データ同化

はじめに

私が思うに、CAE 技術とは「対象の物理システムを計算機という別の物理システム上で表現する道具」である。計算機上での表現を正しいものにするために、高品質計算格子生成手法や高次差分技術など、いわゆる高精度 CAE 技術が研究されてきた。長年の技術発展の末、いまでは CAE 解析結果を信頼できるものと見なしている。更に、異なる分野にあった CAD 技術と融合したことにより、既に製品開発での CAE 活用も当たり前になった。

一方で CAE の精度が向上するにつれ、CAE 以外の問題が顕在化し始めた。例えば FEM でリビングルームの音響解析を行う場合に B.C. の情報が必要になるが、それらを正しく測定することは難しい (とくにソファーやカーテンの吸音特性)。B.C. 以外にも、I.C. や物性値に誤りがあることは十分考えられる。CAE 側で設定したこれらパラメータと実際の値に差異がある場合、正しくない CAE 結果となることは当然であろう。これまで実現象に近づくよう発展してきた CAE だが、この問題はもはや CAE 側の努力だけでは解決できない。

実現象と CAE 結果のずれをなくす戦略のひとつとして、データ同化の利用が挙げられる。データ同化の解釈は驚くほど人それぞれだが、本資料では「CAE のためのベイズ機械学習」とする。データ同化は、B.C. など不確かなパラメータを学習対象とみなし、実空間から得た観測値 (つまり教師データ) から学習する。これは線形回帰における、「$y = ax + b$」のうち a と b を学習対象とし、教師データ $\{(x_i, y_i)|i = 1, ..., N\}$ からパラメータを決定するアルゴリズムと同様である。

それゆえ、データ同化にとって CAE は $y = ax + b$ のような数理モデル、いわばひな形にすぎない (表 1)。データ同化の本質は「学習」にあり、その部分はベイズ機械学習が支えている。データ同化を勉強したいならば、たとえ CAE エンジニアであっても AI 技術に興味を持たなければならない。

本資料の目的はデータ同化の解説である。基礎から学べるように、確率論の概要から説明することにした。データ同化の和書で確率論から説明を開始している本はない。一方で確率論とベイズ機械学習を説明している書籍は多数存在する。それらの書籍よりも素晴らしい説明をする自信はないが、ベイズ機械学習の細かな内容を大胆に省くことで差異化を図った。ベイズ機械学習の適用先は広く、既存の本はそれら全てを説明しがちに思える。回り道と思える箇所は回避し、息切れすることなくデータ同化を習得できるよう心がけた。

(文責：TL25 中西　佑児)

1　確率論

実現象と CAE にずれがあり、その原因が CAE 側の誤ったパラメータ設定にあるならば、この問題を「実現象に近づくことを是としたパラメータの最尤値探査」と考えないのはなぜか。最尤値探査は勾配降下法などで達成できるので、本章の確率論は不要に思える。

この指摘は案外的を得ているが、「実現象がいつも正しい値を提供する」ことを前提としている点で惜しい。表 1 中の観測値はセンサなどから得られるため、ノイズの存在は認めるべきである。不確かさのある結果を対象にした最尤値探査である以上、やはり確率論は必要になってくる。確率論と最尤値探査を組み合わせてようやく機械学習らしくなる。何より最尤値以外も重要なことだってあるだろう。観測結果をある値に言い切れない以上、推定対象のパラメータも言い切れないはずである (この考えはベイズ統計特有のもので、頻度主義の統計学では見られない)。「どの程度言い切れないか」の定量的表現は工学的にもビジネス的にも価値がある。

1.1　確率変数と確率分布

1.1.1　確率変数

データ同化ではセンサノイズのような変動する数値 (つまり変数) を扱う。変数は離散的であっても連続的であってもよいが、データ同化では専ら連続変数を扱う。しかしながら、離散変数の観点も非常に重要であるため、本資料では両方を取り扱うことにする。

