プログラミング教育と可視化
—数値天文学を事例にして—

伊藤 誠[†]

1 はじめに

プログラムも言語の一種であるから、プログラムを学習するためには文法をきちんと覚えなければならない。そのためには多数のプログラムを作成し実行することが重要である。そして、最終的に自分で考え、自身が必要とするプログラムを自らの手で作成できるようになればプログラミングの学習の目的を達成したことになる。

ただ、「最終的に自分で考え自身が必要とするプログラムの作成」と一言で言うのは簡単だが、プログラム初学者にとって基本を学んだ後で、次に何かを始めようと思っても、何をすればよいのかを考えるのはなかなか難しい。文法だけ覚えても具体的な問題に対処するプログラムを作成するのは初学者には至難の業である。具体的な問題に対処するには、文法を学んだあとに、基本的なアルゴリズム・プログラミング技法を理解しなければならない。そのためには何か適切な教材があればそれに越したことはない。

筆者の専門分野は天文学 (宇宙物理学、物理学) であるので、そちらに関連した題材がないものかと気にかけていた。そんなとき「パソコンで宇宙物理学：計算宇宙物理学入門 (川端潔訳)」[1] という本があることを知った (以後「PC で物理学」)[1]。タイトルだけを見ると天文学・宇宙物理学に関する初歩的なプログラミングの入門書に思える。実際、内容を確認してみると「PC で物理学」は全部で 13 章からなり、代表的な数値計算法から、天文学・宇宙物理学の初歩的内容を含んでいて、また具体的なサンプルプログラムも完全な形で掲載されている。したがってこのテキストは一見するとプログラミング及び天文学初学者にとって次のステップに進むための適切なテキストになり得る可能性があると考えられる。しかし「PC で物理学」に掲載されているプログラムを作成・実行しようとすると様々な問題点があることが分かった。

問題点については次節にまとめることとし、これらの様々な問題点を解決し、タイトルにあるように「可視化」も含めてこの「PC で物理学」が教育あるいは自学自習の良い教材となりうるかどうかを考察し、「プログラミング教育＋天文学」の教材として利用可能かどうかを探る事例研究を行うことが本稿の目的である。

また世には専門家ならずとも、天文に興味を持つ人は数多いるであろう。そのような人たちの目的に叶うテキストがあれば素晴らしいことであるが、この「PC で物理学」がその目的に叶うかどうかも考察してみることにする。

2 「PC で物理学」の持つ問題点

2.1 問題点その 1: 本の出版年が古いこと

邦訳は 2008 年が出版年であるが、原書は 1994 年である。つまり原書は今から四半世紀も前に出版されたことになる。近年の天文学・宇宙物理学の進歩は目覚ましいものがあり、四半世紀前に書

[†]大阪産業大学 経済学部 経済学科 教授
草 稿 提 出 日 10 月 15 日
最終原稿提出日 10 月 15 日
[1]原書名「ASTROPHYSICS WITH A PC: An Introduction to Computational Astrophysics」(Paul Hellings 著)

かれたことは、時代遅れとみなされてもおかしくはない。ただ誰もが即専門家ではなく、初めは誰しも初心者である。全く天文学を知らない初学者にとって「PC で物理学」の内容は決して時代遅れのものではないし、天文学を志す者にとって知っていて然るべき内容がきちんと含まれている。またプログラミングに関しても、いまでも通用する基本的なアルゴリズムがしっかりと解説されている。したがって出版された年はかなり昔になるが、内容的には特に問題はならない。

2.2　問題点その 2: プログラムが QuickBASIC で書かれていること

1994 年頃というと Intel Pentium プロセッサが PC で用いられ始めた頃である。すでに Linux (Slackware 等) は利用可能であったが、Windows3.1 が主流であったと記憶している。したがって当時の PC を用いたプログラムが、QuickBASIC で書かれていることはやむを得ないことかもしれない。ただし邦訳は訳者によって FreeBASIC で書き換えられている。FreeBASIC は現在でも入手可能であるが、実際 BASIC を主力言語として学習しようとする初学者は少数派であろう。また決定的な問題として、ソースプログラムの記述が古いため、プログラムを実際動かそうとしてみたがコンパイルエラーとなり (FreeBASIC はコンパイラ言語である) 動作しなかった。FreeBASIC を今更勉強して、エラーの原因を探ってプログラムを書き直そうとは思わないので、プログラムに関しては、そのまま本に記載されている FreeBASIC プログラムを実行するのは諦めるしかないようである。

