現代社会の中の数理、プログラミング教育

安田英典

1　はじめに

数学は古代ギリシャ以来の学問で尊重すべき歴史の一部でもある。一方、現代社会で進む情報化は、世界の数学化でもある。世界は数学による抽象化を経て情報として取り扱うことができるようになる。現代社会の中で、大学の数理的な手法とモデリング、プログラミングコースの教育は重要な役割を果たしている。

現代社会に現れる問題に対応するために必要となる幾つかの数理的な手法がある。それらの中には、特異値分解やデルタ関数のように数学科の学部教育に現れないものがある。学部教育のカリキュラム刷新が望まれている。また、昨今、感染症流行などによって、社会現象のモデリングの重要性が認識された。自然現象の数理モデルは、第 1 原理たる物理法則の持つ普遍的真理の数理化として価値が担保されている。第 1 原理が存在しない社会現象の数理モデルの価値を保証するものは何であろうか。これは、モデリングのコースで取り上げるべきテーマである。さて、昔、筆者が Fortran77 で科学計算プログラムを書いていた時代、プログラミングは計算アルゴリズムを実装するものであった。世界中で同じアルゴリズムを用いたプログラムが何度も書かれたことだろう。そして、プログラミングは文字通りの手作業であった。作図プログラムでは、ペンアップ、ペンダウンまでコーディングした。このような時代はすでに終焉した。いまや、コード生成 AI がプログラミングの手間さえ省こうとしている。教育にも反映すべきだろう。ところで、優れたプログラマーは世界を抽象化してコードを書く能力が卓越しているといわれる。そうでない平凡なプログラマーの作成したプログラムは、しばしば、スパゲッティと呼ばれるものになる。プログラミングコースで抽象化能力を身につけることが重要なのは間違いない。数学科の教育に求められるものでもあろう。

城西大学に奉職して 20 年を迎える。それ以前は企業に２６年間在籍した。立場が異なると見える景色も異なってくる。本稿では応用数学、情報数理を志向する紀尾井町キャンパスの教員という立場から数理、プログラミング教育について私見を述べてみたいと思う。

2　数理的な手法

2.1　特異値分解：長方形行列の"逆行列"

齋藤正彦著”線型代数入門”[1] は線型代数の代表的なテキストである。1966 年 3 月 31 日初版発行で一度も改訂されずに刷数を上げている。我が国で出版されている多くのテキストは同書の影響を受けており、線型代数の教育体系は歴史的に完成されたものとみなされているようである。しかしながら、同書について著者は以下のようにコメントしている。

"そのころ，友人からの刺戟もあって，私はアルゴリズムというものに関心をもっていました．"、"この本が出たころ，ある人から何のために書いたのかと聞かれ，私は「十年後の日本の技術水準を上げるためだ」と答えました。[2]"

著者がコンピュータの時代を意識していたことは明らかと思われる。しかし、著者の意図と同書の世評との間にはギャップがあるように見える。同書の発行より半世紀以上経った現在、線型代数の教育を取り巻く状況は大きく変わっている。特に、WWW(ワールドワイドウェッブ) をはじめとする情報環境の変化である。ネット上の情報から生まれてくる巨大な行列が線形代数の教育の変革を迫っている。その中でも喫緊の課題は一般化逆行列の初年次教育への導入である。いま、人に関するある情報を得るために百万元の連立方程式を解く必要があるとしよう。百万人からデータをとって百万個の方程式をたてようとしたとき、もう千人ほどデータ提供者が現れたらどうしたらよいのだろうか。統計学のテキストには最小二乗法で正規化方程式を作れと書いてある。要は、係数行列の転置行列との積とって係数行列を正方形にするのである。この手法は行列が小さいうちは有効であろう。巨大な行列同士の積と分解は高コストである。現在もっともよい解法は一般化逆行列と特異値分解 (SVD, singular value decomposition) のアルゴリズムである。

（注）一般化逆行列 $A^\dagger$ は次の性質を持っている。$AA^\dagger A = A,\ A^\dagger AA^\dagger = A^\dagger$. 一般化逆行列の表示には、通常、印刷記号の dagger † を添字に用いる。一例をあげる。

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix},\quad A^\dagger = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

写像 Z の特異値とは、要するに $\sqrt{ZZ^*}$ の固有値のことをいう。特異値は行列に限らずヒルベルト空間のコンパクト作用素に対しても計算される [3, 4]。$m \times n$ 実行列 A の特異値分解は $A = U\Sigma^t V$ と書ける。ここに、U は m 次直交行列、V は n 次直交行列で、Σ は対角成分が正の実数で他の成分が零の $m \times n$ 行列である。Σ の対角成分が特異値であり、$\sigma_{1,1} \geq \sigma_{2,2} \geq \sigma_{3,3}, \geq \cdots \geq \sigma_{r,r},\ r \leq \min(m,n)$ と

降順に並ぶ。

特異値分解を用いた一般化逆行列の計算は応用分野では古くから知られている。筆者はかって、性質の極めて類似したタングステンとモリブデンを含む数種類の金属データの解析に使用して良好な結果を得た。小さな一次方程式系であったが、数値がほとんど同じ行が現れてガウスの消去法では計算できなかったので、計算誤差以下の小さな特異値を零と置いて一般化逆行列を求めた。以前は、特異値分解はこのような問題の対処法と位置付けられていた。いまや、特異値分解は機械学習をはじめ多くの分野で巨大な長方形行列に適用されている。特異値分解の計算法は簡便なものではないが、プログラミングするのに手間はかからない。例えば、Python で行列 A の特異値分解を求めるためには、数値計算の numpy ライブラリーを呼んで、”u,s,v=np.linalg.svd(A)”とするだけである。特異値分解の計算法そのものは現在も開発が行われているが、ユーザは計算法の詳細に立ち入る必要はない。

