Pythonと実例で学ぶ
微分方程式

― はりの方程式から感染症の数理モデルまで ―

博士(理学)　神永 正博　著

コ ロ ナ 社

ま　え　が　き

本書は IT 活用型の微分方程式の教科書です。おもに，微分方程式を現実問題に応用したい学生・エンジニアの方々へ向けて書かれています。微分方程式の標準的な解法や，微分方程式が現実問題にどのように応用されるかを理解するとともに，Python を活用して現実問題を解けるようになることが目的です。

微分方程式は，自然科学，工学，医学などの共通言語です。多様な分野における現象を微分方程式の言葉によって記述し，現象の背後にあるメカニズムを解明し，未来を予測します。機械工学や土木工学を学ぶ方は，機械や建物の振動を解析するために微分方程式の応用としてモード解析を学ぶでしょうし，医学や疫学を学ぶ方は，感染症の拡大を予測するために微分方程式で記述された感染症の数理モデルを学ぶでしょう。本書では，こうした諸分野の入り口から少し内側にまで立ち入って，微分方程式がどのように活用されているのかまで説明します。さまざまな分野に興味を持ち，あちこち寄り道しながら微分方程式を学びたい方に向いています。

こうした多様な分野の入門的解説に加え，その多くに Python コードが付属しており，実際に動かして学ぶことができます（サンプルコードはコロナ社の書籍紹介ページ[†]からダウンロードできます）。Matplotlib ライブラリを使ったグラフの描画はもちろんですが，`scipy.optimize` を用いた実データへの当てはめ方を学んだり，気象予報士の試験問題や，電験三種（第三種電気主任技術者試験）の問題を解いたりもします。その他，電気工学で使われるインピーダンスや，ローパスフィルタ回路がどのように雑音を除去するかなども学べます。微分方程式を数式処理ライブラリ SymPy で解く方法を知り，`scipy.odeint` ライブラリによる数値計算ソルバを使えるようになります。

興味がある方のために，数値解析の理論も紹介しました。例えば，`scipy.odeint` ライブラリは内部で数値計算アルゴリズムを切り替えています。なぜ切り替えなければならないのか，なぜ標準解法として知られるルンゲ・クッタ法だけでは済まないのかなどについて，疑問に思う方がいらっしゃるかもしれません。これらについても説明し，数値計算ソルバを自分で実装できるところまで案内いたします。

事前に必要なのは，大学初年次に学ぶ微分積分学と線形代数学の知識，それに Python の実行環境がインストールされている PC だけです。理解を深めるための章末問題も 100 題用意しました（解答は，コロナ社の書籍紹介ページからダウンロードできます）。楽しんでいただけたら幸いです。

[†] https://www.coronasha.co.jp/np/isbn/9784339061239/

本書執筆にあたり，感染症の数理モデルについては東京大学大学院数理科学研究科教授の稲葉寿先生，電気回路については東北学院大学工学部教授の吉川英機先生，Python プログラムについては同じく東北学院大学工学部講師の森島佑先生からご助言いただきました。記して感謝申し上げます。

2021 年 8 月

神永 正博

目　　次

1．変数分離形の微分方程式

2．変数分離形以外の 1 階微分方程式

3．定数係数線形方程式

4. ラプラス変換，Python で厳密解・流れの可視化

5. Python で微分方程式を解く

6. Python で数値解析

1 変数分離形の微分方程式

本章では，最初に微分方程式とは何なのか，微分方程式を解くとはどういうことかを説明し，積分するだけで解ける微分方程式と変数分離形の微分方程式について多くの具体例とともに説明していきます。1 階の微分方程式はそれこそ無数にあるのですが，特に基本的で重要なものは変数分離形の方程式です。じつのところ，実用面で最も役立つタイプの微分方程式といっても過言ではありません。微分方程式の説明とともに，関数のグラフの描画や科学技術計算処理に利用されるライブラリ NumPy と Matplotlib についても説明します。

1.1 微分方程式とは何か

最初に準備運動を兼ねて，1 階に限らず，微分方程式とは何かを簡単に説明してから，ただ積分すれば解が求まる微分方程式について説明します。

1.1.1 微分方程式を解くということ

定数 C_1，C_2 を含む関数

$$x(t) = C_1 \cos t + C_2 \sin t \tag{1.1}$$

を考えます。つぎの x と，x の微分を使って，定数を消去してみましょう。

$$x(t) = C_1 \cos t + C_2 \sin t$$
$$x'(t) = -C_1 \sin t + C_2 \cos t$$
$$x''(t) = -C_1 \cos t - C_2 \sin t$$

