# Python による数値解析教育の実践

箕原辰夫[*1 *2]
Email: *minohara@cuc.ac.jp*

*1: 千葉商科大学政策情報学部政策情報学科
*2: 慶應義塾大学大学環境情報学部非常勤講師

◎Key Words Python, 数値解析, Anaconda

## 1. はじめに

米国でも日本でも、Python 言語を用いた数値解析のプログラミングを行なう教育が普及してきており、そのための様々な書籍も出されている。今年度、慶應義塾大学湘南藤沢キャンパスで春学期の半期 1 コマの講義として数値解析を行なう授業を担当することになった。この報告では、その授業の設計、開発環境、実際の講義内容、および学生の習熟状況などについて報告する。高校では数学Ⅲが必修ではないが、その中で出てくる行列・ベクトルなどについて学んでいない学生が多い。また、新課程では行列自体が高校の数学の教科から外されてしまっている。ところが、機械学習なども含めて、数値解析では行列などの演算が基本になっている。理工系の学部であれば、線形代数を必修科目にすることができるが、湘南藤沢キャンパスのように、総合政策学部と環境情報学部の学生が混在しているような社会系・人文系を含めた学部では大学の数学系の科目を必修にすることはできない。そのような分野でも、研究内容に応じて、コンピュータのソフトウェアを用いて、数値解析を行なっていかなければならない状況があり、このような状況の中で導入教育と行なうべき数値解析の講義の在り方についても考察する。

## 2. 授業の設計

### 2.1 科目の配当

湘南藤沢キャンパスでの 2 学部では、1 年次の学生に必修科目として、情報基礎 I を春学期に、情報基礎 II を秋学期に学ぶ。この中で、JavaScript を Web ページのマッシュアップのために用いるのであるが、問題のあることに、プログラミング言語としての教育がなされていない。そのため、2 年次以降の学生が履修できるオブジェクト指向プログラミング基礎、あるいはスクリプト言語プログラミング基礎などの科目において、再度、最初から「プログラミング言語として」の教育をしなければならない状況になっている。個人的には、米国のコンピュータ科学を教えている主要大学と同様に Python あるいは Java をプログラミングの基礎科目として教えて、その上で JavaScript なり数値計算なりの科目を配当すれば順当な教育が可能であると考えるが、社会系の総合政策学部からの要請もあり、理系で考えられるような順当なプログラミング教育をすることが難しい状況になっているのかも知れない。

今回担当することになった「スクリプト言語プログラム基礎」という科目については、本来は、スクリプト言語の役割は、複数のアプリケーションを連携させることにあり、たとえば Mac OS X を主体として考えるのであれば、AppleScript を用いて Illustrator や Photoshop などを Excel などのソフトウェアと連携させることを教えるべき科目名になっている。ただし、学生の手持ちのコンピュータを用いて実習を行なう授業なので、Windows のノート機を持っている学生がいることもあり、また、授業管轄の担当の先生から、「機会学習に繋げるために数値計算を教えて欲しい」という内容を依頼されたため、Python で数値計算を含んだ授業を設計することとなった。

なお、2 年次以降のプログラミング教育科目として設置されているオブジェクト指向プログラミング基礎 1 コマも担当して開講しており、こちらでも Python でプログラミング教育の基礎を行なっているが、こちらでは tkinter のライブラリを利用して GUI のような内容も教えている。その科目は、この科目の前の時限に同じ教室で開講しているので、学生の中には、2 コマ続けて、その科目とこちらの科目を履修する学生も半数ぐらいいる。

「スクリプト言語プログラミング基礎」として開講されている他の科目では、英語で授業を行なう教員が Python を使って、また授業管轄の担当の先生は Ruby on Rails を使って授業を展開している。

### 2.2 授業構成

履修者の人数は、履修者数を教室の大きさにあわせて 50 名程度に制限したが、最終的には 37 名になった。まずまずの人数であるが、その中の 2 名ぐらいがほとんど出てこない状況で、散発的な休み含めて、毎回の授業の平均の参加人数は 32 名程度となっている。