一般化逆行列、特異値分解は通常の線型代数のコースでは扱われない。一般化逆行列、特異値分解を扱った線形代数のテキストは目の届いた範囲ではストラングの邦訳が数点あるだけであった。例えば [5] である。工学系のセミナーでの使用を想定したテキストには一般化逆行列、特異値分解に力点を置いているものはある [6]。米国の大学には、Linear Algebra と Numerical Linear Algebra を併設したカリキュラムを持つところがある。Numerical Linear Algebra の主なテーマは、1 次方程式の高速解法や固有値問題のアルゴリズムであり、特異値分解も大きなテーマとして扱われている。Numerical Linear Algebra の代表的なテキストとしては [7] が挙げられる。

ところで、一次方程式の解法は、共役勾配法を発展した高速解法の開発によって 1980-1990 年頃に大きく変わった。このことも、我が国の教育には反映されていない。GMRESS 法（generalized minimal residual method）などの高速解法ではクリロフ部分空間の直交基底の計算がポイントとなるが、コンピュータの発展によってグラムシュミットの直交化の計算量は大きな問題ではなくなった。一方、行列の構造をほとんど利用しないヤコビ法、ガウス-ザイデル法、SOR 法などの反復法は歴史的なものとなってしまった。なお、ガウスの消去法のような計算量の多い直接法は使える局面が限られている。

筆者は Numerical Linear Algebra のテーマをいくつかの授業の中で取り上げているが、一つのまとまったコースにする方が望ましいと考えている。20 世紀後半、ソリトン、カオスやフラクタルのようにコンピュータの処理能力の向上を背景にした非線形数学の興隆があった。そして、世紀が変わりネットワークが世界を覆うようになると、WWW 上の膨大なデータの出現によって新たな線形数学の時代が出現したと思う。線型代数はジョルダン標準形の講義で終わるというのは過去の話となった。そして、何よりも Deep Learning（深層学習）が新たな線型代数を必要としている。

（注）クリロフはロシア帝国海軍の将官でバルチック艦隊の主な戦艦の設計者でもある。クリロフ部分空間は造船工学の問題を計算するためにコンピュータ以前の時代に生まれた。

2.2 デルタ関数：微分方程式の計算

数学のテキストで優先されるのは分かりやすさより厳密性だろう。しかし、次のような前書きのある微分方程式のテキストもある；”電磁場の量子論の厳密な定式化ができました．その結果、電荷が存在しないことが厳密に証明されました [8]。” 著者は物理学者である。ニュートンの運動方程式、熱方程式、波動方程式、シュレーディンガー方程式、アインシュタインの宇宙方程式など多くの微分方程式は物理法則を支配あるいは記述する方程式である。物理には方程式から現象を導くという言葉があるが、現象を導くためには方程式を計算する必要がある。現在、ほとんどの微分方程式の計算はコンピュータを用いた数値計算だが、ここでは”厳密な”計算について考えてみたい。世間を見渡すと応用数学と銘打った数学のテキストに於いて、微分方程式の厳密解を求めるために使われているのは”分かりやすい”ラプラス変換が相場であろう。しかし、伊藤清先生は昭和の前半に次のように述べられている。

”我々はできる限り本質を把握し易い形でさかのぼることが望ましい。例えば、Heaviside calculus の基礎付けは Laplace 変換によっても可能であるが、L. Schwartz の distribution による方法の方が優れているのは、Heviside calculus のもつ味わいが少しもこわされずに基礎付けられているからである。[9]”

超関数 (distribution) というとすぐに思い浮かぶのはデルタ関数であろう。”ディラックのデルタ関数”は、量子力学の建設者の一人であるディラックによって次の形で導入された [10]。

$$\int_{-\infty}^{\infty} \delta(x)dx = 1, \quad x \neq 0 \text{ に対しては } \delta(x) = 0.$$

明らかに”ディラックのデルタ関数”は数学の意味では関数ではない。ディラックは半ページほどのヒューリスティックな議論で次の公式を導いている。

$$\frac{d}{dx}\log x = \frac{1}{x} - i\pi\delta(x)$$

多くの数学のテキストではデルタ関数を次のように導入する。無限回可微分でかつコンパクトな台を持つ関数のなす線型空間 $\mathscr{D}$ 上の連続な線型汎函数を超関数という。すべての $\phi \in \mathscr{D}$ に対して $\delta(\phi) = \phi(0)$ あるいは $(\delta, \phi) = \phi(0)$ によってデルタ測度（デルタ関数）を定義する。取りつく島もないが、もう少し丁寧なテキストではデルタ関数に収束する関数列を作ってディラックのアイデアに沿った導入を行う。しかし、収束の意味するところに数学的な困難が隠れている。超関数の意味での対数関数の微分につい

ては、デルタ関数とコーシー積分との類似性に着目した複素解析風の計算、あるいは、実解析での発散積分の有限部分つまり主値の計算から次の式が導かれる。

$$\frac{1}{x \pm i0} = \mp i\pi\delta(x) + \mathscr{P}\frac{1}{x} \quad \text{(複号同順)}$$