したがって、このままでは「PC で物理学」は、少なくとも筆者にとっては無用の長物になってしまう。内容的には問題のないこの本を無用の長物で終わらせない解決策はないものであろうか。

その解決策について、次の「問題点その 3」も含めて次小節で改めて探ることとする。

2.3　問題点その 3: プログラムが数値結果しか出力しないこと

BASIC には基本的な描画の機能が含まれるのだが、「PC で物理学」に掲載されているプログラムは数値結果しか出力されない。プログラムを動かし、何らかの計算結果が出力された場合、最も重要なことはその計算結果をどのように考察・解釈するかであろう。しかしコンピュータが出力する数値や文字の羅列を眺めているだけでは何も分からない。グラフや図にして考察することによって、新しい知見が得られるのである。これは専門分野を問わず当然のことであろう。「PC で物理学」には確かに図は載っているが、代表的な計算結果についての図しか描かれていないのである。

原著者による「日本語版の読者へ」という「まえがき」には、PC の進歩や利用言語が Java に変化しており、グラフィックスの機能も向上してきたことに触れられているが、「PC で物理学」の目的はあくまでも「宇宙物理学」であって、計算手法が重要であると書かれている。したがって日本語版で BASIC から Java に書き換えられているわけではないし、当然グラフィックスの機能が追加されているわけでもない。確かに数値結果を得るための手法も重要であるが、物理現象を理解するにはそれを可視化することの重要性も忘れてはならない。「PC で物理学」にはせっかくプログラムが完全な形で掲載されているのだから、色々初期条件を変えたりパラメータを変えたりしてその結果がどのように変化するのかを作図・描画 (可視化) して考察することができれば、「PC で物理学」の利用価値もより増すであろうことは想像に難くない。

更に PC が進化した今の時代、「PC で物理学」に掲載されているプログラムであれば、実行しても待つことなく結果が得られ、即座に可視化することが可能である。現在ではこのような可視化機能 (作図・描画) も併せ持ったプログラミング言語は Java に限らず多数ある。

そこで「問題点その 2」も含めた解決策であるが、「PC で物理学」に掲載されているプログラムを別のグラフィックス機能を持つプログラミング言語で記述し直し、可視化の機能をプログラムに追加するのである。そうすば「PC で物理学」を現在でも利用可能な教材となりうる可能性が出てくる。

次節では実際に「PC で物理学」に掲載されているいくつかの BASIC プログラムを、別なプログラミング言語に移植が可能であるかどうか、また作図機能を追加することにすることにより、可視化の有用性を検証する。

3　プログラムの移植と可視化

さて、簡単に別言語に移植すると言っても数多あるプログラミング言語のうち、何を選択するかは重要な問題である。原著者が言う Java も一つの選択肢ではあるが、Java が必ずしも初学者向けの言語とは思われない。筆者の趣味を押し付けるのも問題があると思われるが、ここでは最近筆者がよく利用している Python3[2],[3],[4] と MATLAB[5],[6],[7] を利用して「PC で物理学」のプログラムを書き換えてみることにする。付け加えると Pythoh3 は近年非常に注目を浴びているプログラミング言語である。

3.1　常微分方程式

まず「第 1 章 代表的な数値計算法」にある微分方程式を解くプログラムを移植する。常微分方程式の解法は、天文学・宇宙物理学の直接的な題材ではないが、天文学・宇宙物理学に限らず、様々な問題を解くための最も基本的な数値計算技法である。「PC で物理学」には常微分方程式を解く方法として**オイラー法、中点法、予測子修正子法**、及び **4 次のルンゲ・クッタ (R-K) 法**が載っている。これらの手法を用いて「PC で物理学」では以下の常微分方程式