例えばサイコロの出る目を変数 X で表したとき、X の取り得る値は $\{1, 2, 3, 4, 5, 6\}$ のいずれかである。一般的な解析学の場合、変数の定義域を知るだけで十分に価値があった。しかしながら統計学の場合はそうもいかない。取り得る各値が起こる確率も興味の対象になってくる。このように、確率情報を有する変数のことを確率変数と言う。

確率変数

変数が取り得る各値 (実現値) に対して確率が与えられているとき、その変数を特別に確率変数と言う。確率変数の実現値は離散的であっても連続的であってもよい。前者を離散確率変数、後者を連続確率変数と言う。

統計学の慣習では、確率変数を大文字 $(X, Y, Z, ...)$、実現値を小文字 $(x, y, z, ...)$ で表記する。ただしデータ同化では統計

表 1　線形回帰とデータ同化の類似性

	モデル	教師データ	学習対象
線形回帰	$y = ax + b$	$\{x_i, y_i\}$	a, b
データ同化	CAE	観測値	B.C., I.C., 物性値など

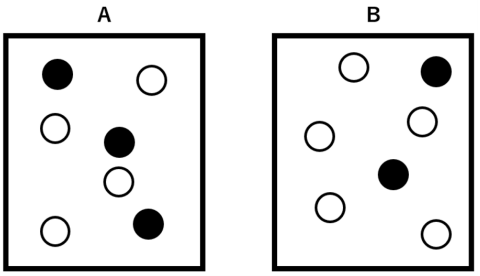

図 1　くじ引きの例

学よりも工学の慣習に従うことの方が多い。例えば質量値が不確かな問題を扱う場合、その確率変数を M と無理に表すのではなく m の方を採用する。

確率変数の例

サイコロ

確率変数に慣れるためにサイコロを例にして考えよう。前述の通り、サイコロの出る目に関する確率変数 X は $\{1,2,3,4,5,6\}$ を取り得る。この実現値の全体集合のことを標本空間と言い、Ω で書き表すことが多い。また、$X=x$ となるときの確率を $\Pr(X=x)$ と書く。理想的なサイコロであれば任意の x に対して $\Pr(X=x)=1/6$ となる。

X が Ω の部分集合 W のいずれかとなる確率を、$\Pr(X\in W)$ と書いてもよい。たとえば X が奇数となる確率は $\Pr(X\in\{1,3,5\})=1/2$ である。当然ながら、$\Pr(X\in\Omega)=1$ である。

ステーキのグラム数

レストランで提供されるステーキのグラム数も表記から多少はずれているだろう。グラム数に関する確率変数 G は連続確率変数に分類される。連続という違いはあれど内容はサイコロの例と変わらない。ステーキがいつも 1000 g のブロック肉からカットして提供している場合、G の標本空間は $\Omega=\{g|g\in[0,1000]\}$ になる。$G=g$ となるときの確率は $\Pr(G=g)$ と表記され、$G\in(a,b)$ となる確率は $\Pr(G\in(a,b))$ で表される。当然ながら $\Pr(G\in\Omega)=1$ でなければならない。

くじ引き

2 つの箱 A と B があり、それぞれに白玉と黒玉が入っているとする (図 1)。ここで玉を 1 つ取り出すゲームを考えよう。プレイヤーは箱 A と B をランダムに選択し、各箱の中の玉もランダムに取り出すとする。

箱 A から白玉を取り出す確率を表現するには、箱に関する確率変数 $X\in\{A,B\}$ と玉の色に関する確率変数 $Y\in\{W,B\}$ が必要になる (同じ色の玉は区別できないと仮定した)。X と Y を用いることで、この確率は $\Pr(X=A,Y=W)$ と表現できるようになる。

