt，WxPython，Tkinterなどが利用できます．pygtk，pyQt，WxPythonは比較的新しく高機能で，Tkinterは開発の歴史が古く，比較的安定しています．ここではpython officialのGUIツールであるTkinterを例に説明しますが，GUIプログラミングの基本は他のツールキットを用いた場合でも同じです．

まずTkinterをインストールします．Mac OSXであればMacportsが便利です．

```
sudo port install py-tkinter
```

でインストールされます．CentOS5.1であれば，

```
yum install tkinter
```

でOKです．この時Tix（tk拡張）も一緒にインストールされます．windowsでは多くの場合，pythonをインストールした時点でtkinterはインストールされています．それではTkinterの使い方を見てみましょう．**例題1**に非常に簡単なサンプルを示しました．

Practical Data Analysis Using Open Source Software 7. Heavyweight Language in an Age of Lightweight Computing

INAGAKI Shigeru　　*authors' e-mail: inagaki@riam.kyushu-u.ac.jp*

※サンプルプログラムをサポートページからダウンロードできます．ご利用ください．
http//www.jspf.or.jp/journal/koza_opens.html

例題 1

```
1  #! /usr/bin/env python
2
3  import Tkinter as Tk
4
5  class Sample(Tk.Frame):
6    def init(self):
7      frm_main  = Tk.LabelFrame(self, text="Input message", labelanchor=Tk.NW)
8      self.ent_skip = Tk.Entry(frm_main, width=20)
9      self.ent_skip.insert(0,"")
10     self.ent_skip.pack(side=Tk.TOP,fill=Tk.BOTH)
11     frm_btns   = Tk.Frame(self, relief=Tk.GROOVE, bd=3)
12     btn_cancel = Tk.Button(frm_btns, text="Cancel", command=self.cancel)
13     btn_ok     = Tk.Button(frm_btns, text="OK", command=self.ok)
14     for btn in [btn_cancel, btn_ok]:
15       btn.pack(side=Tk.LEFT)
16     frm_main.pack(side=Tk.TOP,fill=Tk.BOTH,pady=2)
17     frm_btns.pack(side=Tk.TOP,fill=Tk.BOTH,pady=2)
18
19   def ok(self, event=None):
20     print self.ent_skip.get()
21     self.master.destroy()
22
23   def cancel(self, event=None):
24     self.master.destroy()
25
26   def __init__(self, master=None):
27     Tk.Frame.__init__(self, master)
28     self.init()
29     self.pack()
30
31 s = Sample()
32 s.mainloop()
```

例題 1 を実行すると**図 1** になります．OK ボタンを押すとエントリーボックス内に記載されたメッセージが標準出力に出力され終了します．
tkinter を使うには基本的に

```
import Tkinter as Tk
```

とするだけです．
その後

```
Tk.Frame(), Tk.Label(), Tk.Checkbutton
(), Tk.Entry(), ...
```

などを用いて GUI の部品（ウィジェット）であるフレーム，ラベル，チェックボタン，エントリーボックス等を並べていきます．定義したウィジェットは明示的に“配置”しなければ実体化しません．“配置”するには pack，grid，place というメソッドを使います．

```
btn.pack(side=Tk.LEFT)
```

では btn を左ツメで配置します．

```
frm_main.pack(side=Tk.TOP,fill=Tk.BOTH,
pady=2)
```

で frm_main を上からツメて配置します．fill=Tk.BOTH

図 1　tkinter のサンプルプログラムの出力．

はウィジェットの周りに空いているスペースがあれば，縦横に広がることを指示してます．pady＝2でウィジェット外側の縦の隙間を2に指定しています．packは基本的に1次元的に配置しますが，**例題1**のようにフレームウィジェットを利用することで，縦配列と横配列を混在させることができます．一方，gridはウィジェットを2次元的に配置し，placeは任意の位置に配置します．基本的にGUIアプリケーションではユーザがウィジェットに何か操作（例えばボタンを押す）をした時，何らかの動作をします．この動作はコールバックルーチンとして与えます．**例題1**では

```
btn_cancel = Tk.Button(frm_btns, text="
Cancel", command=self.cancel)
```

とあり，btn_cancelが押されたら，cancelというメソッドが実行されます．このサンプルではclassを使っています．classを使わないともっとソースコードを短くできますが，実際のアプリケーションでは小さなGUIがたくさん必要です．このような場合はclassを使った方が開発が容易になります．