シュワルツは次のように述べている；”Dirac が彼の有名な関数 $\delta(x)$ を導入して以来、演算子法の諸公式は数学者の厳密性には益々受け入れが難いものとなった。$\cdots$ 理論的な部分はどこでも、一般位相幾何学と関数解析（位相ベクトル空間）をかなりよく知っている必要がある。[11]” しかし、ラプラス変換で微分方程式の計算するとき位相に関する議論は必要なかった。微分方程式の計算では、超関数の位相に関する煩雑な議論はスキップできるのではないだろうか。心強いことに、偏微分方程式論の古典といえる溝畑茂先生のテキストには、超関数の位相は単純位相でも”応用上の不便はあまりない”と書かれている [12]。また、distribution とは別の超関数の流れであるロシア風の generalized function のテキストでは位相の議論はあまり出てこない [13, 14]。ディラックの”量子力學”[10] の第 1 版前書きの日付は 1930 年 5 月 29 日となっている。シュワルツの原著 [11] の発行は 1951 年である。20 年間、多くの人は位相のことは気にせず使っていたということになる。

さらに、シュワルツは次のようにも述べている；”Heaviside の関数の微係数が、数学的には定義の矛盾した Dirac の $\delta(x)$ であると言ったり、$\cdots$ 我々に許される範囲を逸脱することである。[11]” この逸脱を回避あるいは解消するためには、超関数とテスト関数の空間の導入が必要になる。そして、フレシェ空間とその共役空間の位相が現れる。さて、話をデルタ関数に戻そう。いまでもヘビサイドの階段関数を微分してデルタ関数とするという素朴なアプローチをとることがある。高校物理での質点の導入のようなものである。しかし、この導入では超関数にはつながらない。位相に立ち入らずに、微分方程式の基本解を求める計算を使ってデルタ関数を構成するというアプローチがある [15]。通常の基本解の構成とは逆の順序である。このアプローチの利点は、微分方程式のコースの早い段階から自然な形でデルタ関数、そして超関数を導入できることにある。早い段階で超関数を導入するメリットは大きい。例えば、差分法や有限要素法などの数値解の収束の議論は見通しの良いものになる [16, 17]。もう少し踏み込むと、数値的な手法をふくむ微分方程式のコースでは超関数を含めた関数解析的手法をなるべく早い段階から導入するのが望ましい。実際の問題では収束がどの意味であるかも重要となる。構造物の熱応力の評価が必要ときに熱方程式の数値解が厳密解に L^2 で収束しても、その数値解は熱応力の計算には使えない。W_2^1 あるいは H^1 の収束が必要である。この観点からも関数解析が必要となる。

関数解析の一環として数値解析を扱うという立場もある [18]。極端な立場であることは数値解析の歴史をみると明らかであるが [19, 20]、一縷の真実は含んでいる。また、現代的な数値解析の誕生はコンピュータの発明を機縁とする立場 [21] は [18] に近い。反対側の極端な立場をとると、コンピュータ

シミュレーションの視点から関数解析を位置付けるということもできる。具体的な数値計算に収束などの意味を与えるのは抽象的な関数解析である。例えば、台形公式などの数値積分の公式 (cubature formulas) の収束の意味を知るためには、積分値そのものではなく、一様有界性の原理あるいは共鳴定理の形でも用いられる Banach-Steinhaus の定理から観る方が分かりやすい。最も古い数値計算法は紀元前 18-16 世紀と推定されている日干しレンガに楔形文字で刻まれた古代バビロニアの平方根のアルゴリズムであろうが、このアルゴリズムの収束の意味を理解するのには集合や位相など近代の数学が必要となる。そして、ニュートン法でもあるこのアルゴリズムは現在でも使われているのである。コンピュータ時代の”新しいぶどう酒”を古い革袋に入れてはいけない。近年、応用数学の観点から体系化された優れた関数解析のテキストも多くなってきた。例えば、[22] があげられる。なお、学部レベルの（純粋）数学の微分方程式のテキストには、微分方程式の関数解析的アプローチは現れないか [23]、現れてもテキストの後半になる [24, 25]。むしろ、物理や工学の微分方程式のテキストには躊躇なくデルタ関数が現れている。

（注）偏微分方程式の数値解析の心の支えである Lax の同等定理 consistency+stability=convergence は超関数の意味のフーリエ解析で了解される。例えば、[16] を参照。

3　社会現象のモデリング

3.1　モデルと現実

ダニエル・ベルヌーイは”人々の幸福に関わるような事柄について、わずかばかりの解析や計算が提供できる知識を使うことなく決めることなどないように切に願います”との述べている [26]。種痘以前の天然痘の時代に、ベルヌーイは確率論を用いて予防接種の是非を論じた。このベルヌーイの数理モデルを用いた議論に対して、ダランベールは予防接種によって死ぬという極めて小さいが身近な危険と自然の天然痘にかかって死ぬというはるかに大きくはあるが遠くの危険との比較が欠如していると論難した。ラプラスはこの論争を政府にとっての予防接種の利点と愛する家族を持つ父親の心配の問題と評した [27]。

1920-30 年代に、ケルマックとマッケンドリックは地域社会での急速かつ短期的な感染症の流行を対象とした一連の論文で以下の SIR モデルに結実する数理モデルを発表した。

$$\frac{dS(t)}{dt} = -\beta S(t)I(t),\ \frac{dI(t)}{dt} = \beta S(t)I(t) - \gamma I(t),\ \frac{dR(t)}{dt} = \gamma I(t).$$

ここに、S は感染可能な人口、I は感染した人口、R は回復または隔離した人口、β は感染率、γ は回復率または隔離率である。

現在用いられている感染症流行の数理モデルの多くは SIR モデルと言われるもので、ケルマック・マッケンドリックモデルをベースとしている。SIR モデルは常微分方程式系で記述され、線型化すると感染人口のマルサスの法則となる。このマルサスの法則から 1 人の感染者からの 2 次感染数である基本再生産数 R_0 が求まる。よく知られているとおり、新型コロナの流行対策の策定では基本再生産数が重要な役割を果たした。SIR モデルは簡単なモデルであるが流行の局面に合わせて種々の変形が行われる。一例として、手前味噌だが、我が国の新型コロナ第 4 波に適用したケーススタディをあげる [28]。