このように複数の確率変数を用いることは問題にならない。$\Pr(X=A,Y=W)$ のことを「$(X=A$ かつ $Y=W$ の同時確率」と言う。確率変数の数が多すぎて列挙するのが難しい場合、$\boldsymbol{Z}=(X,Y)^{\mathrm{T}}$ のように確率変数をベクトルで纏め、$\Pr(\boldsymbol{Z}=\boldsymbol{z})$ のように表記してもよい。このような確率変数を多次元確率変数と言う。X と Y の標本空間から、$\boldsymbol{Z}$ の標本空間は $\{(A,W),(B,W),(A,B),(B,B)\}$ であると分かる。

アーチェリー

座標 (0, 0) に的の中心があるアーチェリーの問題を考える。選手はよく訓練されており、横方向にも縦方向にも 1 cm 以上軸から外さない。この場合、横軸方向のずれに関する確率変数 X と縦軸方向の確率変数 Y を考えることができる。また、X と Y の標本空間はどちらも $[-1,1]$ となる。

ランダムウォーク

時系列のある問題でも確率変数はよく現れる。例えばランダムウォークの問題を考えよう。各時刻の座標を X_t(連続値) とし、時刻 T まで移動を行ったとする。このとき、時系列 $\{x_1,x_2,...,x_T\}$ が得られる確率は $\Pr(X_1=x_1,X_2=x_2,...,X_T=x_T)$ のように表記できる。

1.1.2　確率分布

確率分布とは確率変数と確率の関係を関数で表現したものである。離散と連続のどちらに対しても使うことのできる言葉だが、その厳密な意味は両者で大きく異なる。形式張った議論では離散確率変数に対する確率分布を「確率質量関数」、連続確率変数に対する確率分布を「確率密度関数」と区別する。

離散確率変数に対する確率分布

確率質量関数

離散確率変数 X を考える。$X = x$ となる確率 $\Pr(X = x)$ が $p(X = x)$ で表されるとき、関数 p を X の確率質量関数、もしくは単に確率分布と呼ぶ。

なお、$p(X = x)$ という書き方は厳密であるが、$p(X)$ と省略して書く人も多い。このときは確率変数 X が実現値 X であるときの確率と解釈する (確率変数と実現値に対して同じ表記を使うことに違和感を感じるかもしれないが、次第になれる)。

確率の公理 (つまり確率は非負で総和は 1) より、確率質量関数 p は任意の実現値に対して

$$0 \leq p(X) \leq 1 \tag{1}$$

$$\sum_{\Omega} p(X) = 1 \tag{2}$$

であることが言える。

確率質量関数の例

サイコロ

理想的なサイコロであれば、確率質量関数は、任意の $x \in \{1, 2, 3, 4, 5, 6\}$ に対して $p(X = x) = 1/6$ が成立する。

くじ引き (1.1.1 節)

1.1.1 節のくじ引きに関して、箱の選択確率をそれぞれ $1/2$ とし、玉も平等に選択されるとする。このとき、図 1 から明らかに

$$p(X = A, Y = W) = \frac{2}{7}$$

$$p(X = A, Y = B) = \frac{3}{14}$$

$$p(X = B, Y = W) = \frac{5}{14}$$

$$p(X = B, Y = B) = \frac{1}{7}$$

だと分かる。

連続確率変数に対する確率分布

確率密度関数

連続確率変数 X を考える。X の値が $X \in (x, x + \delta x)$ を満たすときの確率が

$$\int_{x}^{x+\delta x} p(X) dX \tag{3}$$

で表されるとき、関数 p を X の確率密度関数と呼ぶ。

このように連続確率変数の場合、確率値の導出に積分計算が必要になる。なお、X の値がちょうど x となる確率は

$$\Pr(X = x) = \lim_{\delta x \to 0} p(x) \delta x$$

で定義される。確率の公理より確率密度関数は

$$0 \leq p(X) \tag{4}$$

$$\int_{\Omega} p(X) \delta X = 1 \tag{5}$$

を満たさなければならない ($p(X) \leq 1$ を満たす必要がないことに注意)。

確率密度関数の例

1.1.1 節のステーキグラム数を例にして考える。200 g のステーキの注文をしたときの実際のグラム数に関して、確率密度関数 $p(G)$ は正規分布 (ガウス分布)

