技術の原点

Pythonで挑む分岐解析

To Tackle Bifurcation Problems with Python

上田哲史 Tetsushi UETA
天羽晟矢 Seiya AMOH

アブストラクト　機械学習やデータサイエンスを実践する基盤プログラミング言語としてPythonが注目されている．本解説では非線形問題，特に系にみられる周期解の分岐問題について，Pythonを用いたアプローチの詳細を述べる．分岐計算アルゴリズムを可読性高く実装でき，計算機やOSに依存せず，対話処理による試行錯誤が可能となる．幾つかの特徴的なコードを示しながら，分岐問題に対するPythonの優位点を述べる．また，Neimark-Sacker分岐におけるbialternate積を用いた解法のコンパクトな実装，及びSympyを用いたヘシアン生成自動化過程についても述べる．

キーワード　Python，分岐解析，テンソル，Bialternate積

Abstract　Python is gaining attention as a fundamental programming language for machine learning and data science. In this paper, we describe a detailed Python approach to nonlinear problems, especially the bifurcation problems of periodic solutions. It is a highly readable implementation of the bifurcation algorithm, independent of the computer and the operating system, and it allows interactive trial-and-error processing. We describe the advantages of Python for bifurcation problems with some illustrated codes. We also show a compact implementation of computation for Neimark–Sacker bifurcation using the bialternate product and an automated process for generating the Hessian using Sympy.

Key words　Python, Bifurcation analysis, Tensor, Bialternate product

1. はじめに

非線形力学系が微分方程式や差分方程式で与えられたとき，その動的性質を把握するには，パラメータの変化に応じた解の振る舞いを網羅的に分析する必要がある．非線形系一般に解析解を求めることは困難であるため，解軌道を数値計算により求め，過渡応答の後に周期解が確認されれば，それが解析の足がかりとなろう．パラメータの変更により，その周期解の安定性の変転，すなわち分岐現象が見いだされることがある．パラメータ空間において分岐現象を与える多様体を分岐集合といい，これらをなるべく簡単に求めることが本解説の主題である．分岐集合で構成されるパラメータ空間上の地図（分岐図）は，与えられた力学系の定性的性質を雄弁に語り，個々の問題の目的への有益な指針を示すであろう．本解説では分岐集合の計算におけるPythonならではのアプローチを示す．“簡単に”とはPythonにおいては，短く，凝縮されて，シンプルに，などの副詞となる．可読性も上がるとともに，バグなどの混入も避けることができよう．非自律系を例にとり，周期解の分岐集合計算におけるPython実装を述べる．

上田哲史　正員：フェロー　徳島大学情報センター
E-mail　ueta@tokushima-u.ac.jp
天羽晟矢　学生員　徳島大学大学院先端技術科学教育部
E-mail　mail@seiya-amoh.jp
Tetsushi UETA, Fellow (Center for Administration of Information Techonology, Tokushima University, 2-1 Minami-Josanjima, 770-8506 Japan) and Seiya AMOH, Student Member (Graduate School of Advanced Techonology and Sciences, Tokushima University, 2-1 Minami-Josanjima, 770-8506 Japan).

電子情報通信学会　基礎・境界ソサイエティ
Fundamentals Review Vol.16 No.3 pp.139–146 2023年1月

1.1　分岐解析ツールと計算機言語

力学系の平衡点や周期解の安定性が変転することを分岐現象といい，それらが生じるパラメータ値を精度良く求める要求は高い．方程式が与えられただけでは，周期解，ましてやその分岐現象に関してあらかじめ知見を得ることは一般に困難であり，何らかの試行錯誤が必要となる．微分方程式のどのようなパラメータで，何周期の解が存在するか，また，副次的に，どのような初期値で，どのような過渡応答でその解に辿り着くかを調べたい．

分岐図を得る簡易な方法として，brute-force法がある．これはパラメータ平面をメッシュで区切り，個々のグリッドに対応するパラメータ値でみられる周期解の周期を色付けするものである．この手法はしかし，全グリッドにおいて過渡応答が収束するまでの数値積分実行を要すること（準周期解やカオスなどの判定にはリアプノフ指数など，別の指標も必要となる），更には同一パラメータ値で複数の周期解が共存するとき，初期値に依存して決まる一つの周期解しか色付けられないことなどの不都合がある．分岐集合を直接曲線として求めることができれば，これらは解決される．

Doedel[(1)]のAUTOをはじめとする海外の関連アルゴリズム・ツールについてはKuznetsovの文献[(2)]に詳しい．本解説では，AUTO初版と同時期に川上[(3)]によって開発されたアルゴリ

ズムを中心にその Python 実装について述べる．関連アルゴリズム間の本質的な違いについては，関連文献[(2),(4),(5)]を参照頂きたい．

アルゴリズムをもとに独自に研究室で開発したツールはかけがえのないものであろう．新しい研究動向を反映させながら改良・拡張など，維持するためには，コンピュータや OS への依存，計算資源の確保・メンテナンスにかかるコスト発生はできるだけ避けたい．また，試行錯誤で有為なデータを得るためにも，ツール動作時には対話的な操作を行いたい．更に，ディスプレイ上の見映えとは別途，論文での使用にも耐え得る品質の視覚化手段も備えておきたい．

Python は，クロスプラットホームなオープンソース計算機言語であり，データ科学や AI の分野において注目されている．Python のライブラリ Numpy, Scipy は，FORTRAN 時代から継承されている科学技術計算ライブラリ BLAS (Basic Linear Algebra Subprograms) [(6)] LAPACK (Linear Algebra PACKage) [(7)]からコードが移植されており，信頼性が高い．また，グラフ描画ライブラリ Matplotlib では品質の高いグラフィックスが対話的処理で使える．

このような背景から，筆者の研究室では既存分岐解析ツールを Python で書き直し始めた．現在では Github において基本的なコードをリポジトリとして `https://github.com/tetsushiwahaha/` に公開している．このうち，本解説で中心的に扱う非自律系に対して，次の各リポジトリを用意している．

- 位相平面図 `nonautonomous_pp`
- 固定点・周期点計算 `nonautonomous_fix`
- 分岐集合計算 `nonautonomous_bf`

これらは，Scipy，Numpy，Matplotlib の 3 ライブラリだけ用いている．

分岐集合の計算原理の解説については，川上により 1980 年代に開発されたアルゴリズム[(3),(8)]，及びそれらの詳細な解説[(4),(9)]，ERATO プログラムによる MATLAB 実装[(10),(11)]など，関連記事の枚挙にいとまがないので，それらを参照頂きたい．本解説では Python を使用して分岐解析を進める際の利点，並びにツール設計上の新しい観点などに絞って述べる．