次に少し実用的な例を見てみましょう．サポートページにあるgetfield.pyは第2章で紹介されたawkのようにデータファイルからfieldを取り出すアプリケーションです．ダウンロードしたgetfield.pyに実行許可を与えて実行すると**図2**のような画面が現れます（**図2**はOSXでの実行例です．実行環境によりウインドウのルックアンドフィールが異なる場合があります．）．画面ではメニューバーにfileというアイテムがあります．fileメニューからopenを選択するとファイル選択ダイアログが開くので，データファイルを選択します．選択したファイルの最初の10行がテキストウィジェットに表示されます．ここでreadという枠内のskipというエントリーボックスの横の＋とーのボタンを押してみます．＋のボタンを押すとtextウィジェットの内容が1行上にシフトし，skipエントリーボックス内の数字が＋1されます．skipというエントリーボックスはファイルの先頭を何行読み飛ばすかを指定し，textウィジェットに表示されていない部分は無視されることを意味します．読み込み時にComment Outというエントリーボックスに指定した文字が先頭にある行は読み飛ばします．read titleをチェックすると最初に読み込んだ1行をtitleとして扱います．さらにIgnor leading '#'をチェックすると行頭の＃を無視します．これはgnuplotユーザのデータファイルにはtitle行の頭に＃がついていることが多いことを考慮しています（ただし，前に示したComment Outエントリーボックスの方が優先されるので，このoptionを指定する場合はComment Outのエントリーに注意が必要です）．Separatorではfieldの区切り文字を指定します．2つ以上の空白は一つの空白として解釈されます．OKボタンを押すと**図3**のような画面が現れます．readtitleをチェックしていなければチェックボタンの名前はv1, v2, …となります．ここで必要なfieldを一つ以上チェックしてOKボタンを押すとその

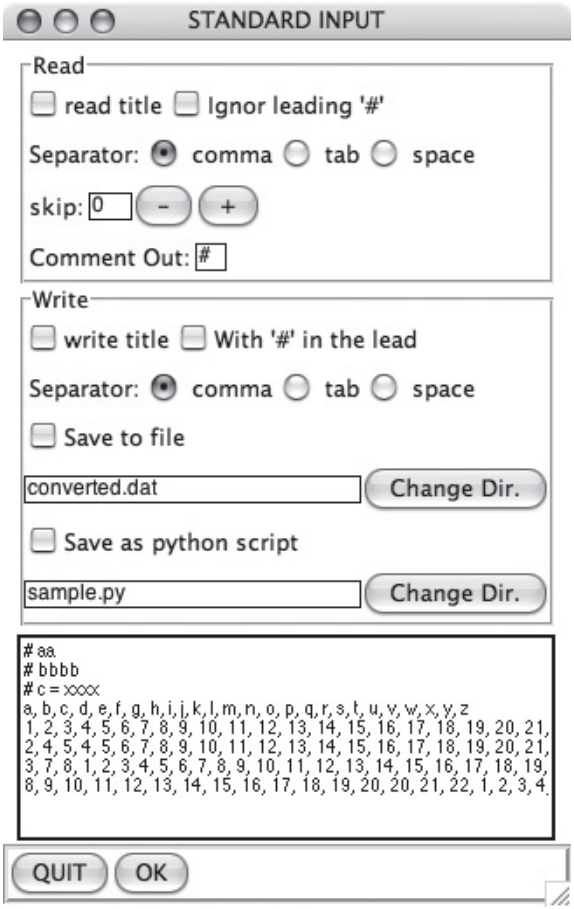

図2　getfield.pyの画面1．

図3　getfield.pyの画面2．

field のみが取り出せます．出力フォーマットは前の画面（**図 3**）の write という枠内で指定します．write title をチェックすると読み込んだ title が出力されます．save to file をチェックすると結果はファイルに出力されます．デフォルトでは結果は標準出力に出ます．getfield.py は入力ファイルをパイプで与えることもできます．

```
% less input.dat | getfield.py > result.dat
```

という使い方ができます．ここで%は端末のプロンプトです．さらに getfield.py の出力を他のコマンドに渡すこともできます．

```
% less input.dat | getfield.py | tee result.dat | xmgrace -pipe
```