$$p(G) = \frac{1}{\sqrt{18\pi}} \exp\left(-\frac{(G-200)^2}{18}\right)$$

に従うとする。1.1.1 節では標本空間を [0,1000] に限定したにも関わらず、正規分布は任意の実数に対して定義されている。これは、$\int_\Omega p(G)dG = \int_0^{1000} p(G)dG = 1$ を満たす確率分布を探すのが面倒だったためである。便宜的に G の標本空間を実数全体にし、正規分布を採用した。標本空間 [0,1000] 外の確率密度関数値が十分にゼロに近いのであれば、このような操作はあまり問題にならない。

G が上記の正規分布に従う場合、ステーキのグラム数が 197 から 203 になる確率は

$$p(G) = \frac{1}{\sqrt{18\pi}} \int_{197}^{203} \exp\left(-\frac{(G-200)^2}{18}\right) dG$$

となる。この積分計算をする代わりに $Z = (G-200)/3$ で変数変換し、

$$p(Z) = \frac{1}{\sqrt{18\pi}} \int_{-1}^{1} \exp\left(-\frac{Z^2}{2}\right) dZ = \frac{1}{\sqrt{18\pi}} \int_{-\infty}^{1} \exp\left(-\frac{Z^2}{2}\right) dZ - \frac{1}{\sqrt{18\pi}} \int_{-\infty}^{-1} \exp\left(-\frac{Z^2}{2}\right) dZ$$

の積分計算を考える (上式の最右辺は関数の対称性より導いた)。実は $(2\pi)^{1/2} \int_{-\infty}^{a} \exp(-0.5Z^2)dZ$ の積分 (a は任意の実数) は統計学において非常に有名である。統計学の書籍の巻末にテーブルデータとして必ずある程で、それを引用して上式を求めることができる。私も計算を続けるのは面倒なので、ここで切り上げさせて頂く (ちなみに答えは 0.683)。

1.2 確率論の基礎

1.2.1 加法定理

改めて 1.1.2 節のくじ引きの例を考える。この問題は 2 つの確率変数 (X と Y) を扱っているが、例えば、「どちらの箱が選ばれたかは問わず、とにかく白玉が選択された確率」、つまり $p(Y = W)$ を知りたいこともある。この問は高校数学でもよく見かけた。当然ながら

$$p(Y = W) = p(X = A, Y = W) + p(X = B, Y = W)$$

より求まる。この解き方は確率の加法定理を利用している。

加法定理

離散確率変数 X と Y を考える。確率質量関数 $p(X,Y)$ に対して、$p(Y)$ は

$$p(Y) = \sum_x p(X = x, Y) \tag{6}$$

で求めることができる。

加法定理は連続確率変数に対しても成立する。確率密度関数 $p(X,Y)$ に対し、$p(Y)$ は

$$p(Y) = \int p(X,Y)dX \tag{7}$$

で求められる。

連続であれ離散であれ、$p(X,Y)$ から求められた $p(Y)$ のことを周辺確率と言う。

周辺確率の例

くじ引き (1.1.2 節)

箱の選択を問わず、ただ黒玉が選択される確率を計算したい場合、加法定理より

$$p(Y = B) = p(X = A, Y = B) + p(X = B, Y = B) = \frac{3}{14} + \frac{1}{7} = \frac{5}{14}$$

と求まる。

1.2.2 乗法定理

再度 1.1.2 節のくじ引きの例を考える。黒玉を引く可能性は先ほどの例より 5/14 であった。しかしながら、前もって箱 A を選択すると分かっている場合、黒玉を引く確率は図 1 より 3/7 になる。

このように、事前情報が加わると不確かさも変わる。新しく求めた確率 3/7 は $\Pr(X=A, Y=B)$ と異なる。後者を高等数学流に言うなら、「$X=A$ かつ $Y=B$ の確率」に相当する。他方事前情報の加わった確率は「$X=A$ のとき $Y=B$ である確率」と表現できる。後者の確率を条件付確率と言い、$\Pr(Y=B|X=A)$ と書き表す (| のことをギブンと呼ぶ)。これまで同様、条件付確率を $\Pr(Y|X)$ と略記してもよい。