1.2 数学的準備

本解説では主に非自律系の初期値問題

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}, \lambda), \quad \text{with} \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{1}$$

を考える．自律系については 4.4 節を参照のこと．ここで，$\boldsymbol{x} \in \boldsymbol{R}^n$ を状態，$\lambda \in \boldsymbol{R}$ をパラメータとする．$\boldsymbol{f} : \boldsymbol{R}^n \to \boldsymbol{R}^n$ は C^∞ 級であり，また，周期 τ の周期関数である．すなわち，$\boldsymbol{f}(t+\tau, \boldsymbol{x}, \lambda) = \boldsymbol{f}(t, \boldsymbol{x}, \lambda)$ と仮定する．このとき，時刻 $t = 0$ で初期値 $\boldsymbol{x}_0$ から出発する解を $\boldsymbol{x}(t) = \boldsymbol{\varphi}(t, \boldsymbol{x}_0, \lambda)$ と書く．$\boldsymbol{x}(0) = \boldsymbol{\varphi}(0, \boldsymbol{x}_0, \lambda) = \boldsymbol{x}_0$ となる．式 (1) に周期解が存在するとき，それは $\boldsymbol{\varphi}(0, \boldsymbol{x}_0, \lambda) = \boldsymbol{\varphi}(\tau, \boldsymbol{x}_0, \lambda)$ と表すことができる．

周期解の分岐現象の解析には，Poincaré 写像による方法が有効である．系 (1) に周期解が存在するとき，

$$\begin{aligned} T: \quad & \boldsymbol{R}^n \to \boldsymbol{R}^n \\ & \boldsymbol{x}_0 \mapsto T(\boldsymbol{x}_0) = \boldsymbol{\varphi}(\tau, \boldsymbol{x}_0, \lambda) \end{aligned} \tag{2}$$

と記述される作用素 T を Poincaré 写像といい，周期解の軌道を周期 τ おきに標本化している．式 (2) は周期軌道上の点 $\boldsymbol{x_0}$ についての局所的な定義であるが，状態空間全域に拡張すると，初期値 $\boldsymbol{x}_0$ から始まる離散軌道 $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \ldots\}$ を生成する離散力学系

$$\boldsymbol{x}_{k+1} = T(\boldsymbol{x}_k) \tag{3}$$

を考えることができる．式 (1) の周期解は T によって $\boldsymbol{x}_0 = T(\boldsymbol{x}_0)$ と表される．このときの $\boldsymbol{x}_0$ を固定点と呼ぶ．$\ell \geqq 2$ に対して $\boldsymbol{x}_0 = T^\ell(\boldsymbol{x}_0)$ であるとき，$\boldsymbol{x}_1 = T(\boldsymbol{x}_0)$, $\boldsymbol{x}_2 = T(\boldsymbol{x}_1)$, ... で求まる ℓ 個の各点 $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{\ell-1}\}$ を ℓ-周期点といい，式 (1) における ℓ-周期解にあたる．

2. 位相平面図での対話処理

与えられた力学系が，パラメータの変化や初期値によってどのように振る舞うかを知りたい．初期値はマウスカーソルで随時与え，パラメータ値の変更もオンラインで行いたいだろう[(12)]．高次元の系でも，そのうちの二次元（位相平面）を切り取って観察し，図も高品質でファイルに残したい．作業工程の再現性も重要である．Python でこれらの要求に一気に応えることができる．

非自律系式 (1) の位相平面図を表示するためのスクリプトをリポジトリ `nonautonomous_pp` に置いている．リスト 1 は位相平面表示スクリプト `pp.py` である．実行は次のコマンドラインとなる：

```
% python3 pp.py in.json Return
```

リスト 1　位相平面図の表示スクリプト pp.py

```
1  import numpy as np
2  from scipy.integrate import solve_ivp
3  import matplotlib.pyplot as plt
4  import pptools
5
6  data = pptools.init()
7  duration = 2.0 * np.pi * data.dic['period']
8  tspan = np.arange(0, duration, data.dic['tick'])
9
10 while True:
11     s = solve_ivp(pptools.func, (0, duration),
12         data.dic['x0'], t_eval=tspan, args=(data,))
13     plt.plot(s.y[0,:], s.y[1,:], c='k', alpha=0.5)
14     plt.plot(s.y[0, -1], s.y[1, -1], 'o', c='r')
15     data.dic['x0'] = s.y[:, -1]
16     plt.pause(0.001)
```

リスト 4 行めに読み込む `pptools.py` は当該リポジトリに同梱されており，その中身はデータ授受のための `data` クラスの定義一つと，Matplotlib の初期化，入力デバイスによる対話的

操作のための処理[13]が記述されているだけの，160 行あまりのスクリプトである．

11〜12 行めの `solve_ivp()` は，Scipy の比較的新しい初期値問題汎用ソルバである．第一引数で微分方程式の右辺を与え，第二引数のタプルで与える区間（1 周期）について，第三引数の初期値から求積し，軌道の時系列をリストで返す．インスタンス `data` を引数で渡すことにより，`pptools.func()` と多くの情報を通信できる．13〜14 行では Matplotlib によって，軌道及び Poincaré 写像点を描画しているが，グラフ描画にかかるビューポートや座標系について気を払う必要がない．また，デバイスのイベントをチェックし（16 行），マウスのクリックがあればその座標値を新しい初期値として拾っている．他言語で実現した場合のコード量と比較すると，著しくコンパクトであることが分かるであろう．高次元系では別途，任意の二次元の描画，マウス入力による初期値の扱いを実装する必要があるが，数行の記述で済む．

リスト 2　設定 JSON ファイル `in.json`

```
1  {
2    "_comment": "Duffing equation",
3    "func": [ "x[1]",
4          "-data.dic['params'][0] * x[1] - x[0]**3
               + data.dic['params'][1]
               + data.dic['params'][2] * cos(t)"
5      ],
6    "x0":   [-1.0, 0.5],
7    "params":   [0.2, 0.08, 0.3],
8    "dparams":  [0.01, 0.01, 0.01],
9    "xrange": [ -2.0, 2.0], "yrange": [ -2.0, 2.0],
10   "tick": 0.01, "period":   1
11 }
```