SIR モデルをはじめとする数理モデルは、感染症流行のすべてをモデル化することはできないが流行のある局面は反映されているので、感染症対策の策定に用いることができると一般に考えられている。しかし、流行のある局面、多くの場合は特定の地域や季節に依存しているモデルには普遍性がないという意見が表明されることがある。モデルは普遍的真理の反映であるべきという自然科学的な立場である。このことについて、Koopman ミシガン大学教授の卓越した見解を紹介する。

”意思決定のためのモデルの検証は科学的なモデルの検証とは異なる。モデルが過去のデータを再現することや実際と異なった点があることを持ってモデルの可否を論じることはできない。感染症流行モデルには現実を単純化する仮定が用いられているので、モデルの検証は重要である。モデルの仮定は現実を単純化しているが、それでも問題となる局面を忠実に反映する必要がある。忠実という意味は、モデルに他の現実的な要件を加えても結果に影響がないことをいう。モデルは現実を単純化しているので完全な検証はありえず、検証できるのは対策の意思決定に齟齬をきたす仮定があるかという点である。[29]”

3.2 モデルの中の世界

モデルと現実の関係についてもう一つ別の考え方がある。吉倉廣国立感染症研究所元所長は次のように述べている。

”過去の大流行に基づく予測は、社会構造、自然および都市環境などが大きく変わっているので限界があり、数理的な予測はこれを補うためのものである。数理シミュレーションに対しては、現実との接点がない、実際の流行でモデルが検証できていない、というのが一般の批判である。これは一見妥当な批判のように見えるが、数理モデルにはモデルとして 1 つの世界があり、そのモデルの世界からどのような問題あるいは視点が現実の世界に投げかけられるかということの方がはるかに面白い。[30]”

ある多人数参加型のオンラインゲームでアップデートの際のバグによって、ゲーム世界の中に高病原性の感染症が発生した。Corrupted blood 事件と呼ばれるこの出来事では、バーチャルなゲーム世界に参加しているプレイヤー達の”現実的な行動”が一種の社会実験として研究者の注目を集めた [31]。特殊なケースだが、モデル世界からの現実世界の問題への投射の一例といえる。

１８世紀半ばの天然痘予防接種の時代から、予防接種の是非は数理モデルを用いて議論されてきた。世界に感染症が流行したとき、無人島に相棒と二人で避難するとしよう。食料などのこともあるので世界との接点を完全に無くすわけにはいかない。皆さんは相棒に予防接種を受けて欲しいと思わないだろうか。そうならば、自分は予防接種を受けなくてもすむだろう。孤島の二人ならばともかく、地域社会の中で我が身を守るためにはどうしたらいいのだろう。感染症も怖いが、ワクチン接種も怖い。個人の合理的判断を追求する数理としてゲーム理論がある。ゲームの解はナッシュ均衡によって与えられる。このワクチン接種ゲームのナッシュ均衡の研究は、地域社会の数理モデルのコンピュータシミュレーションによって遂行された [32]。

（注）エレベーターで立ち止まるのは右側か左側かという問題の解はナッシュ均衡によって与えられる。ナッシュ均衡では相手が戦略を変えないとき自分だけ戦略を変えると不利益を被る。個人合理性を追求するナッシュ均衡に対して、全体最適化を目指すパレート効率的という経済学的な概念がある。有名な囚人のジレンマではナッシュ均衡とパレート効率的の対立が現れる。ワクチン接種ゲームもこの対立構造を研究する。

4　科学計算とプログラミング言語

4.1　Python の計算コード

IT 業界のビッグネームである Paul Graham は言う；”ハッカーの一部は、自分の慣れた言語を好み、ほかのすべてを嫌う。他のハッカーの一部は、すべての言語は同じだという。真実はこの両極端の間のどこかにある。· · · この分野はまだ進化途上にある。[34]”　コンピュータと一緒に生まれたといってよい、言わば自然発生したプログラミング言語がある。数値計算の Fortran(Formula translation) と記号微分の Lisp(List processor) である。Fortran はスーパーコンピュータの世界を開拓し、Lisp は AI の世界へ繋がっていく。爾来、数多くの言語が生まれそして滅んでいったが、この２つの自然発生言語が滅びることはないだろう。しかし、こんにち大学のプログラミングコースでこの２つを扱うことはほとんどない。注目を集めているのは Python である。クリスマス休暇に生まれた Python は最小限を提供する言語と言われる。Python で専門的なプログラミングを行うときは多彩なライブラリーをインターネットからダウンロードする必要がある。深層学習ならば TensorFlow などである。多くのライブラリーは Python 以外の言語で書かれている。

さて、ロトカ・ボルテラ方程式を Python で計算してみよう。ここでは、科学計算のための scipy ライブラリーの中から常微分方程式ソールバーである solve_ivp を用いる。計算コードと計算結果のプロット図を以下に示す。

計算コード

```
#ロトカ・ボルテラ方程式の右辺を定義する
def lotkavolterra(t, z): #x fish, y shark
    x, y = z
    return [1.5*x - x*y, -3*y + x*y]
#常微分方程式の初期値問題の解を計算する: 計算時刻 [0,30], 初期値 [10,5]
from scipy.integrate import solve_ivp  #scipy の読み込み
sol = solve_ivp(lotkavolterra, [0, 30], [10, 5],
                dense_output=True)     #微分方程式を数値的に積分する
```