$\Pr(Y=B|X=A)$ は $X=A$ が起こることを前提にしている。したがって、以下の乗法定理が成り立つ。

乗法定理

確率変数 X と Y を考える。このとき、

$$\Pr(X,Y) = \Pr(X|Y)\Pr(Y) \tag{8}$$

が成立する。

もちろん、どちらを前提に持ってきてもよいので、式 (8) 同様 $\Pr(X,Y) = \Pr(Y|X)\Pr(X)$ も成立する。これと式 (8) より、ベイズの定理を導くことができる。

ベイズの定理

乗法定理より

$$\Pr(X|Y) = \frac{\Pr(Y|X)\Pr(X)}{\Pr(Y)} \tag{9}$$

が成立する。

左辺にとって Y の値は前提となっている。つまり Y はすでに決まっていると考えることができる。すると右辺の $\Pr(Y)$ はもはや定数と考えてもよい。いわば確率分布 $\Pr(Y|X)\Pr(X)$ が確率の公理 (総和が 1) を満たすための規格化係数にすぎないので、ベイズの定理は

$$\Pr(X|Y) \propto \Pr(Y|X)\Pr(X) \tag{10}$$

と書かれることもある。

ベイズの定理に関して、両辺で前提となっているものが入れ替わっていることに注目してほしい。人によっては $\Pr(X|Y)$ のうち Y を原因、X を結果と呼ぶ。したがってベイズの定理は因果を反転させる式として便利である。

ベイズの定理の例

おなじみのくじ引きの例を考える (1.1.2 節)。1.1.2 節の例より、箱 A が選択される確率は 1/2 であった。ここでは「白玉が出たとき、それが箱 A からのものである確率」$\Pr(X=A|Y=W)$ を考える。$\Pr(Y=W|X=A)$ は図 1 より明らかだが、$\Pr(X=A|Y=W)$ の計算は意外と難しい。そこでベイズの定理が登場する。

まず、$p(X=A)$ は 1/2、$p(Y=W|X=A)$ は 4/7 なので、式 (9) より

$$p(X=A|Y=W) = \frac{p(Y=W|X=A)p(X=A)}{C} = \frac{2}{7C}$$

となる。ここで C は $p(Y=W)$ だが、規格化係数であることを強調するため C という書き方にした。C の計算のために直接 $p(Y=W)$ を求める人もいるが、その代わりに

$$p(X=B|Y=W) = \frac{p(Y=W|X=B)p(X=B)}{C} = \frac{5}{14C}$$

を求め、確率の公理 $p(X=B|Y=W)+p(X=A|Y=W)=1$ から規格化定数 C を求めることもできる。$p(Y)$ よりも $p(X)$ や $p(X|Y)$ の計算の方が楽な場合は、この方法を用いるべきだろう。最終的に、本問題の解は 4/9 だと求まる。

条件確率に関して $p(X|Y)=p(X)$ が成立するとき、X と Y は独立であるという。この結果は Y の結果が X に影響を及ぼさないことを示している。たとえば 2 個のサイコロを投げる問題を考えよう。それぞれのサイコロを区別して考えるとき、確率変数は 2 つ必要になる (X と Y とする)。当然ながら X の実現値は Y の結果に影響を及ぼさない。確率変数に独立の関係があると同時確率が