リスト 2 は設定ファイル `in.json` の例である．3〜4 行めのキー `func` で微分方程式の右辺を記述する（リスト 1 における `pptools.func()` に当たる）．ここでは Duffing 方程式の例となっている．パラメータを配列 `data.dic['params']` によって設定する．4 行めはリスト 1 上では見やすいよう複数行で表示されているが．JSON は値としての文字列内で改行は許容されないことに注意されたい．キー `func` は `pptools.py` において `eval` 関数により式が評価され，リスト 1 における `pptools.func()` となるが，このことによる速度低下は認められなかった．設定ファイル内で微分方程式を記述できるため，力学系ごとに異なるスクリプトを用意する必要がなくなり，管理が容易となる．

図 1 はリスト 1 を動作させ，初期値入力の試行錯誤をキャプチャしたものである．キーボード操作によってパラメータの増減，redraw などが行える．マウス入力などの履歴はターミナルに記載されるため，再現性が確保される．また，結果はラスタ画像ではなく，ベクタグラフィックスとして PDF に保存できる．詳細は同梱の `README.md` を参照されたい．

3.　固定点の計算

固定点の条件は境界値問題として定式化される：

$$T(\boldsymbol{x}_0) - \boldsymbol{x}_0 = \boldsymbol{0} \tag{4}$$

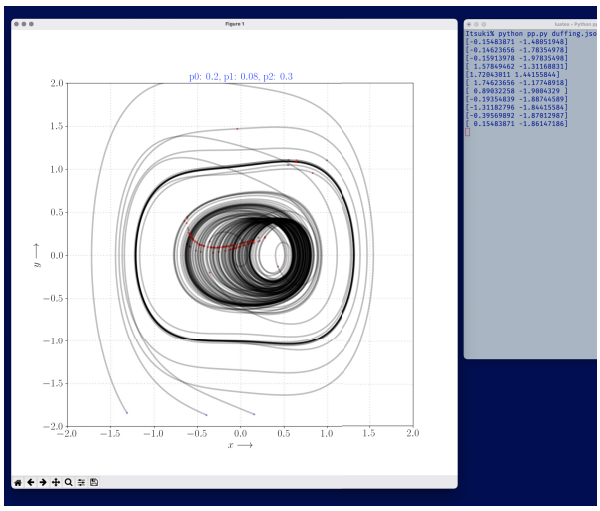

図 1　リスト 1 による位相平面図表示例

周期点の場合も T^ℓ を改めて T と置けば同一視できる．この固定点 $\boldsymbol{x}_0$ を高精度で計算しよう．

ニュートン法は二乗収束であるため，問題が微分可能であればまず選択される解法である．実装する際に，$\boldsymbol{x}_0$ による式 (4) のヤコビ行列が必要となるが，それは実際，

$$\frac{\partial}{\partial \boldsymbol{x}_0}\Big(T(\boldsymbol{x}_0) - \boldsymbol{x}_0\Big) = \left.\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{x}_0}\right|_{t=\tau} - \boldsymbol{I}_n \tag{5}$$

となる．ここで，$\boldsymbol{I}_n$ は $n \times n$ の単位行列である．また，特性方程式は次式となる：

$$\chi(\mu) = \det\left(\left.\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{x}_0}\right|_{t=\tau} - \mu \boldsymbol{I}_n\right) = 0 \tag{6}$$

この式の n 個の根である特性乗数が，固定点の安定性を示す．$\partial\boldsymbol{\varphi}/\partial\boldsymbol{x}_0$ は変分と呼ばれ，式 (5), (6) の双方に現れているが，この変分をどう求めるかが問題となる．

変分は解の導関数であるので，式 (1) の解軌道が数値的に得られたとき，それらの数値微分を変分として代用することが考えられる．例えば，十分小さい $\Delta > 0$ を用意し，$\boldsymbol{x}_0 = (x_{01}, x_{02}, \ldots, x_{0n})^\top$ としたとき，次式によって x_{01} に関する変分ベクトルを近似できる：

$$\begin{aligned}\frac{\partial \boldsymbol{\varphi}}{\partial x_{01}}(\tau, x_{01}, x_{02}, \ldots, x_{0n}) \approx \\ \frac{1}{\Delta}\Big(\boldsymbol{\varphi}(\tau, x_{01}+\Delta, x_{02}, \ldots, x_{0n}) \\ -\boldsymbol{\varphi}(\tau, x_{01}, x_{02}, \ldots, x_{0n})\Big)\end{aligned}$$

しかし，数値微分は本質的に誤差を包含し得るため，特性乗数の値やニュートン法の収束性能に影響することが考えられ，なるべく使用を避けたい．

さて，解をもとの微分方程式 (1) に代入し，微分順序を変更することによって，変係数線形微分方程式である第一変分方程式が得られる[8],[14]：

$$\frac{d}{dt}\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{x}_0}, \quad \text{with} \quad \left.\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{x}_0}\right|_{t=0} = \boldsymbol{I}_n \tag{7}$$

ここで，$\partial\boldsymbol{f}/\partial\boldsymbol{x}$ は式 (1) のヤコビ行列（式 (1) の右辺を状態変数 $\boldsymbol{x}$ で記号的に偏微分したもの）である．式 (7) の要素を明示

すると

$$
\frac{d}{dt}\left(\begin{array}{c|c|c|c}
\frac{\partial\varphi_1}{\partial x_{01}} & \frac{\partial\varphi_1}{\partial x_{02}} & \cdots & \frac{\partial\varphi_1}{\partial x_{0n}} \\
\frac{\partial\varphi_2}{\partial x_{01}} & \frac{\partial\varphi_2}{\partial x_{02}} & \cdots & \frac{\partial\varphi_2}{\partial x_{0n}} \\
\vdots & \vdots & & \vdots \\
\frac{\partial\varphi_n}{\partial x_{01}} & \frac{\partial\varphi_n}{\partial x_{02}} & \cdots & \frac{\partial\varphi_n}{\partial x_{0n}}
\end{array}\right) =
$$

$$
\left(\begin{array}{cccc}
\frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\
\vdots & \vdots & & \vdots \\
\frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n}
\end{array}\right)
\left(\begin{array}{c|c|c|c}
\frac{\partial\varphi_1}{\partial x_{01}} & \frac{\partial\varphi_1}{\partial x_{02}} & \cdots & \frac{\partial\varphi_1}{\partial x_{0n}} \\
\frac{\partial\varphi_2}{\partial x_{01}} & \frac{\partial\varphi_2}{\partial x_{02}} & \cdots & \frac{\partial\varphi_2}{\partial x_{0n}} \\
\vdots & \vdots & & \vdots \\
\frac{\partial\varphi_n}{\partial x_{01}} & \frac{\partial\varphi_n}{\partial x_{02}} & \cdots & \frac{\partial\varphi_n}{\partial x_{0n}}
\end{array}\right) \tag{8}
$$