授業の最初の 10〜15 分程度を利用して、復習的な内容を回答してもらう（コンピュータ利用可能）小テストを実施している。毎年 3 月ぐらいに行なわれる情報科目担当者会議で他の教員からの報告があり、ここ数年、自分の授業の中でも採用することにしているのだが、この小テストは案外に学生に好評で学習内容の振返りをすることができるようである。ただし、そのため、実質の授業時間は残りの 75 分に限られてしまうという問題は残されている。

授業資料は、手持ちのスマートフォンでも見られるような形で、PDF 形式として講義スライドを授業の

Web ページから見られるようにしている[1]。また、授業で入力したスクリプトについては、Web 上の授業サポートシステム（SFC-SFS と呼ばれている）上に翌週までにはアップロードして、休んだ学生やタイピングについて行けなかった学生にが、ダウンロードできるようにしている。

### 2.3 プラットフォーム

コンピュータが備え付けてある実習室で行なうのではなく、学生の手持ちのノート機を持ち込んでもらいそこで実習をしてもらうということで、そのノート機の仕様によってかなり煩わされることがあるが、湘南藤沢キャンパスでは、コンピュータ実習室の標準が iMac になっているという恵まれた状態にあるために、履修者の8割がMacBook のいずれかの機種を持ってきている。残りの２割が Windows 機である。一般の大学であれば、この割合は逆転するのではないかと考えられる。ただし、Python および後述の Anaconda では、プラットフォームに依存しない部分が多い。しかし、Mac OS X が標準だとやりやすい部分が多いのでこのプラットフォーム配分状況には助けられている。

### 2.4 シラバス作成時の大まかな設計

それまで、旧カリキュラムの学生対象に２コマ連続でオブジェクト指向プログラミングの授業を展開してきたのだが、今回は 2014 年度以降に入学した新カリキュラムの学生も対象になり、積極的な学生しか履修しないだろうと考えられ、加えて１コマの授業であるために、かなり進行速度を速めにして次のような授業設計を行なった。

前半７回　Python 言語の大まかな紹介
中盤３回　Num.py、matplotlib の紹介
後半５回　数値計算のアルゴリズムと
　　Sci.py を使った数値解析

しかし、先に述べたように、そもそも１年次のプログラミング教育が次学年以降に応用が利く標準的なプログラミング教育ではないために、プログラミング教育を一から再度やり直さなければならない状況であったので、まったくこの通りには行かなかった。

## 3. 授業実践の結果

### 3.1 学生の高校までの数学教科の履修状態

高校での必修の数学教科は、数学Ⅰおよび数学 A である。この中では、微分・積分も入っていないし、数学Ⅲを履修しないとベクトルや行列なども扱わない。また、新課程では行列も、高校数学から排除されている。また、Python には、標準型として複素数型が用意されているであるが、虚数ぐらいは高校数学で履修しているが、本格的なガウス平面を伴う複素数についてはまったく理解がないことが授業を通して判明した。参考までに、授業でのアンケート結果を以下に示す。ここでの複素数は飽くまでも高校数学の範囲での複素数として考えてもらいたい。

表１ クラスでの高校で履修していない項目の割合

| 履修していない項目 | 割合 |
|---|---|
| ベクトル | 6％ |
| 複素数 | 9％ |
| 行列 | 38％ |
| 整数論（完全数） | 12％ |
| 整数論（ピタゴラス数） | 18％ |
| 整数論（エラトステネスの篩い） | 69％ |

このアンケートの細かな数値は意味を持たない。それよりも大まかな傾向として、旧課程の学生を含めても行列を高校で学んでこなかった学生が４割程度になるという点が大きな問題として映る。社会系も含めて、大学での数値解析では行列演算がその中心になるにも拘わらず、である。また、この数学 A で出てくる素数を求める方式の中で、エラトステネスの篩いは教科書のコメント欄に良く出てくる内容であるが、７割が教科書のコメントまでは学んでいないのが高校数学教育の実情である。

なお、この傾向は、数学１科目の入試を行なっており、その偏差値が一番高い部類に入る慶應義塾大学環境情報学部の学生を約７割含むクラスであるとしても、この程度なのであり、他大学の社会系主体の学部では、ここまで良い数値は期待できないであろう。