$$y' = -2yx \qquad (\text{初期条件} : x_0 = 0,\ y_0 = 0)$$

を解いている。厳密解は

$$y = \exp(-x^2)$$

である。「PC で物理学」にはこれら 4 つの手法で解いた結果が数値表として掲載されている。以下の表は「PC で物理学 (P13)」からの引用である。

表 1-1

x	y(オイラー)	y(中点)	y(予測修正)	y(R-K)	y(厳密)
0.00	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000
0.10	1.00000000	0.99000000	0.99000000	0.99004983	0.99004983
0.20	0.98000000	0.96059700	0.96069600	0.96078944	0.96078944
0.30	0.94080000	0.91352775	0.91381404	0.91393117	0.91393118
0.40	0.88435200	0.85149921	0.85204021	0.85214377	0.85214379
0.50	0.81360384	0.77792968	0.77876475	0.77880078	0.77880078
0.60	0.73224346	0.69663603	0.69777321	0.69767639	0.69767633
0.70	0.64437424	0.61150711	0.61292399	0.61262661	0.61262639
1.60	0.06751343	0.07812552	0.07929913	0.07731173	0.07730474
1.70	0.04590914	0.05646913	0.05744429	0.05558359	0.05557621
1.80	0.03030003	0.04006485	0.04085438	0.03917137	0.03916390

さてこの表から何を読み取ることができるであろうか?『「オイラー法」は「厳密解」からちょっとズレが大きくて、「ルンゲ・クッタ法」はかなり正確なようだ』と思う人もいれば、あるいは『所詮「数値解」は近似に過ぎない。結局のところ「厳密解」からはズレている』と思う人もいるであろう。では実際どの程度のズレ (誤差) なのだろう。数値だけ見ていても実感が湧かないであろう。そこで結果をグラフ (可視化) にしてみるとよい。残念なことに「PC で物理学」には、表 1–1 をグラフ化したものは載っていない。

3.2　オイラー法

以下のプログラムは「PC で物理学」に掲載されている「オイラー法」の BASIC プログラムを Python3 で書き直し、結果をグラフとしてプロットするようにしたものである。プログラム内の `# plot` 以下が筆者が付け加えた描画のプログラムである。作図には matplotlib というモジュールを用いている。また数学関数に関しては numpy モジュールを用いた。作図以外の部分はほぼ元のプログラムを忠実に書き換えてある。ただし注釈等は省略した。元の FreeBASIC プログラムは「PC で物理学」を参照して欲しい。

```
import matplotlib.pyplot as plt
import numpy as np

def f(x, y):
    return -2 * y * x

x1 = float(input('Initial condition x(0) = '))
y1 = float(input('Initial condition y(0) = '))
dx = float(input('Stepsize          dx   = '))
n  = int(input('Number of iteration to be computed = '))
print("x = {:.4f}, y = {:.4f}".format(x1, y1))

x = []
y = []
x.append(x1)
y.append(y1)

for i in range(n):
    # compute new iteration
    x1 = x[i] + dx
    y1 = y[i] + dx * f(x[i], y[i])

    # display new iteration
    print("x = {:.4f}, y = {:.4f}".format(x1, y1))

    # prepare for next iteration
    x.append(x1)
    y.append(y1)

# plot
X = np.linspace(x[0], dx * n, 100)
Y = np.exp(- X ** 2)
plt.plot(X, Y, label = "exact")
plt.plot(x, y, "o", label = "numerical")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Euler's method")
plt.legend()
plt.show()
```

前頁のプログラムを実行すると、初期条件・パラメータを尋ねて来るので

```
Initial condition x(0) = 0
Initial condition y(0) = 1
Stepsize          dx   = 0.1
Number of iteration to be computed = 20
```

として実行した。画面に計算結果が表示されたあと、図 1 が描画される。

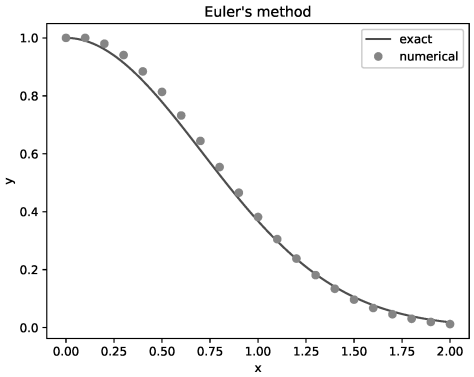

図 1: オイラー法

図 1 から明らかなように、「厳密解 (実線)」と「数値解 (○印)」はかなりズレていることがよく分かる。表 1–1 の数値の違いがどの程度のものであるか、視覚的に認識できたわけである。