となりますので

$$x''(t) = -x(t) \tag{1.2}$$

とすることで定数を消去できます。式 (1.2) のような，関数とその微分からなる方程式を**微分方程式**（differential equation）といい，これを満たす式 (1.1) のような関数を求めることを式 (1.2) を解くといいます。式 (1.2) は 2 階の微分を含んでおり，2 階よりも階数の高い微分は含まれていないので，2 階の微分方程式といいます。一般にその微分方程式が含む微分の最大の

階数が k のとき，k 階の微分方程式といいます。現象の解析に使われる微分方程式は，1 階か 2 階のものが多いのですが，3 階，4 階の微分方程式もあります。

後ほどさまざまな例を見ていきますが，自然現象は微分方程式の形で記述されることが多く，その振舞いを調べるために微分方程式を解くわけです。

定数を消去する方法は 1 通りとは限りません。$x''''(t) = x(t)$ は，4 階の微分方程式ですが，明らかに式 (1.1) を含んでいますし，定数を 1 つだけ消去した関係式として

$$x^2 + (x')^2 = C \tag{1.3}$$

も微分方程式（$C \geqq 0$ は定数）です。これは 1 階の微分方程式です。

1.1.2　常微分方程式と偏微分方程式

いま述べた微分方程式は，正確には，**常微分方程式**（ordinary differential equation）と呼ばれます。これは変数が 1 つだけ，例えば時刻 t のみの微分方程式という意味です。変数が 2 つ以上の微分方程式を**偏微分方程式**（partial differential equation）といいます。本書では偏微分方程式を扱いませんが，常微分方程式との違いと両者の関係について簡単に触れておきます。

例えば

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{1.4}$$

は，時刻 t，（直線上の）位置 x という 2 つの変数を持つ関数 $u = u(t, x)$ が満たす方程式で，偏微分方程式の 1 つです。この方程式は，**波動方程式**（wave equation）と呼ばれている重要な方程式で，速度 $c\ (\neq 0)$ の波の運動を記述しています。常微分方程式と偏微分方程式を区別する理由の 1 つは，常微分方程式では定まらない定数（式 (1.1) で出てきた C_1，C_2 のようなもの）の数が有限であるのに対し，偏微分方程式では，（標語的にいえば，ということですが）解が無数の定数を含むということです。有限と無限という点で扱いが根本的に違ってくるのです。常微分方程式よりも偏微分方程式のほうが数学的には難しい対象ですが，偏微分方程式を解く際には常微分方程式が基礎となります。歴史的には，フーリエ（Jean Baptiste Joseph Fourier）[†]が，熱方程式（拡散方程式ということもあります）の初期値・境界値問題

$$\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \\ &u(0, x) = f(x) \\ &u(t, a) = u(t, b) = 0 \end{aligned}$$

を解く際に，次節で扱う変数分離形の微分方程式と，3 章で扱う定数係数の線形微分方程式が

[†] ジャン・バティスト・ジョゼフ・フーリエは，18 世紀後半から 19 世紀前半にかけて活躍したフランスの数学者・物理学者です。熱伝導に関する研究から**熱方程式**（heat equation）を導出して，これを解くためにフーリエ級数の理論を展開しました。

現れました[†1]。これらの常微分方程式を解くことで，熱方程式を解くことができたのです。このように，解の構成を行う際に常微分方程式を解くことがありますが，特殊な形の解を求める際に常微分方程式を解く必要が出てくることもあります。例えば，光ファイバ内の光パルスの運動は，空間 1 次元の**非線形シュレディンガー方程式**（nonlinear Schrödinger equation）

$$i\frac{\partial u}{\partial t}+\frac{\partial^2 u}{\partial x^2}+|u|^{p-1}u=0 \tag{1.5}$$

で記述されます。i は虚数単位です。$p>1$ は定数ですが，応用上は，$p=3$ に取ることが多いようです。u の正確な意味づけは難しいですが，だいたい光波の包絡線関数と考えればよいでしょう。工学的には，大容量の光通信を実現する際に出会う方程式です。この方程式の**定在波解**（standing wave solution）は，$u(t,x)=e^{i\omega t}\varphi(x)$ という形の解ですが，これを式 (1.5) に代入すると

$$i(i\omega)e^{i\omega t}\varphi+e^{i\omega t}\varphi''+|\varphi|^{p-1}e^{i\omega t}\varphi=0$$

となります。両辺を $e^{i\omega t}$ で割ると，つぎの常微分方程式が得られます。

$$\varphi''-\omega\varphi+|\varphi|^{p-1}\varphi=0 \tag{1.6}$$

詳しい話は多くの準備が必要なので立ち入りませんが，式 (1.6) の解を調べることで，光ファイバ中の光波の動きがわかるのです。このように，偏微分方程式の特別な解を調べる際に常微分方程式が現れることがあります。