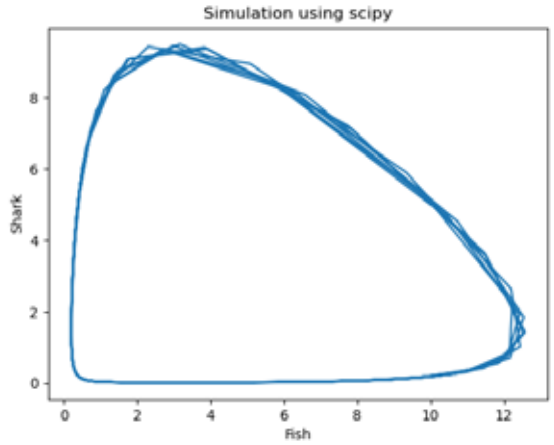

図 1　ロトカ・ボルテラ系：scipy の計算

プロットのためのコードは以下である。numpy と matplotlib の 2 つのライブラリーを使っている。

プロットコード

```
#プロット用に時刻 30 までのデータを等間隔の時刻 300 点で補間する
import numpy as np  #numpy の読み込み
t = np.linspace(0, 30, 300)
z = sol.sol(t)      #数値解の補間点での値を求める
#(Fish,Shark) の相図を作図する
import matplotlib.pyplot as plt  #matplotlib の読み込み
```

```
xp = z.T[:,0]
yp = z.T[:,1]
plt.plot(xp,yp)  #相図をプロットする
plt.xlabel('Fish')
plt.ylabel('Shark')
plt.title('Simulation using scipy')
plt.show()       #プロット図を出力する
```

上のロトカ・ボルテラ系のコードは次の scipy.org のドキュメントから取った。可読性を上げるため一部変更した; https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html . なお、計算とプロットに必要なのは上記コードだけである。C 言語で書いたら OpenGL でプロットするとして 5 ページは超えるだろう。

もう一人の IT 業界のビッグネームである Larry Wall は言う ; ”コンピュータが怠慢なときに感じる怒り。この怒りの持ち主は、今ある問題に対するプログラムにとどまらず、今後起こり得る問題を想定したプログラムを書く。少なくともそうしようとする [35]。” 職業としてプログラミングする人々はかくあるべきだろう。しかし、このことに関心のある数理科学の研究者では稀である。Larry Wall の主張するところは、Python ではライブラリーが受け持つ。ライブラリーができることをコーディングすべきでない。Python の標準的なコースでは、numpy, matplotlib, pandas, scipy, scikit-lean などの Python を支えるライブラリーを取りあげる必要がある。

(注) ここで使用したライブラリーのうち scipy は最適化、微分方程式などの専門分野の数値計算アルゴリズムを提供する。numpy は Python の科学計算の基盤レベルのアルゴリズムを実装しているライブラリーである。numpy の決定的な点は Fortran 風の多次元配列のクラス (ndarray) を有していることである。例えば、2 次元配列は Python だけでは 1 次元配列を要素とする 1 次元配列で記述するが、行列計算の速度は極端に低下する。matplotlib は 2 次元と基本的な 3 次元の作図を行うライブラリーである。

4.2 科学計算とライブラリー

常微分方程式については、scipy は solve_ivp と odeint の 2 つのインターフェイスを持っている。solve_ivp が推奨されている新しいインターフェイスであるが、いまでも odeint を用いることは多い。odeint は 1980 年代に Fortarn77 で書かれたライブラリー ODEPACK[33] のいわゆるラッパーである。ODEPACK は単段法のルンゲクッタ法、多段法のアダムス法 (predictor-corrector method)、硬い方程式のための後退微分法 (BDF) やギア法などの複数のアルゴリズムで構成されている。ODEPACK は、C/C++ にも移植されており、長年の実績によって精度、頑健さなどの数値計算上の性能についての信頼

がある。筆者は、30 数年前、自分の Fortarn コードで ODEPACK を利用していた。それ以前はルンゲクッタ法などの ODE ソルバーを自作した。ODEPACK を使用するにはアルゴリズムの選択、各アルゴリズムのパラメータの設定など数値解析の知識が要求されるが、scipy のインターフェイスはその大部分をユーザから隠している。scipy を利用すれば、数値解析の知識を要求されずにプログラムを作成できる。この場合、Python の主たる役割は”糊付け”ということになる。コンピュータ科学の半世紀に及ぶ蓄積で実用上のことはおおむね”糊付け”で対応できるようになった。車輪の再発明を行う時代は終わったといえる。

ところで、数値計算のための Fortran をプログラミングコースで扱うべきかという悩ましい問題が残っている。Fortran77 だけでなく Fortran90/95 以降の Fortran2018 あるいは 202x なども含めての話である。科学計算のヘビーユーザは、HPC(ハイパフォーマンスコンピューティング) のために C/C++ と並んで Fortran も必要となる。しかし、いまや大型の科学計算を行う計算機ユーザは少数派となった。これは糊で貼られる方の教育をどうするかという問題でもある。例えるならば、ユーザーのためにワインを選んで勧めるのはソムリエたる Python だが、ワインを醸造するのはブドウ農家という話だろう。

数値計算ライブラリーの資産の多くは Fortran で書かれている。Fortran ライブラリーの多くは原子力コードとスーパーコンピュータが必要とされたいわゆる原子力時代に開発されて、米国エネルギー省 (DOE) 傘下のアルゴンヌ国立研究所 (ANL) のコードセンターから公開された。IT 世界でデファクトスタンダードとなっている BLAS(Basic Linear Algebra Subprograms) も ANL から公開された。1970 年代に線型代数ライブラリー LINPACK を開発する中で生まれた初期の BLAS は FORTRAN77 で書かれた素朴なアルゴリズムのコードの塊だったが、並列計算がメインとなった現在の BLAS はアルゴリズムもコード構成も複雑である。ビッグサイエンスのためのスーパーコンピュータの開発は、21 世紀の AI、宇宙の時代を迎えても続き、米国、中国、日本をはじめ各国政府は多額の資金を投入している。HPC のためのブドウ農家を育てるのもスーパーコンピュータと同様に国家レベルの課題であろう。