3.3　4 次のルンゲ・クッタ法

では次に常微分方程式を解く場合によく用いられる「ルンゲ・クッタ法」のプログラムを書き換えることにする。ここでは言語として MATLAB を用いた。MATLAB には標準で作図機能が備わっているので、特に作図のための機能を追加する必要ない。`%plot` 以下が、描画のために追加した部分である。

```
clear x  y;

f = @(x, y) -2 .* y .* x;

x(1) = input('Initial condition x(0) = ');
y(1) = input('Initial condition y(0) = ');
dx = input('Stepsize          dx   = ');
n = input('Number of iteration to be computed = ');
fprintf("x = %7.4f, y = %7.4f\n", x(1), y(1));

for i = 1: n
    % compute the predicted new iteration
    x1 = x(i) + dx;

    k1 = dx * f(x(i), y(i));
    k2 = dx * f(x(i) + 0.5 * dx, y(i) + 0.5 * k1);
```

```
    k3 = dx * f(x(i) + 0.5 * dx, y(i) + 0.5 * k2);
    k4 = dx * f(x(i) + dx, y(i) + k3);

    y1 = y(i) + (k1 + 2 * k2 + 2 * k3 + k4) / 6;

    % display new iteration
    fprintf("x = %7.4f, y = %7.4f\n", x1, y1);

    % prepare for next iteration
    x(i+1) = x1;
    y(i+1) = y1;
end

% plot
fplot(@(x) exp(-x .^ 2), [x(1), dx * n])
hold on
plot(x, y, 'o')
title('Runge-Kutta method')
xlabel('x')
ylabel('y')
legend('exact', 'numerical')
```

結果を図 2 に示す。実行条件は「オイラー法」の場合と同一である。

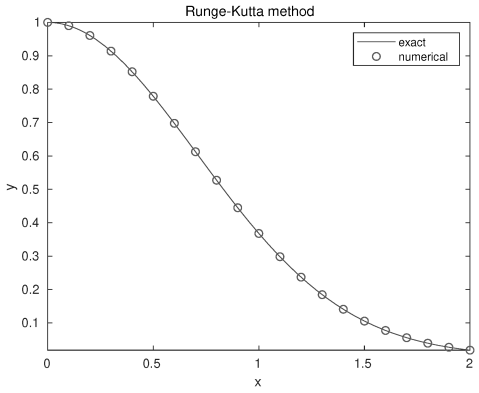

図 2: ルンゲ・クッタ法

先に示した表 1–1 では多少数値的に誤差が見て取れたが、グラフにしてみると「厳密解 (実線)」と「数値解 (○印)」は完璧に一致していることが分かる。つまりグラフにしたとき、数値的ズレを人間は認識することができないのである。「数値解」はもはや「厳密解」と言っても過言でない。図 2 では「数値解」を○印で示してあるが、これを実線で表示すると「厳密解」と「数値解」は重なってしまい区別がつかない。このような一致の様子は表 1–1 より図 2 を見た方がより明らかとなる。ここに可視化の大きな意味がある。

「オイラー法」や「ルンゲ・クッタ法」は、基本的な数値計算のテキストには必ずと言っていいほど載っている常微分方程式の代表的な解法である。その精度についても多くのテキストで議論されているが、存外それを可視化して比較しているテキストは少ないようである。たとえ図が載っていたとしても、その図の描き方のプログラムまで言及しているものはない。

ここでは厳密解が求まる問題を例に挙げたが、解析的に解けない問題でも、ルンゲ・クッタ法を用いれば具体的でかつ正確な解が得られることが納得できるであろう。

さらに注目すべきこととして、ルンゲ・クッタ法は常微分方程式を高精度に数値的に解くための常套手段であるにもかかわらず、「ルンゲ・クッタ法」に由来する本質的な部分は、わずか 8 行しかないことである。「精度の高いプログラム = 複雑で長いプログラム」であるとは限らないことは興味深い。

ここまで常微分方程式の代表的な解法について見てきた。初学者でない人にとっては、あまり新味が感じられないかもしれないが、「PC で物理学」はこのような初歩的な数値計算から順に書かれているのである。では次に、天文学関係のプログラムを 1 つ取り挙げることにする。