となる．そこで，微分方程式初期値問題 (7) を，$t=0$ から $t=\tau$ まで，`solve_ivp()` などを用いて求積することによって，基本行列解としての変分 $\partial\boldsymbol{\varphi}/\partial\boldsymbol{x}_0|_{t=\tau}$ が得られ，式 (5), (6) に利用できる．微分方程式 (1) の右辺に対して初期値による微分・連鎖律を適用した結果が，行列の積の形式で整理されている．

式 (7) の Python での実装を検討する．`ndarray` の二次元配列を考え，ヤコビ行列を `dfdx`，変分を `dphidx` として格納すると，式 (8) 右辺の計算は，

```
dfdx @ dphidx
```

と記述できる．ここで '`@`' は Python のバージョン 3.5 から登場した演算子であり，matmul 関数のマクロである．しかし，`solve_ivp()` は入出力はベクトルでなければならない．そこで式 (8) 右辺の演算結果の行列を，n 本の縦ベクトルに分解し（左辺の縦線を参照），1 本のベクトルに連結する．それを，式 (1) の右辺ベクトル `func` に追加する．この手続は，

```
func.extend((dfdx @ dphidx).T.flatten())
```

とできる．`.T` 属性は転置，`.flatten()` はベクトルへの変換メソッド呼び出しである．これらの操作や行列積は，一般的な計算機言語では二重反復を構成する必要があるが，Python では 1 行で済む．

以上でニュートン法により固定点を計算する準備ができた．リポジトリ `nonautonomous_fix` には Duffing 方程式についての固定点計算スクリプト `fix.py` を配置している．

4. 分岐集合の計算

特性乗数の特定の値により発現する分岐現象を局所分岐といい，以下の種類がある．

- $\mu=1$：接線分岐．固定点が発生消滅する
- $\mu=-1$：周期倍分岐：不安定化した固定点の周りに安定な 2 周期点（2 周期解）が発生
- $\mu=e^{j\theta}$, $0<\theta<\pi$：Neimark-Sacker 分岐．不安定化した固定点周りに安定な不変閉曲線（トーラス）が発生

ここで，$j=\sqrt{-1}$ である．これらの分岐による固定点や軌道の様子を図 2 に示す．

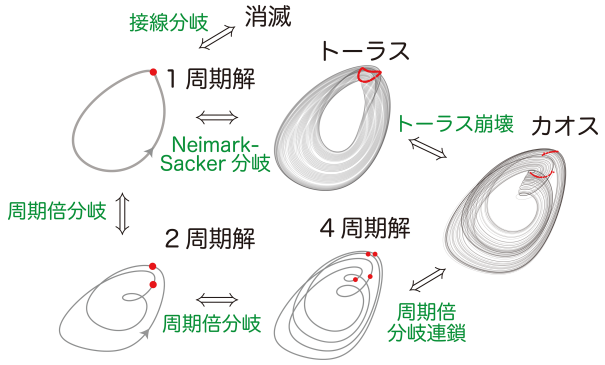

図 2　局所分岐と軌道の様子

分岐集合の計算には，式 (4), (6) を連立させる：

$$
\begin{cases} T(\boldsymbol{x}_0)-\boldsymbol{x}_0=\boldsymbol{0} \\ \chi(\mu)=0 \end{cases} \tag{9}
$$

ここで，μ は分岐現象に応じた特性乗数を指定する．1.1 節で紹介した既存のツールも，基本的には同じ境界値問題として定式化されている．

位相平面図でパラメータを変化させたとき，解軌道の形状が大きく変化するようであれば，そのパラメータ値付近で前節の固定点計算ツールを用いて固定点と特性乗数を算出，観察する．特性乗数が上記の特定の値近くにあれば，その $(\boldsymbol{x}_0,\lambda)$ を初期値として，式 (9) をニュートン法で解くと分岐集合を得る．

ニュートン法のヤコビ行列は次式の構成となる．

$$
\left(\begin{array}{c|c}
\frac{\partial\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0}-\boldsymbol{I}_n & \frac{\partial\boldsymbol{\varphi}}{\partial\lambda} \\ \hline
\frac{\partial\chi}{\partial\boldsymbol{x}_0} & \frac{\partial\chi}{\partial\lambda}
\end{array}\right) \tag{10}
$$

固定点の計算に比較し，新たに解のパラメータによる変分，解の初期値・パラメータによる二階微分（第二変分）が必要となってくるが，数値偏微分は回避したい．

4.1 パラメータに関する変分

固定点のパラメータに関する微分は，以下の n 次元変係数線形非同次方程式の数値積分で得られる．

$$
\frac{d}{dt}\frac{\partial\boldsymbol{\varphi}}{\partial\lambda}=\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}\frac{\partial\boldsymbol{\varphi}}{\partial\lambda}+\frac{\partial\boldsymbol{f}}{\partial\lambda},\quad \text{with}\quad \left.\frac{\partial\boldsymbol{\varphi}}{\partial\lambda}\right|_{t=0}=\boldsymbol{0} \tag{11}
$$

Python の実装に移ろう．パラメータによる変分を `dphidl` とする．式 (1) をパラメータにより記号的に偏微分したベクトルを `dfdl` に格納したとき，式 (11) を，`solve_ivp()` に渡すベクトル `func` に追加するには，

```
func.extend(dfdx @ dphidl + dfdl)
```

とする．行列とベクトルの積であるが，演算子は '`@`' が使える．

4.2 初期値に関する第二変分

式 (7) を $\boldsymbol{x}_0$ で偏微分し，連鎖律を整理すると，n^3 本の微分方程式が出来上がる．これを第二変分方程式[14]と呼ぶ．$\partial^2\boldsymbol{\varphi}/\partial\boldsymbol{x}_0^2$ を n 次元 3 階テンソルとすると，第二変分方程式は，

$$\frac{d}{dt}\frac{\partial^2\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0^2} = \frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}\frac{\partial^2\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0^2} + \frac{\partial^2\boldsymbol{f}}{\partial\boldsymbol{x}^2}\left(\frac{\partial\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0}\right)^2, \quad \text{with} \quad \left.\frac{\partial^2\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0^2}\right|_{t=0} = \boldsymbol{O} \tag{12}$$