4.3 プログラミングの抽象化

Python のようなライブラリーが充実している”糊付け言語とはプログラミングのスペクトルの逆の端にある ··· 難しい問題を解く洗練されたプログラム [34]” のための言語として Lisp がある。Lisp は、もともと、プログラミング言語ではなく万能チューリングマシン、つまり、機械知性（Artificial Intelligence）の一つのモデルとして開発されたものであり、プログラミング言語の中では特異な立ち位置を占めている。Lisp の開発者であるジョン・マッカーシーは次のように述べている。

”Lisp がチューリングマシンより扱いやすいモデルであることを示すもう一つの方法は、万能 Lisp 関数を書いてそれが万能チューリングマシンの表記より簡潔で分かりやすいことを示すことだった。··· そ

の表記はあくまで論文のためのもので、実際に Lisp プログラムがそれで書かれるようになるなんて考えもしなかったよ。[34]”

さて、プログラムの基本的な手続きは判断と繰り返しで構成されている。多くの言語では、通常、繰り返しは反復 (iteration) ループで記述する。他方、Lisp では再帰 (recursion) を使うことが多い。反復ループは繰り返されるアルゴリズムの外側に記述されるが、再帰では繰り返しはアルゴリズムの中に現れる。再帰を用いたプログラムの方が反復ループを用いたものよりも夾雑物が少なくなる。Lisp の教育分野の方言である Scheme で、ユークリッドの互除法の手続きを書くと次の関数になる [36]。

```
(define  (gcd  a  b)
          (if  (=b 0)
                a
                (gcd b  (remainder a b))))
```

手続きに余分な記述はなくアルゴリズムそのものである。関数の引数 a, b は特定の数ではなく、ユークリッドの互除法の手続きを gcd という関数名で抽象化している。さらに、Lisp の関数は、引数に変数だけでなく手続きをとることができる。また、値として手続きを返すこともできる。このような手続きは高階手続きと呼ばれる。高階手続きの抽象度は高い。歴史的にユークリッドの互除法は対象を広げていき、紀元前 5 世紀のギリシャ人は自然数を対象としたが、1600 年頃ステヴィンは対象を多項式に拡張した。その後、19 世紀になるとガウス整数なども対象になった [37]。優れたプログラマーならこれらのアルゴリズムを統一的に扱うジェネリックプログラミングを試みるだろう。

プログラムを抽象的に記述することは重要である。抽象度の高い手続きの特殊化としてのプログラミングを行えば、”今ある問題に対するプログラムにとどまらず、今後起こり得る問題を想定したプログラム”を作成できる。Lisp はプログラミングで求められる抽象化能力を高めるコースに適している。優れたプログラマーは、プログラミングに際して実行効率やメンテナンスなども考慮して、必要に応じた抽象度を選択することができる。Lisp は学生が手続きによる抽象化 (abstraction with procedures [36]) という情報科学の”深淵”の一つに触れることを助けるだろう。

（注）Lisp は方言の多い言語である。ビジネスでは Common Lisp が使われるが教育の場では Scheme が選ばれることが多い。なお、実際の計算では反復の方が実行効率がよい。再帰は可能ならば末尾最適化などの手法でコンパイル時に反復に変換される。そうでないとき、再帰を選ぶか反復を選ぶかはプログラマーの判断となる。

5　おわりに

数学者の方々にどのようなものが応用数学のテーマかと尋ねると、数学会の応用数学分科会の発表などを思い浮かべられるようである。しかし、応用数理学会、計算工学会、シミュレーション学会などでは、数学会の応用数学分科会は純粋数学の発表の場と見なされている。この応用数学に対する認識に関する違いはどこから来るのであろうか。トレフェゼンは”An applied mathematian's apology”の中で以下のように述べている。

”数学は一つであり、純粋数学と応用数学の違いは幻想であるという見解が世に広まっています。これはナンセンスです。私の経験では、これは一般に純粋数学者が持っている意見です。彼らはしばしば自分自身は応用数学者でもある思い込んでいたり、あるいは、その気になれば簡単に応用数学者になれると思っています。皆さんは、昆虫学者（entomologist）と語源学者（etymologist）の違いは何ですか?というジョークをご存知でしょうか。語源学者は違いを知っています。 これは数学にも当てはまると思います。 純粋数学者と応用数学者の違いは何でしょうか? このエッセイの目的は、数学は一つであるなどと言い張らずに、数学の多くの部分の間のより良いコミュニケーションを促すことです。[38]”

明治以来、我が国では応用数学は主に工学部で教授され、数学科では純粋数学に偏重した教育が行われてきた。このためか、我が国の純粋数学者には応用など片手間でできると思われている方々がおられる。しかも、しばしば、自分は応用数学にも取り組んできたと言われる。さらに、現実社会での数学的な問題は出来合いの数学の演習問題であると思い込んでいるので、問題の定式化、つまり、モデリングにおいて対象分野のドメインノリッジなどたいして必要はなく、モデルを解析した結果は数学的に正しいから対象分野の知見を用いた検証などは 2 次的なものであるとまで主張される。まるで、数学の果てが世界の終わりのようである。応用数学あるいは数理科学を純粋数学の一分野だと思っているのだろう。