3.4　制限 3 体問題

物理学を学んだことがある人は知っていることであるが、重力 2 体問題は厳密に解くことができ解析解を求めることができる。しかし 3 体問題になるともはや解析解を求めることはできない。従って数値的に解く以外手段がないのであるが、3 体問題を数値的に解こうとすると、衝突系であるため色々と厄介な問題が生じてくる (例えばハードバイナリ形成によるタイムスケールの問題等)。

3 体問題のうち 3 体目の質量が、残りの 2 体に比べて無視できる場合を「制限 3 体問題」と言う。「PC で物理学」では第 4 章で制限 3 体問題 (太陽＋木星＋トロヤ群小惑星がなす典型的な制限三体問題) を扱っている。ここではその制限 3 体問題のプログラムを Python3 で書き換え、木星軌道上にあるトロヤ群小惑星 [8],[9] の軌道計算を実行し、結果を可視化することにする。以下に書き換えたプログラムを示す。同様に `# plot` 以下が筆者が追加した部分である。

```
import matplotlib.pyplot as plt
import numpy as np

def effes(x ,y, u, v, mu):
    r1 = np.sqrt((x - mu) * (x - mu) + y * y)
    r2 = np.sqrt((x + 1 - mu) * (x + 1 - mu) + y * y)
    fu = -(1 - mu) * (x - mu) / r1 ** 3 \
         - mu * (x + 1 - mu) / r2 ** 3 + x + 2 * v
    fv = -(1 - mu) * y / r1 ** 3 - mu * y / r2 ** 3 + y - 2 * u
    return fu, fv

n = 201
x = np.zeros(n+1, dtype = 'float64')
y = np.zeros(n+1, dtype = 'float64')
u = np.zeros(n+1, dtype = 'float64')
v = np.zeros(n+1, dtype = 'float64')

mu   =  0.000953875    #  mu   = float(input('Mass parameter mu         : '))
x[0] = -0.509046125    #  x[0] = float(input('Initial conditions : x(0) : '))
y[0] =  0.883345912    #  y[0] = float(input('Initial conditions : y(0) : '))
u[0] =  0.0258975212   #  u[0] = float(input('Initial conditions : u(0) : '))
v[0] =  0.0149272418   #  v[0] = float(input('Initial conditions : v(0) : '))
dt   =  0.4            #  dt   = float(input('Time step                 : '))

t = 0
print('  t        x           y           u           v')
print('-------------------------------------------------------------')
print("{:4.1f} {:12.9f} {:12.9f} {:12.9f} {:12.9f}\n" \
      .format(t,  x[0], y[0], u[0], v[0]))
```

```
t = dt
for i in range(n):

    # next block computes half step values of Caucy method
    x1 = x[i] + 0.5 * dt * u[i]
    y1 = y[i] + 0.5 * dt * v[i]
    (fu, fv) = effes(x[i], y[i], u[i], v[i], mu)
    u1 = u[i] + 0.5 * dt * fu
    v1 = v[i] + 0.5 * dt * fv

    # next block computes new state i + 1 of Caucy method
    x[i+1] = x[i] + dt * u1
    y[i+1] = y[i] + dt * v1
    (fu, fv) = effes(x1, y1, u1, v1, mu)
    u[i+1] = u[i] + dt * fu
    v[i+1] = v[i] + dt * fv

    # new state is shown on screen
    print("{:4.1f} {:12.9f} {:12.9f} {:12.9f} {:12.9f}\n" \
          .format(t,  x[i+1], y[i+1], u[i+1], v[i+1]))

    t = t + dt

# plot
plt.figure(figsize = (5, 5))
plt.xlim(-1.1, 0.1)
plt.ylim(-0.1, 1.1)

plt.plot(x, y, '-b')

t = np.linspace(0, 2 * np.pi, 100)
cx = np.cos(t)
cy = np.sin(t)
plt.plot(cx, cy, ':k')

plt.plot(0, 0, 'oy', markersize = 15)
plt.plot(-1, 0, 'oc', markersize = 10)
cx = np.cos(2 * np.pi / 3)
cy = np.sin(2 * np.pi / 3)
plt.plot(cx, cy, 'om', markersize = 5)

plt.text(-0.1, 0.05, 'Sun')
plt.text(-0.95, 0.05, 'Jupiter')
plt.text(cx + 0.05, cy - 0.1, 'L4')

plt.show()
```