と書ける．ここで，$\boldsymbol{O}$ は全ての要素が 0 であるテンソルである．また式中の $\partial^2\boldsymbol{f}/\partial\boldsymbol{x}^2$ は，ヤコビ行列 $\partial\boldsymbol{f}/\partial\boldsymbol{x}$ を状態変数 $\boldsymbol{x}$ で記号的に偏微分して得られるヘシアン（テンソル）である．$\partial\boldsymbol{\varphi}/\partial\boldsymbol{x}_0$ は，式 (7) の求積で得られている．右辺第二項は便宜上この記述を取ったが，テンソルと行列の積を定義する必要があり，ベクトルと行列で記述すると複雑となる．例として，$n=2$ の場合の第二変分方程式の一部を抽出すると，

$$\frac{d}{dt}\begin{pmatrix}\dfrac{\partial^2\varphi_1}{\partial x_0^2}\\ \dfrac{\partial^2\varphi_2}{\partial x_0^2}\end{pmatrix} = \frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}\begin{pmatrix}\dfrac{\partial^2\varphi_1}{\partial x_0^2}\\ \dfrac{\partial^2\varphi_2}{\partial x_0^2}\end{pmatrix} + \underbrace{\frac{\partial}{\partial x_0}\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}}\begin{pmatrix}\dfrac{\partial\varphi_1}{\partial x_0}\\ \dfrac{\partial\varphi_2}{\partial x_0}\end{pmatrix} \tag{13}$$

となる．ここで波下線部は，

$$\begin{pmatrix}\dfrac{\partial^2 f_1}{\partial x^2}\dfrac{\partial\varphi_1}{\partial x_0} + \dfrac{\partial^2 f_1}{\partial y\partial x}\dfrac{\partial\varphi_2}{\partial x_0} & \dfrac{\partial^2 f_1}{\partial x\partial y}\dfrac{\partial\varphi_1}{\partial x_0} + \dfrac{\partial^2 f_1}{\partial y^2}\dfrac{\partial\varphi_2}{\partial x_0}\\ \dfrac{\partial^2 f_2}{\partial x^2}\dfrac{\partial\varphi_1}{\partial x_0} + \dfrac{\partial^2 f_2}{\partial y\partial x}\dfrac{\partial\varphi_2}{\partial x_0} & \dfrac{\partial^2 f_2}{\partial x\partial y}\dfrac{\partial\varphi_1}{\partial x_0} + \dfrac{\partial^2 f_2}{\partial y^2}\dfrac{\partial\varphi_2}{\partial x_0}\end{pmatrix} \tag{14}$$

となり，見通しが悪い．実際，式 (12) において，$\boldsymbol{f} = (f_1, f_2, \ldots, f_n)^\top$ 及び $\boldsymbol{\varphi} = (\varphi_1, \varphi_2, \ldots, \varphi_n)^\top$ としたとき，n^3 本の第二変分方程式を個々に展開すれば次式となる．

$$\begin{aligned}\frac{d}{dt}\frac{\partial^2\varphi_i}{\partial x_{0k}\partial x_{0\ell}} &= \sum_{p=1}^{n}\frac{\partial f_i}{\partial x_p}\frac{\partial^2\varphi_p}{\partial x_{0k}\partial x_{0\ell}}\\ &\quad + \sum_{p=1}^{n}\sum_{q=1}^{n}\frac{\partial^2 f_i}{\partial x_p\partial x_q}\frac{\partial\varphi_p}{\partial x_{0k}}\frac{\partial\varphi_q}{\partial x_{0\ell}},\\ \text{with}\quad & \left.\frac{\partial^2\varphi_i}{\partial x_{0k}\partial x_{0\ell}}\right|_{t=0} = 0\\ \text{for}\quad & i,k,\ell = 1,2,\ldots,n\end{aligned} \tag{15}$$

これをアルゴリズムとして表現しようとすると，多重・複数の反復を記述したうえ，添字計算に注意が必要となる．また，この展開式（若しくは式 (7) に対する微分・連鎖律の逐次計算）の正当性検証を行うのは容易ではない．

さて，式 (12) 右辺を Python で記述しよう．第二変分を `d2phidx2`，ヘシアンを `d2fdx2` とすると，式 (12) の右辺は，

```
dfdx @ d2phidx2 + (d2fdx2 @ dphidx).T @ dphidx
```

となる．反復やスライス，添字を全く使うことなく，ほとんど式 (12) 右辺の見た目どおりの形式となる．この式の値を `P` とすると，ベクトルへの展開・連結には，

```
P.transpose(0,2,1).flatten()
```

とすればよい．`transpose()` によりテンソルの転置軸の変更を行うことがポイントである．

ところで，2 変数偏微分の順番交換に関するヤングの定理より，$\partial^2\boldsymbol{\varphi}_i/\partial x_k\partial x_\ell = \partial^2\boldsymbol{\varphi}_i/\partial x_\ell\partial x_k$ であるため，n^3 本の連立微分方程式のうち，おおむね半分は計算不要である．

リスト 3　第二変分方程式 (12) 右辺の計算

```
1  ui, uj = np.triu_indices(n)
2  v = x.reshape(int(n*(n+1)/2), n)
3  X = np.zeros(n**3).reshape(n, n, n)
4  X[ui, uj] = v
5  X[uj, ui] = v
6  d2phidx2 = X.transpose(0, 2, 1)
7  P = (dfdx @ d2phidx2 + (d2fdx2 @ dphidx).T @ dphidx)
       .transpose(0, 2, 1)
8  func.extend(P[ui, uj].flatten())
```

リスト 3 は，必要最小限な第二変分のみ `solve_ivp()` に渡す流れを示している．1 行めで上三角行列における有効な要素の行，列それぞれの添字リストを得る．必要最小限な数の第二変分が既にベクトル `x` に格納されているとき，それを対称な第二変分テンソル `d2phidx2` に戻し（2〜7 行），上三角行列の添字リストで対称なテンソルを生成できる．スライスさえ用いる必要がない点は興味深い．そのあと，第二変分を計算し（7 行め），最小限の数の第二変分のみ `solve_ivp()` に渡している（8 行め）．