とすれば，getfield.py の出力を result.dat に保存しつつ，grace によってグラフ表示することができます．サポートページにある graceplot.py は grace でのマルチプロットアシスタントです．

```
% less input.dat | getfield.py | graceplot.py -m | xmgrace -pipe
```

のように使います．
同様に epyon.py は第 3 章で紹介した matplotlib を使った簡易ビュワーです．

```
% less input.dat | getfield.py | epyon.py
```

のように使います．epyon.py は grace の苦手な image plot や contour をサポートしています．epyon.py には Tkinter による GUI が付属します．フォーカスのある画面上で右クリックすると，各種メニューが選ぶことができます．このメニューからプロットのカラー，線幅や legend 等を編集できます．詳細は README や epyon -h を参照ください．

他のアプリケーションと連携したり，同じフォーマットのファイルに対して同じ操作を行ったりするには GUI よりも CUI の方が便利なことが多いです．getfield.py の出力設定（**図 3**）で save as python script をチェックすると getfield.py の設定を python script として保存できます．getfield.py が作るスクリプトは標準入力から読み，標準出力に書くので shell 等によって他のアプリケーションと連携するのが簡単です．また，スクリプトファイルのコメントアウトを外し，代わりに print ... という行をコメントアウトし，インデントを調整するとカレントディレクトリ以下のすべての＊.dat というファイルのフォーマットを変更する，というスクリプトになります．

7.3 GUI で入力ファイルを作成

既存のアプリケーションに GUI をつけることを考えます．例として，FORTRAN で書かれたアプリケーション fortapp.exe があり，setup.inp という namelist ベースのファイルを読み込み動作するとします．この時は**例題 2** のようにするとアプリケーションが GUI を持っているように見えます．

例題 2

```
1  import os
2  import tkMessageBox
3
4  中略
5
6    def ok(self, event=None):
7
8      GUI 処理
9
10     if intval < 0 and intval > 253:
11       tkMessageBox.showinfo(message="intval out of range")
12       return
13
14     ここから setup.inp を作ります
15     f = open("setup.inp", "w")
16     f.write("&PARAM/n")
17     f.write("INTVAL = %s,/n"%intval)
18     f.write("DBLVAL = %s,/n"%dblval)
19     f.write("DBLVEC = ")
20     for val in dblvec:
21       f.write("%s,"%val)
22     f.write("/n")
```

```
23      f.write("CHARVAL = '%s',/n"%charval)
24      f.write("/n")
25      f.close()
26      Fortran プログラムを実行します
27      os.system("fortapp.exe")
28      self.master.destroy()
```

GUIをつけることのメリットとして，あらかじめDefaultの値をセットできる，入力された値が適正かどうかのcheckができる等があげられます．**例題2**ではintvalが0-253以外の値の場合，ダイアログが現れて警告し，再入力を促します．

ダイアログを使うには

```
import tkMessageBox
```

とするだけです．これもTkinterの持つ機能の一つです．**例題2**では入力ファイルの名前がいつも一緒ですが，もし新たにFORTRANアプリケーションを作るなら入力ファイルは引数で指定する方が安全です．Fortranプログラムで

```
  INTEGER             :: ierr
  CHARACTER(LEN=128) :: inputfile
!
  IF COMMAND_ARGUMENT_COUNT() <= 0 STOP
  CALL GET_COMMAND_ARGUMENT(1, input-
file, STATUS=ierr)
!
```

とすればinputfileを引数から得ることができます．fortran標準ではFortran95以前の仕様ではアプリケーションは引数を取ることはできませんでした（ただし，ほとんどのコンパイラは拡張としてサポートしていましたが）．COMMAND_ARGUMENT_COUNT，GET_COMMAND_ARGUMENTはFortran2003の機能です．多くのコンパイラがこの機能をサポートし始めています．一方，小さなツールであるためinputfileをいちいちopenするのが面倒な時，はinputfileは標準入力から読み込むようにします．

```
!
  INTEGER(i4b)        :: intval
  REAL(dp)            :: dblval, dblvec(10)
  CHARACTER(LEN=128) :: outfile
  NAMELIST /PARAM/intval, dblval,
dblvec, outfile
!
  READ(*,NML=PARAM)
!
!ここで
!  i4b = SELECTED_INT_KIND(9)
!  dp  = KIND(1.0D0)
!と定義されているとします．
!
```