ここでは「PC で物理学」(P76) に載っている軌道 A の計算を行うためのパラメータを設定している。初期条件は桁数が多く、プログラムの実行時にキーボードから入力するのは大変であるので、上記プログラムでは初期条件をプログラム内に直接記述した。キーボドから入力するように変更したい場合は、初期条件を設定してある部分を、右側注釈に書いてある `input` 文を含む部分と入れ替えて欲しい。

太陽と木星の重力が釣り合う点をラグランジュ点と言い、全部で 5 つ (L1～L5) ある。そのうち 3 個 (L1～L3) は不安定であるので別の小天体が安定して存在することはできないが、太陽–木星系の場合、残りの 2 つは安定した周期軌道が存在するため、トロヤ群小惑星のようにラグランジュ点 L4(あるいは L5) の周りを周期運動する軌道を描くことが可能になる。その様子を視覚化したものが図 3 である。

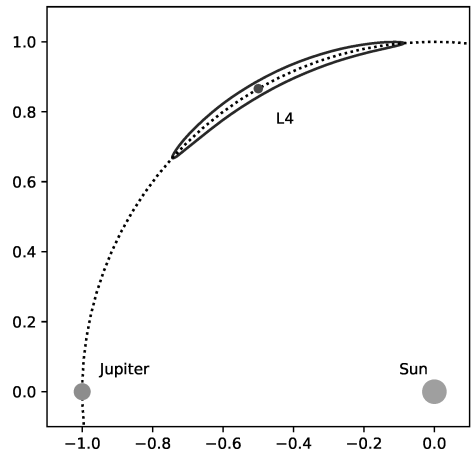

図 3: 制限 3 体問題

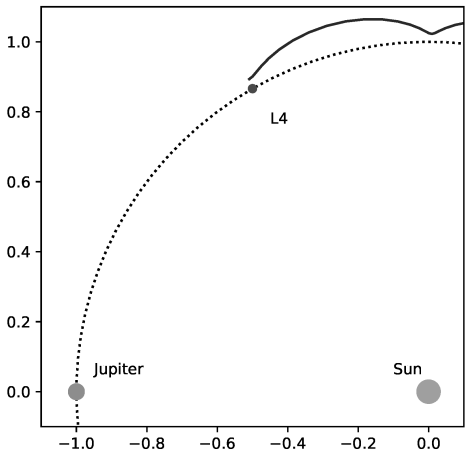

図 4: 制限 3 体問題-その 2-

ちなみに木星・太陽・L4 のなす角は 60 度である。つまりこれら 3 点は正三角形をなしている。図 3 で注意してほしいのは、木星 (jupiter) が点線に沿って太陽の周りを公転するのではないことである。この図は太陽 + 木星の静止系で見た図であり、実際は木星と太陽とトロヤ小惑星が図の位置関係を保ちつつ全体として回転するのである。

この制限 3 体問題で、小惑星がどのような軌道を描くのかは、数値を見ただけでは決してわからない。L4 との関係も合わせ、可視化をすることによって初めて認識することが可能となる。可視化の重要性がここでも示されているのである。

また、このプログラムは惑星の 1 周期分の軌道を計算するために全部で 201 回の繰り返し (イテレーション) をしている。実際の BASIC プログラムは 20 回繰返す毎に計算を継続するかどうかを確認する仕様になっている。上記で書き換えたプログラムを現在の PC で実行すると一瞬で結果が表示される。四半世紀前の PC との性能差がよく分かる。

ここで用いた初期条件はラーベ (E.Rabe) が 1961 年に考案した方法で求めた多くの初期条件のうちの一つである。そこで、初期条件を少し変えてみて、もう一つ計算を実行してみることにする。例えば初期条件を

```
y[0] =  0.883345912
```

から、

```
y[0] =  0.893345912
```

に変更して実行した結果を図 4 に示す。(計算の精度が影響しているかもしれないが) この小惑星は、初期位置を y 方向に 0.01/0.883 ∼ 1% 程度わずかにずらしただけで、ラグランジュ点 L4 の周りに安定した軌道を描かないことが分かる。つまりトロヤ群小惑星は奇跡的な初期条件の下で形成されたことがこの数値計算から推察できる。