4.3 パラメータに関する第二変分

最後はパラメータに関する第二変分である．式 (7) を λ で偏微分し連鎖律を適用して，次式を得る：

$$\begin{aligned}\frac{d}{dt}\frac{\partial^2\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0\partial\lambda} &= \frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}\frac{\partial^2\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0\partial\lambda} + \frac{\partial^2\boldsymbol{f}}{\partial\boldsymbol{x}^2}\frac{\partial\boldsymbol{\varphi}}{\partial\boldsymbol{x}}\frac{\partial\boldsymbol{\varphi}}{\partial\lambda} + \frac{\partial^2\boldsymbol{f}}{\partial\boldsymbol{x}\partial\lambda}\frac{\partial\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0}\\ \text{with}\quad & \left.\frac{\partial^2\boldsymbol{\varphi}}{\partial\boldsymbol{x}_0\partial\lambda}\right|_{t=0} = \boldsymbol{O}\end{aligned} \tag{16}$$

ここでの $\boldsymbol{O}$ は $n\times n$ の零行列である．式 (16) 右辺は Python では以下で記述できる：

```
func.extend(dfdx @ d2phidxdl
  + ((d2fdx2 @ dphidx).T @ dphidl).T
  + (d2fdxdl @ dphidx)).T.flatten()
```

ここで，`dphidl` は式 (11) の解を用いる．

以上で式 (9) をニュートン法で解くために必要な要素が全て揃った．詳細はリポジトリ `nonautonomous_bf` のコードを参照してほしい．分岐直前の初期値を与えていれば数回の反復で精度のよい解（固定点と分岐パラメータ値）が得られる．brute-force 法では表現不能な，不安定な周期解の分岐集合も計算できる．

4.4 自律系への対応

式 (1) において，右辺 $\boldsymbol{f}$ が時間 t に対し不変である場合を自律系という．自律系においても周期解が存在し得て，その分岐解析は，軌道に横断的な Poincaré 断面を選んでその標本点を評価することにより，式 (2) と同様な $n-1$ 次元離散写像の分岐問題に帰着させることができる[15),16)]．式 (7), (12) などの変分方程式の計算は，自律系でも同様に適用できる．

`solve_ivp()` には `events` パラメータがあり，これを Poincaré 切断面のトリガに利用できる．軌道が切断面に接触した位置・時刻が高精度で求められるうえ，積分が中断されるため，固定点や分岐集合の計算に都合がよい．

5. Neimark–Sacker 分岐と bialternate 積

接線分岐及び周期倍分岐は，前節までの手法で分岐集合が計算できる．しかし，Neimark-Sacker (NS) 分岐では特性乗数 $\mu = e^{j\theta}$ が複素単位円周上に配置されるため，未知数の偏角 θ が介在し，特性方程式は実質 2 本の独立した式となる．よってこのままではニュートン法における変数が足りない．

次元の低い問題であれば，特性方程式に $\mu = e^{j\theta}$ を代入・展開し，実部・虚部から θ を消去した式を新たな条件式に採用できる．ただし，必ず不等式による付帯条件が発生するため，計算が収束した後に別途検証が必要となる．文献(17) では，複素数型をもたない計算機言語を用い，式 (6) の実部・虚部分離アルゴリズム，θ を独立変数とみなした NS 分岐計算手法が提案されている．

Python は複素数型をもっているが，複素数を代入した特性方程式を用いたニュートン法を実装すると，その反復過程において実変数に対して複素数の更新量を許容させる必要が生じ，また，実数解への収束は保証されず，不都合である．そこで本節では，NS 分岐に対して特性方程式と等価な条件 bialternate 積[18)]を導入し，実数のみでニュートン法を実装する手法[2)]を Python で実装しよう．

$\boldsymbol{A}, \boldsymbol{B} \in \boldsymbol{R}^{n\times n}$ を任意の行列としたとき，bialternate 積 $\boldsymbol{A} \odot \boldsymbol{B}$ は次を満たす．

（1） $\boldsymbol{A} \odot \boldsymbol{A}$ は固有値 $\mu_i \mu_j$ をもつ

（2） $2\boldsymbol{A} \odot \boldsymbol{I}_n$ は固有値 $\mu_i + \mu_j$ をもつ

NS 分岐における複素特性乗数 $\mu, \overline{\mu} = e^{\pm j\theta}$ は，性質（1）で $\mu\overline{\mu} = 1$ を満たす．これを特性方程式に代わる条件として用いる．すなわち，NS 分岐集合は，次の連立方程式を $(\boldsymbol{x}_0, \lambda)$ について解くことにより求められる．

$$\begin{cases} T(\boldsymbol{x}_0) - \boldsymbol{x}_0 = \boldsymbol{0} \\ \det\left(\dfrac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{x}_0} \odot \dfrac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{x}_0} - \boldsymbol{I}_m\right) = 0 \end{cases} \tag{17}$$

bialternate 積を用いることで θ が消去されており，また，別途検証の必要もなくなる．

以下，bialternate 積とその Python コードについて説明する．$\boldsymbol{A} \odot \boldsymbol{B} \in \boldsymbol{R}^{m\times m}$ の各要素は，その要素の添字を多重指数 (multi-index) を用いて行方向を (p,q) $(p = 2, 3, \ldots, n;\ q = 1, 2, \ldots, p-1)$，列方向を (r,s) $(r = 2, 3, \ldots, n;\ s = 1, 2, \ldots, r-1)$ で表すと，

$$(\boldsymbol{A} \odot \boldsymbol{B})_{(p,q),(r,s)} = \frac{1}{2}\left\{\begin{vmatrix} a_{pr} & a_{ps} \\ b_{qr} & b_{qs} \end{vmatrix} + \begin{vmatrix} b_{pr} & b_{ps} \\ a_{qr} & a_{qs} \end{vmatrix}\right\} \tag{18}$$

で定まる．ここで，$m = n(n-1)/2$ であり，$a_{k\ell}, b_{k\ell}$ は行列 $\boldsymbol{A}$, $\boldsymbol{B}$ の (k,ℓ) 要素を表す．また，$|\cdot|$ は行列式を表す．例えば，$n = 3$ のときは $m = 3$ となり，bialternate 積 $\boldsymbol{A} \odot \boldsymbol{B}$ によって生成される行列は，

$$\begin{matrix} (\boldsymbol{A} \odot \boldsymbol{B})_{(2,1),(2,1)} & (\boldsymbol{A} \odot \boldsymbol{B})_{(2,1),(3,1)} & (\boldsymbol{A} \odot \boldsymbol{B})_{(2,1),(3,2)} \\ (\boldsymbol{A} \odot \boldsymbol{B})_{(3,1),(2,1)} & (\boldsymbol{A} \odot \boldsymbol{B})_{(3,1),(3,1)} & (\boldsymbol{A} \odot \boldsymbol{B})_{(3,1),(3,2)} \\ (\boldsymbol{A} \odot \boldsymbol{B})_{(3,2),(2,1)} & (\boldsymbol{A} \odot \boldsymbol{B})_{(3,2),(3,1)} & (\boldsymbol{A} \odot \boldsymbol{B})_{(3,2),(3,2)} \end{matrix}$$