冷戦時代、旧ソビエトの卓越した数学者は国家から工学の問題に取り組むことが要請された。盲目の幾何学者ポントリャーギンは制御工学の分野で優れた業績を上げた。しかし、ポントリャーギンのスタイルは多くの純粋数学者が思い描くようなものではなかった；”私たちはエンジニアを講師に迎えましたが、装置について調べたことに基づいて彼らが得ている純粋に数学的な問題を、私どもに提示することを堅く禁じました [39]。”

応用数学（applied mathematics）は応用（applied）と数学（mathematics）の熟語である。次の指摘がある；”また、「応用された数学」という表現で強調されているのは、「数学」という言葉ではなく、「応用されている」という点であることも重要である [40]。” 数学と応用の間には然るべき距離がある。よく知られている通り、文明開化の折に明治政府が西洋から学問を輸入したとき数学では純粋数学だけが持ち込まれた。日本数学会は 1877 年に設立された東京数学社を起源として 1946 に設立された。しか

し、数学者社会での応用数学の組織化は遅く、各国の応用数学の組織は米国に本部を置く SIAM(Society for Industrial and Applied Mathematics) と連携する形をとるが、SIAM と連携する日本応用数理学会 (JSIAM) が発足したのは 1990 年である。なお、台湾で同様の学会が発足したのは 2012 年と聞いている。数学者社会では純粋数学がはじめに組織化されかなりの時間を置いて応用数学というのは、西洋からの輸入学問であるアジアの数学に共通の発展形態かもしれない。

(注) 一般的な数学教育については日米あまり違いがないようである。”自分がアメリカで受けた数学の授業をふり返って考えると、もう少し Applied Mathematics に時間を割いて教えてもらえたらよかったと思います。単純な方程式を解くだけでなく、社会で起こっている事象に数学を応用して問題を解く。そのような数学の使い方をもっと教えたほうが良いのではないかと思います。”グレン・S・フクシマ元米国大統領府通商代表部日本担当部長; https://www.su-gaku.biz/about/interview/no10.php.

さて、トレフェゼンがハーディの”A mathematian's apology”[41] を踏まえていることは言うまでもない。ハーディは純粋数学、応用数学について多くのことを述べている。純粋数学者の中にはハーディの”応用数学とは幻想である”を好む向きもあるが、この言葉は、まるで禅の公案のようである。ハーディの名は集団遺伝学のハーディ・ワインベルク平衡、あるいは、ハーディ空間で制御工学の H^∞ 制御理論が構築されるなど生物学や工学など純粋数学以外のいろいろなところにも現れる。ハーディが弁明しているのは、純粋数学と応用数学の違いなどではなく数学の有用性、特に、”戦争に対する数学の有用性”についてである。第二次世界大戦中、英国ではチューリングをはじめ数学者はドイツのエニグマ暗号の解読や OR(オペレーションズリサーチ) の契機ともなった広く知られている U ボートに対する機雷投下戦略の策定などに多大な貢献を行った。ハーディの幻想は単なる幻想ではあるまい。トレフェゼンには応用数学者としてのハーディについてのエッセイもある [42]。

(注) ハーディはクリケットの試合観戦中にハーディ・ワインベルク平衡の着想を得たと伝えられている。

最初のフォンノイマン型コンピュータである 1 万 7 千本の真空管を持つ ENIAC は 1946 年 2 月 14 日夕方に完成し、1955 年 10 月 2 日午後 11 時 45 分まで稼働したと伝えられている。1956 年、政治哲学者のハンナ アレントは以下のように述べている ; ”ここで重要なことは、近代の初め、人びとが、まだプラトンのように、宇宙の数学的構造を信じていたということではない。また一世代後に、人びとが、デカルトのように、一定の知識は、精神がそれ自身の形式と定式を相手にするときのみ可能であると信じたことも重要ではない。むしろ決定的なのは、まったく非プラトン的に、代数学的処理の下に置かれたということである。[43]” 筆者は、非プラトン的な代数学的処理の最たるものはコンピュータで実行されるアルゴリズムであり、それはプログラミングによって実装されると考えている。ハンナ アレントはさらに続ける ; ”ここで周知の「科学を数学に還元すること (レドウクチオ・スキエンチアエ・アド・マテマチカム)」によって、感覚的に与えられるものは、数

学方程式の体系に置き換えられ、すべての現実的関係は、人工的なシンボル間の論理関係に解体される [43]。”

量子論はシュレーディンガー方程式とヒルベルト空間によって単なる論のレベルから力学たる量子力学へと進化して原子力時代を生み出し、いまや、量子コンピュータの開発が始まっている。生命科学も物理科学と同様の道を進んでいるようである。メイナード スミスは述べている；”数学は生物学において、この上なく役に立ちます。なぜなら数学は私たちに、あくまで正確さを求めるからです [44]。”遺伝子情報は応用数学、情報科学の対象でもある。ノーベル賞を受賞したカタリン カリコ博士は新型コロナウィルスの mRNA ワクチンの設計を極めて短時間で行ったと伝えられている。 ウェットな生命科学の中に、コンピュータの時代に現れたドライな応用数学、情報科学の力と思われる。

これからも、数学、特に応用数学によって、現象論が解体と同時に体系化され、そして、情報科学によってテクノロジーへと昇華していくのであろう。一方、コンピュータのネットワークが世界を覆ってからすでに久しい。世界は情報化によって解体され、ネットワークの中のサイバー空間で再構成される。そして、機械知性の果てしない可能性が人々の未来に希望と恐怖を投げかけている。サイバー空間と現実空間が高度に融合した Society 5.0 で自由闊達に活躍できる人材育成が求められている我が国に於いて、トレフェゼンやアレントのような観点を踏まえて、応用数学、情報数理を志向する紀尾井町キャンパスにおける新しいカリキュラムを検討・刷新することが望まれる。

参考文献

[1] 齋藤正彦: **線型代数入門**, 東大出版会, 1966.