このように初期条件を変えて、小惑星の軌道がどのように変わるのかを即座に計算し、可視化することができるため、初期条件の影響を容易に視察することが可能となる。このようなことは四半世紀前では考えられないことである。

4　まとめ

前節では、3 つの例 (オイラー法、ルンゲ・クッタ法、制限 3 体問題) を挙げた。「PC で物理学」には全部で 13 章あるので、まだ例として挙げることのできるプログラムは多数あるが、以下の結論を導くにはこれら 3 つの例で十分であろう。

まず第一に可視化の有用性を示すことができた。数値を眺めただけでは決して分からないことも、可視化することによって一目瞭然となる。まさに百聞は一見にしかずである。人の目を通した認識力は偉大なものがある。可視化を通じたプログラミング教育は非常に重要なものとなることが上記の例からも明らかである。可視化の機能を備えたプログラミング言語を教育の現場で採用することは実に重要なことであろう。

次にプログラミング言語の基礎を習得したものが、次のステップに進む場合、この「PC で物理学」は有効な教材となり得るのだろうか。これが本事例研究の主目的である。

一旦プログラミング言語 (ここでは Python3 と MATLAB) の文法を習得していれば、元の BASIC プログラムが「PC で物理学」に完全な形で掲載されているので、プログラムを別のプログラミング言語を用いて書き換える作業はそれほどの困難はない。作図のためのプログラムは自身で作成しなければならないが、こちらも作図方法を学習していれば、さほどの困難は伴わないであろう。プログラムを自身が習得した言語に書き換える作業は、実際に行ってみると、うろ覚えの知識を定着させるには非常に有効である。「PC で物理学」は、『BASIC で書かれたそのままでは動かないプログラムが載っているつまらないテキスト』ではなく、『プログラミング言語を学習するための、とても適切な教材』となり得るのである。また「PC で物理学」を用いると、プログラムを書き換えることによってプログラミングの知識も定着させると同時に、天文学の基礎知識を習得しつつ視覚化のためのプログラムを追加することにより、天文現象を視覚的に認識することが可能となる。また図 3、4 で示したように、PC の高速化と相俟って、様々な初期条件・パラメータを容易に試すことができ、その結果を即座に視覚的に確認することができるため、天文学の初学者にとって、この「PC で物理学」は現在でも天文学の格好の教材になると考えられる。

ではプログラミング言語は何を選択すればよいのであろうか? 筆者は Fortran77 からプログラムの学習を始めた。残念ながら Fortran 言語は数値計算に特化した言語であるため、作図機能は標準では装備されていない。学生の頃はカルコンプと呼ばれるプロッタ用のサブルーチン群を用いて、Fortran77 で作図用プログラムを別途作成した。近年の計算機環境でも、計算を何らかのプログラム言語で行い、別のソフト (例えば gnuplot) 利用して作図するという方法もある。しかし超大計算 (長時間かかる計算) を行うのでなければ、計算と作図を別途行うのは 2 度手間であろう。昨今の PC の高速化を考えると「PC で物理学」に掲載されているプログラムであれば、作図機能を備えたプログラミング言語を用いてすぐに結果を確認することができる。

MATLAB は最初から言語に作図の機能を標準に兼ね備えているので、言語の学習と合わせて視覚化の手段も習得することが可能である。しかし MATLAB は有償 (評価版は無料で利用できる) であるため、必ずしもすべての人が自由に利用可能というわけには行かない[2]。また MATLAB プ

[2]MATLAB 互換ソフトとして GNU Octave がある。こちらは無償で利用可能であり、本稿に掲載した MATLAB プログラムは変更することなく Octave を用いて実行可能である。ただし MATLAB 互換とはいえ、その互換性は完全なものではない。