としてname listを標準入力から読み込むようにし，通常は

```
% less setup.inp | fortapp.exe
```

のように使います．このように入力を標準入力にしておくとパイプを介したpythonとの連携が可能となります．

ここまで議論してきたプログラムの起動時ではなく，実行途中でGUIが必要な場合があります．その時もパイプを使うのがお手軽です．サポートページにあるfortranpipe.cとpipe_interface.f95はFortranプログラムからc言語を介してパイプを使うためのサブルーチンが入っています．**例題3**はFortranからgetfield.pyを呼び出し，その出力を読み込みます．

例題3

```
1  PROGRAM TEST
2    USE FORTRAN_PIPE
3  !
4    IMPLICIT NONE
5  !
6    INTEGER :: istat
7    CHARACTER(LEN=256) :: line
8  !
9    CALL PIPE_OPEN("getfield.py", "r", STAT=istat)
10   DO
11     CALL PIPE_GET(line, STAT=istat)
12     IF (istat > 0) EXIT
13     WRITE(*,*) TRIM(line)
14   END DO
15   CALL PIPE_CLOSE()
16 END PROGRAM
```

これは以下のようにしてコンパイルします．

```
% gcc -c fortranpipe.c
% g95 -c pipe_interface.f95
% g95 -o pipe_test pipe_test.f95 *.o
```

ここで getfield.py を他のプログラムに変える事で，GUI を介して様々な入力を得ることができます．パイプを介したデータのやり取りを行う時は，アプリケーションをフィルター（標準入力から読み込み標準出力に出力）として設計することが重要になります．パイプを使った方式では GUI の部分だけ独立に開発できるという利点があります．Tkinter のルックアンドフィールに飽きたら他のツールキットに乗り換えることも可能です．

7.4 Python から Fortran を呼ぶ

前節ではパイプを介して python と Fortran との間でデータを交換していました．しかし，パイプは読むか書くかの一方向しかサポートしてません．また，大量のデータを送るのには適していません．巨大な配列のやりとりには共有メモリを用いる方法がありますが，今回は Python と fortran との間で相互にデータをやりとりする例として，Python から直接 Fortran のサブルーチンを呼んでみます．

第 3 章で紹介した numpy には f 2 py というツールが含まれています．f 2 py を用いると簡単に python から Fortran を呼ぶインターフェイスを作成できます．**例題 4** の Fortran サブルーチンを python から呼び出すことを考えます．

例題 4

```
4   !     f2py_sample.f95
5   !
6   !
7   !-----------------------------------------------------------------------
8   !
9       SUBROUTINE square(x, y)
10        IMPLICIT NONE
11        REAL(8) :: x
12        REAL(8) :: y
13  !f2py intent(in)  x
14  !f2py intent(out) y
15        y = x**2
16      END SUBROUTINE square
17  !
18  !-----------------------------------------------------------------------
19  !
20      SUBROUTINE VSQUARE(x, y, n)
21        IMPLICIT NONE
22        INTEGER :: n
23        REAL(8) :: x(n)
24        REAL(8) :: y(n)
25  !f2py intent(hide), check(len(x) >= n), depend(x) :: n = len(x)
26  !f2py intent(in) x
27  !f2py intent(out) y
28        INTEGER :: i
29        y(1:n) = x(1:n)**2
30      END SUBROUTINE VSQUARE
```

ここで

```
> f2py -c -m f2py_sample f2py_sample.f95
```

とすれば，Linux 系の OS なら f2py_sample.so が，windows では f2py_sample.dll ができます．-m で指定するのはモジュール名です．ここで

例題 5

```
5   #!/usr/bin/env python
6
7   import numpy
8   import f2py_sample
9
10  f2py_sample.hello(10)
11
12  x = 2.0
13  y = f2py_sample.square(x)
14  print x, y
15
16  x = numpy.arange(0.0, 1.0, 0.1)
17  y = f2py_sample.vsquare(x)
```

```
18  for i in range(10):
19    print "%10.5f %10.5f" % (x[i], y[i])
```

を実行すればpythonからsquareやvsquareを呼び出す事ができます．呼び出すには**例題5**のように

```
import numpy
import f2py_sample
...
y = f2py_sample.vsquare(x)
```