である．NS 分岐条件に用いる $\boldsymbol{A} \odot \boldsymbol{A}$ の各要素は明らかに，

$$(\boldsymbol{A} \odot \boldsymbol{A})_{(p,q),(r,s)} = \begin{vmatrix} a_{pr} & a_{ps} \\ a_{qr} & a_{qs} \end{vmatrix} \tag{19}$$

であり，これを計算する関数 `bialt_square` をリスト 4 に示す．

リスト 4　bialternate 積の実装例

```
def bialt_square(A):
  n = A.shape[0]
  bialt_dim = sum(range(n))
  result = np.zeros((bialt_dim, bialt_dim))
  temp = np.zeros((2, 2))
  result_idx = ((i, j) for i in range(bialt_dim)
                        for j in range(bialt_dim))
  mul_idx = [(i, j) for i in range(1, n)
                    for j in range(i)]
  for row, col in it.product(mul_idx, mul_idx):
    for i, j in it.product([0, 1], [0, 1]):
      temp[i, j] = A[row[i], col[j]]
    result[next(result_idx)] = np.linalg.det(temp)
  return result
```

リスト 4 では，はじめに n と m に相当する変数 `n`, `bialt_dim`, bialternate 積の計算結果を保存する変数 `result`，またその各要素を一時的に保存する変数 `temp` を用意している．Numpy 配列の要素へのアクセスはタプルが利用できるため，`result` の要素指定のためにイテレータ `result_idx` も用意する．続いて，bialternate 積では式 (18) で多重指数が添字として用いられている．この添字 $(p,q), (r,s)$ は，多重指数の第一指数と第二指数の集合の直積で表現できるため，この各集合を `mul_idx` とし，`it.product()` を用いて添字のイテレータを生成する．bialternate 積の Python 上での実装においては，多重指数の管理が 8〜10 行めだけで記述できることが重要である．なお，あらかじめ `import itertools as it` として反復操作の補助を行うライブラリをインポートする必要がある．

6. Sympy を用いた導関数の自動導出

力学系の分岐計算はこれまでに示したとおり，Newton 法の

計算のための変分方程式の基本行列解を求める際に，式 (1) の右辺 $\boldsymbol{f}$ のヤコビ行列やヘシアンを準備する必要がある．これらは単純な系であれば手計算したものを直接コーディングすればよいが，高次元系やパラメータを多数含む系の場合は，手計算するコストが高い．Python には四則演算や微分など代数計算の操作を記号的に取り扱うことのできる Sympy パッケージが存在する．分岐計算対象の系の右辺が `func()` 関数として記述されているとき，リスト 5 に示す操作で $\boldsymbol{f}$ の状態変数とパラメータによる微分を記号的に導出できる[19]．`func()` は第 2 節の方法のように，微分方程式を JSON ファイルに記述できるように設計してもよい．ここで，`n` 及び `pnum` は，それぞれ状態変数の次元 n と，系がもつパラメータの数である．

リスト 5　Sympy を用いた導関数の導出

```
import sympy as sp
def func(x, p, t):
    return sp.Matrix([f_1, f_2, ..., f_n])
sym_x = sp.MatrixSymbol("x", n, 1)
sym_p = sp.MatrixSymbol("p", pnum, 1)
sym_t = sp.Symbol("t")
f = func(sym_x, sym_p, sym_t)
dfdx = sp.derive_by_array(
       [f[i] for i in range(n)],
       [sym_x[i] for i in range(n)]).transpose()
dfdl = [sp.diff(f, sym_p[i])
        for i in range(pnum)]
d2fdx2 = [sp.diff(dfdx, sym_x[i])
          for i in range(n)]
d2fdxdl = [sp.diff(dfdx, sym_p[i])
           for i in range(pnum)]
```

$\boldsymbol{f}$ やその微分は分岐計算の実行ごとに変化するものではないため，一度の実行でコードとして書き出しておくことが望ましい．Sympy を用いて導出した式オブジェクトは `print()` などで Python 形式のコードとして出力することができる．

7. 計　算　例

本節では，ここまでに記述した分岐解析手法及び Python スクリプトを用いて実際に非自律系の分岐曲線計算を行う．次の三階 Duffing 形方程式[20]を考える．

$$\begin{aligned}\dot{x} &= y\\ \dot{y} &= -0.05y - (3z^2 + x^2)x/8 + B\cos t\\ \dot{z} &= -0.05(3x^2 + z^2)z/8 + B_0\end{aligned} \tag{20}$$

分岐計算にあたり，まず JSON 形式の入力ファイルに微分方程式 (20) を記述の上，6. 節の手法を適用する．得られたヘシアンをスクリプト内にハードコードさせた．本系では 4. 節に示した三つの局所分岐のうち，接線分岐と NS 分岐が発生する．NS 分岐曲線の追跡には特性方程式の実虚部分離操作を行わずに，5. 節の手法を用いて $(\boldsymbol{x}_0, \lambda)$ について解いた．

B–B_0 平面の分岐図を図 3 に示す．図中 NS は NS 分岐，G^i は i 周期解の接線分岐を表す．青色領域は二つの 1 周期解が共存する領域である．また，赤色領域は 1 周期解と 3 周期解が共存する．灰色領域は 1 周期解が NS 分岐によって準周期化する

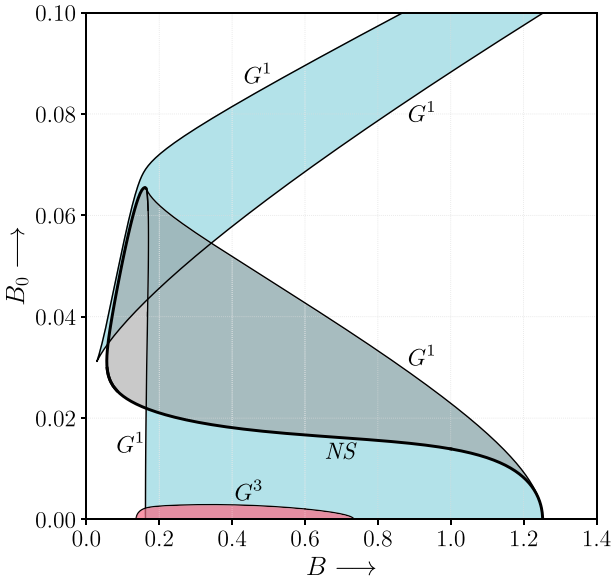

図 3　式 (20) の分岐図

パラメータであり，この領域ではトーラス崩壊によるカオス解が存在する．青色と灰色が重なる領域では，一つの 1 周期解と準周期解が共存する．青色と赤色の重なる領域では，三つの安定解が共存する．着色のない領域はただ一つの 1 周期解が存在する．