ログラムは MATLAB という環境の中でなければ利用できない。その点 Python3 は無償であり、様々なプラットフォーム (Windows、mac、Linux) で利用可能であり、OS(コマンドプロンプト・シェル) のレベルで利用することができる。作図機能は Python3 のシステムに標準に備わっているわけではないが、ここで例示したように matplotlib というモジュールで作図機能が提供されている。各プラットフォームへのインストールも容易であるので[3]、MATLAB より利用の自由度は高いと思われる。前にも書いたが Python3 は近年著しく注目を浴びており、また初学者にとっても学習が容易な言語である。時代は「一人に一台 (以上) の PC」であるので、Python に限らず何らかプログラミング言語をインストールさえしておけば、プログラミングの学習は、どこでもいつでもすることが可能である。

ところで「理論的研究は紙と鉛筆があればよい。解析解は万能である。」と考える諸氏もいると思われるが、それは必ずしも正しくはない。また「解析解とは異なり、数値計算は近似解過ぎない」という認識も捨て去る方が良い。解析解を求めることはもちろん重要であり、解の定性的な性質を理解するには必須のものである。しかし、一般に解析解を求めることができるのはごく限られた場合であり、何らかの仮定 (対称性の仮定等) をするのが普通である。たとえ正確無比な全く近似がなされていない解析解が得られたとしても、その解の振る舞いを定量的に確かめるためには、結局は解析解を数値化し、可視化しなければならないのである。実際適切な数値計算を行えば、具体的なしかも正しい結果を即座に目視で確認できる。早いうち (若いうち) から可視化して物理現象を具体的に目で見て感性を養っておくことは重要である。

まとめると「PC で物理学」＋「可視化機能を備えたプログラミング言語」は、「PC で物理学」の内容がそれほど陳腐化しているわけではないことを考え合わせると、天文学を志す大学初年級の学生にはプログラミングと天文学を学習するための手頃な教材であることが今回の事例研究で明らかとなった。プログラムを学習する上で「PC で物理学」は四半世紀たった今でも優れた教材であると考えられる。

最後に「はじめに」の部分に述べた、専門家を目指すわけではないが、天文に興味を持つ人が、この教材を利用できるであろうか。また大学の教育現場としては文化系学生に対してプログラミング教材として用いることができるであろうか。既存のプログラムを書き換え、その結果を視覚化することはさほど難しいことではない。しかし「PC で物理学」の天文学的な内容まで学習しようとすると、前提条件として、物理学の基礎、天文学の基礎、微分方程式の基礎等がどうしても必要である[4]。したがって適切な指導者の元であっても、これらの基礎知識を身に着けて後でなければ、「PC で物理学」利用するには難しいかも知れない。

今後の目標として、さらに様々な目的に適したテキストもあるので、それらについても教材として利用可能かどうか、今後機会があれば調査・研究を行ってみたいと思う。

[3] Anaconda[10] というディストリビューションにはよく利用されるモジュールが全て含まれている。
[4] 基礎とはいっても、理科系の大学生のレベルでの基礎である。

参考文献等

[1] 川端潔 (2008) パソコンで宇宙物理学 計算宇宙物理学入門 (国書刊行会)

Python3・matplotlib

[2] 柴田淳 (2016) みんなの Python(SB クリエイティブ)

[3] 大重美幸 (2017) 詳細! Python3 入門ノート (ソーテック社)

[4] 池内孝啓他 (2017) Python ユーザのための Jupyter[実践] 入門 (技術評論社)

MATLAB

[5] 上坂吉則 (2013) MATLAB 可視化プログラミング

[6] Amos Gilat (2015) MATLAB An Introduction with Applications(WILEY)

[7] Stormy Attaway (2017) MATLAB A Practical Introduction to Programming and Problem Solving(Butterworth-Heinemann)

天文学

[8] 岡村定矩監訳 (2003) オックスフォード 天文学辞典 (朝倉書店)

[9] 谷口義明 (2013) 新天文学事典 (講談社)

Anaconda

[10] `https://www.anaconda.com`

Programming Education with Visualization: The Case Study Using Computational Astrophysics

ITOH Makoto

Key Words: programming education, visualization, astrophysics

Abstract

We carried out the case study using the textbook ASTROPHYSICS WITH A PC: An Introduction to Computational Astrophysics(1994) . We rewrote BASIC programs in this textbook using recent programming languages. We also added the programming codes to visualize the computational results. We investigated the availability of this textbook in the class, and we also showed the importance of the visualization as well as the execution of the programs.