[2] 齋藤正彦: https://www.mathsoc.jp/assets/pdf/publications/tushin/backnumber/1102/112saito.pdf. 2023/8/4 アクセス.

[3] P. D. Lax: *Linear algebra*, Wiley, 1977. (see pp. 140.)

[4] P. D. Lax: *Functional analysis*, Wiley, 2002. (see pp. 329.)

[5] G. ストラング, 山口昌哉（監訳）, 井上昭（翻訳）: **線形代数とその応用**, 産業図書, 1978.

[6] 金谷健一: **線形代数セミナー**, 共立出版, 2018.

[7] L.N. Trefethen, D. Bau III: *Numerical linear algebra*, SIAM, 1997.

[8] 中西:**微分方程式**, 丸善出版, 2016.

[9] 伊藤清: **数学の基礎としての集合論**, 科学基礎論研究, Autumn, 8-12, 1954.

[10] ディラック; : **量子力學 原書第４版**, 岩波書店, 1968.

[11] L. Schwartz: **超関数の理論**, 岩波書店, 1971.

[12] 溝畑茂: **偏微分方程式論**, 岩波書店, 1965. (see pp 48)

[13] ゲリファンド, シーロフ, 功金二郎. 井関清, 麦林布道: **超関数論入門** I,II, 共立全書, 1963.

[14] V.S. Vladimirov: *Methods of the theory of generalized functions*, Taylor & Francis, 2002.
[15] R.M.M Mattheij, S.W. Rienstra, J.H.M. ten Thije Boonkkamp: *Partial differential equations*, SIAM, 2005.
[16] 山口昌哉, 野木達夫: **数値解析の基礎**, 共立出版, 1969.
[17] 田端正久: **偏微分方程式の数値解析**, 岩波書店, 2010.
[18] 柴垣和三雄: **関数解析と数値解析入門**, 森北出版, 1973.
[19] H.H. Goldstine: *A history of numerical analysis from the 16th through the 19th century*, Springer-Verlag, 1977.
[20] C. Brezinski, L. Wuytack (Eds.): *Numerical analysis: historical developments in the 20th century*, North-Holland, 2001.
[21] A. Bultheel, R. Cools (Eds.): *The birth of numerical analysis*, World Scientific, 2010.
[22] E. Zelder: *Applied functional analysis (*AMS108 AMS109), Springer-Verlarg, 1995.
[23] 加藤義夫: **偏微分方程式** [新版], サイエンス社, 2003.
[24] 神保秀一: **偏微分方程式入門**, 共立出版, 2006.
[25] L.C. Evans: *Partial differential equations*, AMS, 1998.
[26] D.Bernoulli (Reviewed by S. Blower): *An attempt at a new analysis of the mortality caused by small pox and the adbantage of inoculation to prevent it.* Rev. Med. Viol.. 14:275-88. 2004.
[27] ラプラス; 内井惣七訳: **確率の哲学的試論**, 岩波文庫, 1997.
[28] H. Yasuda, F. Ito, K. Hanaki, K. Suzuki: COVID-19 pandemic vaccination strategies of early 2021 based on behavioral differences between residents of Tokyo and Osaka, Japan. Archives of Public Health, (2022)80:180. https://doi.org/10.1186/s1390-022-00933-z.
[29] J. Koopman: Modeling infection transmission Annual Review of public health 25: 303-26. 2004.
[30] 吉倉廣: **感染症シミュレーションの考え方**, 生体防御医学事典, 朝倉書店, 2007.
[31] E.T. Lofgren, N.H. Fellerman: The untapped potential of virtual game worlds to shed light on real world epidemics. The Lancet Infectious Diseases. 7:625-9. 2007.
[32] C.T. Bush, D.J.M. Earn: *Vaccination and theory of games.* PNAS. 101:13391-4. 2004.
[33] A. C. Hindmarsh: ODEPACK, A Systematized Collection of ODE Solvers. in Scientific Computing, R. S. Stepleman et al. eds., North-Holland, 55-64,1983.
[34] Paul Graham: **ハッカーと画家**, オーム社, 2005.
[35] 小飼弾: **コードなエッセイ**, 技術評論社, 2013.
[36] J.J. サスマン, H. エイブルソン, J. サスマン: **計算機プログラムの構造と解釈 第２版**, ピアソン, 1996.
[37] A.A. Stepanov, D.E. Rose: **その数式、プログラムできますか？**, 翔泳社, 2015.

[38] L.N. Trefethen: *An applied mathematician's apology,* SIAM, 2022.
[39] L.S. ポントリャーギン: **最適制御理論における最大値原理 付：小自伝**, 森北出版, 2000.
[40] D.P. Wilson: *Mathematics is applied by everyone except applied mathematicians.* Applied mathematics letters. 22:636-7, 2009.
[41] G.H. ハーディ, C.P. スノー: **ある数学者の生涯と弁明**, シュプリンガー・フェアラーク東京, 1994.
[42] L.N. Trefethen: *G.H. Hardy, Applied Mathematician,*
https : //people.maths.ox.ac.uk/trefethen/publication/PDF/2008_129.pdf. 2023/8/4 アクセス.
[43] ハンナ・アレント; 清水速雄訳: **人間の条件**, ちくま学芸文庫, 1994.(see Chap.6)
[44] メイナード スミス: **生物学のすすめ**, ちくま学芸文庫, 2016 .