### 3.2 開発環境の選定

Python の数値解析ライブラリである Num.py[2]、Sci.py[3]を導入するにあたって、２つの選択肢が存在する。１つは、Python の素の環境に、pip などのインストーラを使って、Num.py、Sci.py、matplotlib[4]を追加でインストールする方式である。今回は、Python の素の開発環境だけを導入するだけに留め、学生にはこの方式を勧められなかった。その原因は、Windows ユーザの学生の存在である。Mac OS X 上からは、元々 Unix 系の OS なので、簡単に導入することができる。しかし、Windows では、Python 上に Num.py や Sci.py のインストールをするのに、セキュリティ的な制約もあり、かなり大変であることがわかった。ただ、こちらでやりたい学生もいるために、講義スライドでは、Windows でのインストールの仕方についても説明した。

もう１つの選択肢は、Anaconda[5]の開発環境である。この場合は、Num.py、Sci.py だけでなく、ほとんどの科学技術計算のライブラリがまとめてインストールされる。加えて、Anaconda についている開発エディタである spyder が使いやすい（ただし、Mac OS X からはアプリケーションの形にはなっていないために、Anaconda Navigator というアプリケーション、あるいはターミナルから起動する必要がある）ので、これを Windows も含めた学生にインストールを推奨することとし、授業での標準の開発環境とした。また、Anaconda についている Anaconda Navigator では、それぞれのモジュールに対応してアップデートを行なうことも可能である。そのため、Anaconda インストール時には Python 3.6.0 であったものを、この報告の執筆時点で

の最新版の Python 3.6.1 にアップデートすることができた。また、Anaconda には、Numba[6]の高速実行環境がついていきている。そのため、授業では Numba の JIT（just in time）コンパイラを用いて、高速に計算を求める事例を実行することが可能となった。

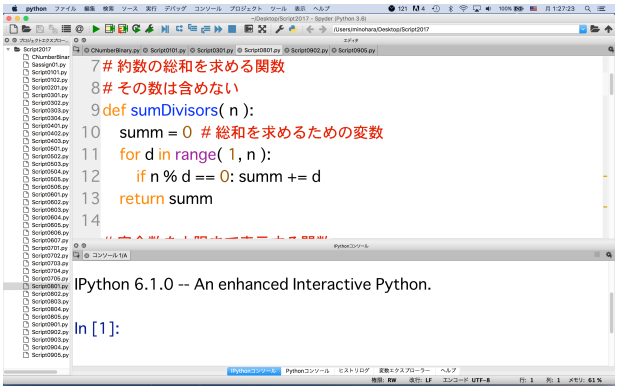


図 1　spyder の開発環境

spyder では、Python の元々の IDE では入力が面倒であった（特に Mac OS X 版では Active tk/tcl のバージョンを新しくする必要があった）日本語入力が問題なく可能となっている。また、行番号も表示されており、メニューなども日本語対応していて学生にも受けが良い。加えて、IPython という Python のインタープリタがコンソールパネルに表示されており、そこで対話的に計算もできるので、非常に使い勝手が良い。

### 3.3　実際の授業回での講義内容

教科書については、和訳された Sci.py のチュートリアル[7]を指定した。これの原文の英訳は、Sci.py のドキュメントのサイトにある[8]。この中でも、1/3 程度は Python の言語の説明に用いている。また、数値解析については、複数の教科書[9],[10]を参考書として使用することとした。この報告を執筆している時点では、第 9 回までの講義回が終了している。第 10 回以降は、今後の予定である。