図 3 を得るために使用したプログラムは，別途著者 github リポジトリの `nonautonomous_bf` に配置している．

8. おわりに

本解説では Python を用いた分岐解析ツールについて述べた．位相平面図にかかる試行錯誤支援ツールや変分方程式の見通しのよい表現について例を挙げて詳述した．特に後者においては，反復やスライスも使わずに，連立微分方程式の行列表現と同様のコンパクトな式表現を実現した．Python は計算速度など，インタプリタの抱える悩みもあるが，コーディング時の人為的誤謬の介入が減ることはかけがえがない．

ところで，本文中で紹介した `solve_ivp()` は 2017 年に登場した関数であり，まだ例題も豊富ではなく，最適化や積分アルゴリズムの追加も継続されている．特に許容相対誤差，絶対誤差パラメータにはデフォルトで比較的大きい値が設定されているなど，今後の Python チューニング動向にも注意を払う必要がある．

謝辞　本解説にかかる一部の研究は，JSPS 科研 JP21K04109，並びに，JST ムーンショット型研究開発事業 JPMJMS2021 の助成を受けた．ここに深甚の謝意を表する．

文　献

(1) E.J. Doedel, “AUTO: A program for the automatic bifurcation analysis of autonomous systems,” Congr. Numer, vol.30, pp.265–284, 1981.
(2) Y.A. Kuznetsov, Elements of Applied Bifurcation Theory, 3rd ed., Springer, 2004.

(3) 川上博，松尾次郎，“ダフィング方程式にみられる周期解の分岐集合，” 信学論，vol.J64-A, pp.1018–1025, 1981.
(4) 川上博，吉永哲哉，上田哲史，“力学系の計算機シミュレーション，” 応用数理，vol.7, no.4, pp.303–311, 1997.
(5) 宮路智行，“力学系の数値分岐解析，” 応用数理，vol.32, no.1, pp.16–26, 2022.
(6) C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh, “Basic linear algebra subprograms for Fortran usage,” ACM Trans. Math. Software, vol.5, no.3, pp.308–323, 1979.
(7) E. Angerson, et al., “LAPACK: A portable linear algebra library for high-performance computers,” Proc. ACM/IEEE Conf. Supercomputing, pp.2–11, 1990.
(8) H. Kawakami, “Bifurcation of periodic responses in forced dynamic nonlinear circuits: Computation of bifurcation values of the system parameters,” IEEE Trans. Circuits Syst., vol.31, no.3, pp.248–260, 1984.
(9) 伊藤大輔，“非線形力学系における分岐理論の解析・応用–I, II,” システム/制御/情報，vol.64, no.2 and 4, pp.70–75, 151–156, 2020.
(10) S. Doi, J. Inoue, Z. Pan, and K. Tsumoto, Computational Electrophysiology, Springer Science & Business Media, 2010.
(11) K. Tsumoto, T. Ueta, T. Yoshinaga, and H. Kawakami, “Bifurcation analyses of nonlinear dynamical systems: From theory to numerical computations,” NOLTA, vol.3, no.4, pp.458–476, 2012.
(12) 川上博，上田哲史，C によるカオス CG，サイエンス社，1994.
(13) B.W. Keller, Mastering Matplotlib 2.X: Effective Data Visualization Techniques With Python, Packt Publishing, 2018.
(14) 川上博，小林邦博，“非線形方程式に現れる分岐集合の計算，” 信学論 (A)，vol.J64-A, pp.88–89, 1981.
(15) 川上博，松村利夫，小林邦博，“非線形自律方程式における周期解の一計算法，” 信学論 (A)，vol.J61-A, pp.1051–1053, 1978.
(16) T. Ueta, M. Tsueike, H. Kawakami, T. Yoshinaga, and Y. Katsuta, “A computation of bifurcation parameter values for limit cycles,” IEICE Trans. Fundamentals, vol.E80-A, no.9, pp.1725–1728, 1997.
(17) 上田哲史，吉永哲哉，川上博，陳関栄，“高次元自律系における Neimark-Sacker 分岐の一計算法，” 信学論 (A)，vol.J83-A, no.10, pp.1141–1147, Oct. 2000.
(18) A.T. Fuller, “Conditions for a matrix to have only characteristic roots with negative real parts,” J. Math. Anal. Appl., vol.23, no.1, pp.71–98, 1968.
(19) S. Amoh, M. Ogura, and T. Ueta, “Computation of bifurcations: Automatic provisioning of variational equations,” NOLTA, vol.13, pp.440–445, 2022.
(20) 川上博，勝田祐司，“3 階ダフィング形方程式のホップ分岐とカオス，” 信学論 (A)，vol.J64-A, pp.940–947, 1981.

（幹事団提案，2022 年 9 月 2 日受付，
2022 年 9 月 26 日再受付）

上田哲史（正員：フェロー）
昭 62 高知高専・電気卒．平 2 徳島大・工・電子卒．平 4 同大・大学院工学研究科・博士前期課程修了，徳島大・工・知能情報・助手．平 8 博士（工学）（徳島大）．平 9 講師．平 10 文部省在外研究員 (Univ. Houston)．平 14 徳島大・高度情報化基盤センター・助教授．平 21 教授．現在，同大・情報センター・教授．平 24 本会非線形問題研究専門委員会委員長．平 27 General Co-Chair, Intl. Symp. NOLTA (Hong Kong)．令元 NOLTA ソサイエティ会長．非線形力学系解析に従事．

天羽晟矢（学生員）
平 30 徳島大・工・知能情報卒．令 2 同大・大学院先端技術科学教育部・博士前期課程修了．現在，博士後期課程在学中．令 3.10～現在科学技術振興機構次世代研究者挑戦的研究プログラム．令 3.12～4.2 学術振興会若手研究者海外挑戦プログラム (Institute of Applied Physics, Russian Academy of Sciences)．分岐計算，遅速力学系解析に従事．