ここで，非線形という言葉が出てきました。波の重ね合せができる（解の 1 次結合もまた同じ方程式の解になっている）方程式を線形方程式，できないものを非線形方程式といいます（線形方程式の正確な定義については，2.2 節をご参照ください）。波動方程式と熱方程式は線形方程式で，非線形シュレディンガー方程式は名前のとおり非線形方程式です。

先ほど，常微分方程式は偏微分方程式より簡単だといいましたが，これは数学的な取扱いが入門段階で比較的簡単ということであって，最先端の研究において常微分方程式のほうが簡単というわけではありません。式 (1.6) でもわからないことがいろいろとありますし[†2]，昔から研究されている難問として，**三体問題**（the three-body problem）があります。三体問題とは，3 つの天体の運動を記述する常微分方程式の性質を調べる問題ですが，その解の複雑さは驚くべきものです。ポアンカレ（Henri Poincaré）は，「三体問題がいかに複雑なものか，またこの

---

†1 波動方程式との違いは，時間微分が 1 階になっただけですが，解の性質は随分違います。例えば，熱方程式では，時間がわずかでも経過すれば，初期値（ここでいえば $f(x)$）が滑らかでなくても解は非常に滑らかになる（平滑化効果がある）のに対し，波動方程式ではそうならず，解の特異性（滑らかでない部分）が伝播する（解の特異性の伝播）ことが知られています。

†2 昔，大学時代の友人と 2 人で，反発的な非線形項を持つ非線形シュレディンガー方程式に，吸引的なデルタポテンシャルが入った場合の定在波解の安定性を巻末の文献1) のように調べたことがあります。論文を書くにあたりいろいろと調べて仰天したのは，このようにごく基本的な問題があまりわかっていないことでした。非線形の微分方程式は非常に難しいのです。

問題を解くためには，われわれの既知のすべての知識とは異なった超越的な知識が，いかに必要であるかもわかってくる」[2)†]といっています。

1992 年，三体問題について，3 つの星が 8 の字型の軌道を描く，いわゆる 8 の字解が存在することが証明されました[3)]。これは数値解が発見され，後に数学的な存在証明がなされたものですが，このような解があるなんてほとんど信じられないような話です。宇宙は広いので，どこかに本当に 8 の字軌道の天体があるのかもしれません。三体問題にはいまでも多くの謎が残されています。

本書では偏微分方程式は扱わないので，以下，特に断らない限り，微分方程式と書けば，常微分方程式を意味するものとします。

1.1.3　NumPy と Matplotlib の基本的な使い方

本書では，特に NumPy と Matplotlib というライブラリを頻繁に使いますので，ここで，必要な知識をまとめておきます。ご存じの方は読み飛ばしていただいて結構です。

以降，Python 3.6（以降）と NumPy，Matplotlib，SciPy，SymPy というライブラリがインストールされていることを前提に説明しています（互換性のない Python 2.x については対応していませんのでご注意ください）。Python のインストールについては，いくつかの方法がありますが，ウェブ上に豊富な情報がありますので，そちらを見ていただいたほうがよいでしょう。本書では，Anaconda をインストールして標準で使える Spyder を使っています。Anaconda をインストールした場合は，すでに上記の 4 つのライブラリがインストールされているはずです。

SciPy は，NumPy ベースの科学技術計算ライブラリで，使いやすいインタフェースにラッピングされています。SymPy は数式処理ライブラリで，代数計算，微分・積分，ラプラス変換などを行うことができます。SymPy と SciPy の使い方については，適宜説明することにして，ここではより頻繁に利用する NumPy と Matplotlib について説明します。

〔**1**〕 **NumPy の n 次元配列（ndarray）**　　Python で配列を扱う場合，リストにする方法と `ndarray` にする方法の 2 つがあります。リスト化は手軽な方法ですが，処理が遅く，高速な処理が必要な科学技術計算には向いていません。Python で高速な計算を行う場合には，`ndarray` というデータ型にすることで，NumPy という高速なライブラリが使えるようになります。以下，単に配列といえば `ndarray` のことを指します。`ndarray` は，n-dimensional array（n 次元配列）という意味です。1 次元の配列はリストと似ていますが，リストでは異なる型が混ざっていてもよいのに対し，`ndarray` ではそれが許されず，すべての要素が同じ型である必要があります。IPython で少し様子を見てみましょう。IPython は，対話型の Python インタフェースで，プログラムを組むほどでないちょっとしたことを試すのに便利です。例えば，$\cos\dfrac{\pi}{2}$，$\sin\dfrac{\pi}{3}$，$\sin\dfrac{\pi}{4}$ を一度に計算させることを考えます。NumPy を使う場合は，このようにします。