表 2 各授業回の内容

| | |
|---|---|
| 第 1 回 | Python IDLE の導入、日本語入力環境 |
| 第 2 回 | Anaconda 開発環境の導入、各原始型のリテラル |
| 第 3 回 | 文字列演算、入出力、指数・複素数演算、型の変換 |
| 第 4 回 | 文字列・タプル・リストのインデックス・スライス記法 |
| 第 5 回 | 制御文（if 文・論理式・if 式・繰返し） |
| 第 6 回 | リストと for 文・その他の制御文 |
| 第 7 回 | 関数の定義と呼出し、整数論のための関数 |
| 第 8 回 | 再帰関数、高階関数 |
| 第 9 回 | リストの走査、シャッフル、リストに対する高階関数（map・filter・reduce） |
| 第 10 回 | 行列計算、Num.py・matplotlib の導入、辞書・集合 |
| 第 11 回 | 実数計算、テイラー展開、数値積分 |
| 第 12 回 | ニュートン法、π の求値 |
| 第 13 回 | ガウスの消去法、Sci.py による連立一次方程式の解法 |
| 第 14 回 | 補間、常微分方程式の数値的解法 |
| 第 15 回 | フーリエ変換、Sci.py を使った高速離散フーリエ変換の利用 |

このように言語の説明が授業回の 10 回に亙り、本格的な数値計算に入るのが 11 回目以降に遅れているのは、学生にとって本格的なプログラミング教育がこの授業が最初であることもさることながら、Python 自体の言語構造の奥行きが深いことにも起因するものである。リスト・タプル構造や、辞書構造を言語構造の中に持ち、言語記述能力が Lisp 並みに優れているからである。たとえば、エラトステネスの篩いを使って 2 から 100 までの素数を求める方法を、次のようにリストと filter 高階関数を使って記述することも可能である。

```
upper = 101
plist = [ n for n in range( 2, upper ) ]
def isnotmultiply( n ):
    return n % divisor != 0 or n == divisor

for divisor in range( 2, int( upper ** 0.5)+1 ):
    plist = list( filter( isnotmultiply, plist) )
print( plist )
```

また、この講義ではクラス定義などのオブジェクト指向プログラミングの内容が省かれている。そちらの内容は、この講義の前の時限に行なっている「オブジェクト指向プログラミング基礎」の授業の中で扱うこととした。連続で受講している学生は、その内容も学ぶことができる。

### 3.4　収束速度と精度

非線形方程式の解を反復法で求めるのに、1 次収束の 2 分法やはさみうち法で求めるのと、2 次収束で求められるニュートン法や 3 次収束のハーレー法で求めるのでは、明らかに 1 回の反復で求められる桁数の差が歴然としている。前年の発表でも指摘したが、解を求めるための収束速度の概念は重要である。残された授業回では、この内容を入れていこうと考えている。

一定桁数の π の値を求めるのには様々な解法がある。これらについても比較して、数値計算においては収束度が重要な意味を持つことを教えたいと考えている。実際にどんな方法があるか調べて比較することを課題にする予定にしている。特に、Python では、decimal パッケージを使うと実数の仮数部を指定した多桁精度で求められる。このライブラリは、言語と親和性が高く、通常の演算子を用いることができる。以下のプログラムでは、ハーレー法を使った 20 回ほどの繰返しで 2 の平方根を 1000 桁の精度で求めることができる。

```
from decimal import *
getcontext().prec = 1000
```

```
n = 2
def f( x): return x ** 2 – n
def ff( x): return 2 * x
def fff( x ): return 2

def halley( x ):
    return x - 2 * f( x ) * ff( x ) / ( 2 * (ff( x ) ** 2) -
    f( x ) * fff( x ) )

value = Decimal( 2 )
for i in range( 20 ):
    value = halley( value )
print( value )
```

### 3.5 Numba による高速化

Anaconda を開発環境として採用したため、Numba の高速コンパイル環境が何の追加設定も必要なく利用することが可能となった。整数論の教科書に載っていた完全数を 2 進数から求める方法であるが、完全数かどうかを求める関数に@jit 修飾子を付けて求めてみた。これを授業の中でも扱う予定にしている。完全数の 8589869056 を計算するときに、だいたい、17 倍ぐらい高速化されるのを time パッケージの関数を用いて時間計測して確認した。

```
from numba import jit

@jit
def isComplete( n ):
    summ = 0
    for m in range( 1, n / /2+1 ):
        if n % m == 0: summ+= m
    return summ == n

cbin="1"
for n in range( 16 ):
    cbin = "1"+cbin+"0"
    cvalue = int( cbin, base=2 )
    print( cvalue, end=": " )
    print( "完全数" if isComplete( cvalue ) else "" )
```