と，f2pyで指定したモジュール名をimportするだけです．**例題4**にあるようにFortranのソースは大文字で書いても小文字で書いても良いです．ただし，pythonから呼び出す時は**例題5**のように関数名はすべて小文字になります．通常のFortranのソースと異なるのは，サブルーチンの引数属性を!f2py INTENT(IN) xなどと指定する必要がある点です．**例題4**では引数が実数，実数配列の場合を示しています．サポートページのf2py_sample.f95には引数が文字列，整数の例もあります．引数属性の書き方(Fortran90の引数属性と似ていますが拡張されてます)，INTENT(OUT)の場合のpythonでの呼び出し方（関数の返り値となる)に注意が必要です．残念ながらf2pyは引数属性指定でREAL(kind=KIND(1.0D0))やREAL(dp)などをサポートしていません．REAL(8)やINTEGER(4)など使用するコンパイラに合った書き方をします．また，INTENT(OUT)属性が複数ある場合は，タプルが返されることに注意します．しかし，モジュール変数へのアクセス，allocatable属性を持ったモジュール変数をpython側からallocateできるなど，pythonからFortranを呼ぶシーンで必要な機能は一通り備えています（構造体入出力をサポートしていないのが残念ですが)．実行速度が必要な部分をFortran化していけば，pythonの柔軟性を保ったままパフォーマンスを高めることができます．マルチコアプロセッサがあたりまえの最近の環境ではOpenMpをサポートしているコンパイラ(gfortran等)により共有メモリ形並列化をすれば，画期的に実行速度が改善される可能性があります．

最後に開発速度という点でpythonからFortranを呼び出すメリットを見てみましょう．サポートページにあるf2py_elliptic.f95は第1種と2種の完全／不完全楕円積分のFortranライブラリ用のelliptic_integral.f95のpythonインターフェイスを作ります．

```
> gfortran -c rp_num_kinds.f95
> gfortran -c elliptic_integral.f95
> f2py -c -m f2py_elliptic f2py_ellip-
tic.f95 *.o
```

とすればpythonインターフェイスの完成です（rp_num_kinds.f95はdp等を定義しています)．筆者の環境（centos 5.1）では，セキュリティの関係でライブラリへのアクセスがブロックされてしまったので，

```
chcon -c -v -R -u system_u -r object_r -
t textrel_shlib_t /home/inagaki/testf2py
/f2py_elliptic.so
```

としてから

```
python f2py_sample2.py
```

と実行しました．楕円積分はscipyなどには含まれていません．このため本やサンプルコードを読みながら自分でコーディングしなければなりませんが，そのサンプルがFortranやCだったり，そもそもFortran/Cではソースコードが手に入ったりします．Pythonで車輪の再発明をする(しかも実行速度の遅い)よりも，既存コードを再利用した方が効率的ですし，それが軽量言語のポリシーでもあります．このようにpythonからFortranを呼び出すと実行速度，開発速度の両方を改善してくれる可能性があります．

7.5 おわりに

この章では，GUIの作成などを通じて軽量言語であるPythonと重量言語のFortranとの連携について説明しました．データ解析を軽量言語で行うことが多くなってきた昨今の環境のなかで，重量言語の意義を考えてみました．軽量言語をデータ解析に利用するユーザが増えたため，相対的に重量言語の出番が減ったように感じますが，絶対値としては重量言語の出番はそれほど変わっていないのではないかと思います．軽量言語はその簡便性からデータ解析にプログラム言語を用いるユーザを増やしたのではないかと思います．本章に示したとおり，軽量言語と重量言語が手を組めば，実行速度，開発速度の両方が改善される可能性があります．データ解析をする機会が増えれば増えるほど，重量言語の必要性は高まるのではないかと思います．今回はPythonを例に用いましたがoctaveもrubyも重量言語インターフェイスを備えています．オープンソースでは実行性能が不安という場合は，ここで紹介した重量言語との連携は一つの選択肢になり得ます．今後，データ解析は言語混在環境で行うことが増えてくると考えられます．今回紹介したf2py以外にもscipyのweaveやruby-inline等で，軽量言語のなかにそのままC関数を書くことができるなど，お手軽に言語混在プログラミングが実現できる環境が整い始めています．

謝辞

本章では，核融合科学研究所鈴木康浩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．