† 肩つき数字は巻末の引用・参考文献を示します。

```
In [1]: import numpy as np
In [2]: PI = np.pi
In [3]: x = np.array([PI/2,PI/3,PI/4])
In [4]: np.cos(x)
Out[4]: array([6.12323400e-17, 5.00000000e-01, 7.07106781e-01])
```

1 行目で Numpy を `np` という名前をつけてインポートしています。これは広く使われている略記法ですので，本書でもこれにならいます。2 行目で NumPy 用の π に `PI` という名前をつけました。3 行目では，$(\pi/2, \pi/3, \pi/4)$ という 1 次元配列（`ndarray` 型のデータ）をつくっています。4 行目では，この `x=(x[0], x[1], x[2])` という配列に対して，`[cos(x[0]), cos(x[1]), cos(x[2])]` の値を計算しています。配列の要素ごとに正弦の値を求めて並べたものが出力されていることがわかるでしょう。$\cos\dfrac{\pi}{2}$ は 0 にならなければなりませんが，ここでは非常に小さい値ではあるものの 0 にはなっていません。これは数値計算の誤差によります。続けて，`x` の要素をまとめて 2 倍してみましょう。つぎのように書くだけです。ベクトルのスカラー倍と同じです。

```
In [5]: 2*x
Out[5]: array([3.14159265, 2.0943951 , 1.57079633])
```

同じ型の配列なら，ベクトルのように足すこともできます。

```
In [6]: y = np.array([1, 2, 3])
In [7]: z = np.array([3, 8, 9])
In [8]: y+z
Out[8]: array([ 4, 10, 12])
```

掛け算やべき乗の計算もできます。例えば，t を `ndarray` 型にして t^2+2t+3 とするとつぎのようになります。Python では，a^b は，`a**b` で表します。

```
In [9]: t = np.array([1, 2, 3, 4])
In [10]: t**2+2*t+3
Out[10]: array([ 6, 11, 18, 27])
```

このように成分ごとに計算してくれるわけです。

`ndarray` では複素数を扱うこともできます。微分方程式を解くために必要となる特性方程式という代数方程式がありますが，その解が複素数のとき，オイラーの公式（$e^{i\theta}=\cos\theta+i\sin\theta$）を経由して複素数成分を持つ配列を扱う場合がありますので，ここで簡単に説明しておきましょう。Python では虚数単位を `j` で表現します。これは電気工学などで一般に使われる記法です。電気工学では，電流を i で表現するため，i を虚数単位としては使わずに j を使うことになったのです。本書では，数学の慣習に従って虚数単位を i で表しますが，Python での表現は `j` になるので混同しないようご注意ください。

例えば，$z=(-1+5i, 1.4+7.6i, -1.7+0.5i)$ というベクトル（配列）はつぎのように表現します†。

† `dtype` を参照して `z.dtype` とするとその型（プラットフォームに依存）が表示されます。筆者の環境では，`dtype('complex128')` と出ます。

索　　引

—— 著 者 略 歴 ——

1991年　東京理科大学理学部数学科卒業
1993年　京都大学大学院理学研究科修士課程修了（数学専攻）
1994年　京都大学大学院理学研究科博士課程中退（数学専攻）
1994年　東京電機大学助手
1998年　株式会社日立製作所勤務
2003年　博士（理学）（大阪大学）
2004年　東北学院大学講師
2005年　東北学院大学助教授
2007年　東北学院大学准教授
2011年　東北学院大学教授
　　現在に至る

Python と実例で学ぶ微分方程式

―― はりの方程式から感染症の数理モデルまで ――

Ordinary Differential Equation with Python
― Learning via Many Real-World Examples, including Beam Equations, Mathematical Models of Infectious Diseases ―

2021 年 10 月 22 日　初版第 1 刷発行　★

検印省略

著　者　神永正博（かみなが まさひろ）
発行者　株式会社 コロナ社
　　代表者 牛来真也
印刷所　三美印刷株式会社
製本所　有限会社 愛千製本所

112–0011　東京都文京区千石 4–46–10
発行所　株式会社 コロナ社
CORONA PUBLISHING CO., LTD.
Tokyo Japan
振替 00140–8–14844・電話(03)3941–3131(代)
ホームページ https://www.coronasha.co.jp

ISBN 978–4–339–06123–9　C3041　Printed in Japan　（齋藤）