## 4. 考察

毎回授業の冒頭で行なっている小テストの結果では、点数が良い学生が多いために、ほとんどの学生に授業内容は理解されていることを示しているが、実際に感想を書かせてみると、８割ぐらいの学生が「難しい」という感想を抱いているようである。残りの２割ぐらいの学生は、もともとプログラミング能力があり、早く数値解析に入って欲しいと考えている。しかしながら、大学の数学を履修していない学生の割合が多いために、数値解析に入った時点で、かなり大学の理系で１年次～２年次に学ぶ数学の内容を講義する要素が増えるのではないかと予想される。このため、今後は、授業後に 1 コマぐらい延長しないとすべての内容を扱えないのではないかと覚悟している。

非常勤として勤務している他大学のカリキュラム体系には口は出せないが、本来の科目設計として、やはり「スクリプト言語プログラミング基礎」は、Python の言語を使った強力なプログラミングスタイルを教えるべきであり、その後継科目として「プログラミングによる数値解析」の授業を持ってくるべきであると思う。半期 1 コマの中で、両方を入れるのは、詰め込み過ぎであると思う。なお、本務校の千葉商科大学では、数値計算を行なうプログラミング科目が、その科目を教えていた教員が定年でいなくなったと同時に、科目自体がなくなってしまってから久しい。入学してくる学生が、そのような計算ができるために必要な高校数学の内容を学んでこないからである。

情報処理学会の大学コンピュータ教育の委員会で、2007 年に J07[11]のカリキュラムを作成したが、数値計算・数値解析については、その中では必修の科目とされていなかった。しかしながら、ACM の CS2013[12]のカリキュラムや、今後策定される J17 のカリキュラムの中では、必修の 1 科目として準備されている。今後の大学のカリキュラムの中でも、Python を用いた形での数値解析・数値計算の科目が標準の必修科目として入ってくることが予想される。そのときには、やはりスクリプト言語の基礎科目を先修科目として用意すべきではないかと考える次第である。

## 5. おわりに

理工系の学部だけでなく、社会系の学部においても、統計などの調査を行なう際には、一般的に SAS、SPSS や R などの統計解析ソフトウェアを使う。また、Python においても pandas などのライブラリが統計解析にも使われる。そのためのベクトル・行列演算の知識は、高校数学に必要と思われる。今後の高大連携教育で扱う内容の議論に関わるが、 ある程度の高校数学教育の拡大化は、今以上に求められるのではないか。大学では、高校数学までで学んだ内容をプログラミングとして表現しなければならないからである。

### 参考文献


(1) 箕原辰夫：“スクリプト言語プログラミング基礎・講義スライド”, http://web.sfc.keio.ac.jp/~minohara/lectureslide/computation（2017）.
(2) “Num.py,” http://www.numpy.org, （2017）.
(3) “Sci.py, https://www.scipy.org, （2017）.
(4) “matplotlib,” http://matplotlib.org, （2017）.
(5) “Anaconda,” https://www.continuum.io/downloads, （2017）.
(6) “Numba,” http://numba.pydata.org, （2017）.
(7) Gaël Varoquaux et al. editors：“Sci.py Lecture Notes 和訳,” http://www.turbare.net/transl/scipy-lecture-notes/index.html, (2015).
(8) Gaël Varoquaux et al. editors：“Sci.py Lecture Notes, ” http://www.scipy-lectures.org, (2016).
(9) 中久喜健司：“科学技術計算のための Python 入門,” 技術評論社, ISBN 4-7741-8388-6 (2016).
(10) 阿部剛久他：“科学技術系の数値解析入門,” 昭晃堂, ISBN 4-7856-7024-X (1992).
(11) 情報処理学会: 情報専門学科カリキュラム標準 J07, https://www.ipa.go.jp/files/000024071.pdf (2008) .
(12) ACM: Computer Science Curricula 2013, http://www.acm.org/education/CS2013-final-report.pdf (2013) .