$$p(X,Y) = p(X|Y)p(Y) = p(X)p(Y)$$

のように、各変数を単独に考えたときの確率の積になり、計算がシンプルになってくれる。

これと似た概念にマルコフ性というものがある。時系列に関する確率変数 $(X_1, X_2, ..., X_T)$ において、

$$p(X_t|X_{t-1}, X_{t-2}, ..., X_1) = p(X_t|X_{t-1}) \tag{11}$$

が成立するとき、確率変数はマルコフ性を有すると言う。これは、X_t に関する条件付確率が前時刻の実現値にのみ依存することを意味している。

1.3 確率分布の代表値

代表値とは確率分布の特性を表す指標である。本資料では基礎中の基礎とも言える 3 つの代表値を紹介する。

1.3.1 期待値

「確率変数が取るであろうと期待される値」のことを期待値という。一般的に確率変数 X の期待値を $E[X]$ と表記し、確率分布による重み付き和で定義する。つまり、離散確率変数の場合は

$$E[X] = \sum_{X \in \Omega} Xp(X) \tag{12}$$

であり、他方連続確率変数の場合は

$$E[X] = \int_{\Omega} Xp(X)dX \tag{13}$$

となる。また、$E[\cdot]$ という表記は X を任意の関数で変換させた確率変数 $f(X)$ に対しても使える。この $E[f(X)]$ は離散と連続のそれぞれで

$$E[X] = \sum_{X \in \Omega} f(X)p(X) \tag{14}$$

$$E[X] = \int_{\Omega} f(X)p(X)dX \tag{15}$$

となる。X を $aX + b$ に変換させたとき (a, b はスカラー)、$E[aX + b]$ は上式より

$$E[aX + b] = aE[X] + b \tag{16}$$

だとわかる。

1.3.2 分散

確率分布の広がりを表す代表値に分散がある。一般的に確率変数 X の分散を $V[X]$ と書く。$V[X]$ は期待値の表記 $E[\cdot]$ を用いて、

$$V[X] = E\left[(X - E[X])^2\right] \tag{17}$$

と定義されている。式 (17) より、分散値は非負だと分かる。また、確率変数が一つの値のみ取り得る場合に限って、分散はゼロになる。

確率変数を $aX + b$ に線形変換したとき、分散は

$$V[aX + b] = a^2V[X] \tag{18}$$

と求まる。

1.3.3 共分散

多次元確率変数 $\boldsymbol{X} = (X_1, X_2, ..., X_n)^{\mathrm{T}}$ における、任意の要素 X_i と X_j に対して、共分散

$$\mathrm{Cov}[X_i, X_j] = E\left[(X_i - E[X_i])(X_j - E[X_j])\right] \tag{19}$$

が定義されている。共分散は確率変数間の相関を表している。共分散が正 (負) のとき、確率変数間には正 (負) の相関がある。また、共分散がゼロのとき確率変数同士は無相関である。同様に独立な確率変数同士の共分散はゼロになる。なお、$i = j$ のとき、共分散は分散と同じになる。

i 行 j 列に $\mathrm{Cov}[X_i, X_j]$ が入る n 次正方行列のことを分散共分散行列という。式 (19) より、分散共分散行列は対象行列であることが分かる。また、任意の確率変数同士が独立である場合、分散共分散行列は対角行列になる。

1.4 確率分布の表現方法

ここまで「確率分布は関数である」と紹介してきたが、そもそもの関数についてあまり深く考えてこなかったと思う。数学において関数は以下のように定義されている。つまり、集合 X の元 x のそれぞれに、ある規則にしたがって集合 Y の元 y を一つずつ対応させるとき、この対応規則を関数と言い、$f : x \to y = f(x)$ のように書く。

単なる対応規則であると考えることができれば、本節の理解は捗るだろう。本節では、離散確率変数と連続確率変数に分けて、パラメトリック確率分布とノンパラメトリック確率分布の概念を紹介する。

表 2　ノンパラメトリック確率分布 (サイコロの場合)

x	1	2	3	4	5	6
p	1/6	1/6	1/6	1/6	1/6	1/6

1.4.1　離散確率分布の場合

離散確率分布の表し方は大きく分けて 2 種類ある。いま、n 種 (有限) の実現値 $\{x_i|i=1,...,n\}$ を取り得る確率変数 X を考えよう。最も単純な確率分布の表現方法は、各 x_i に対応する確率値 p_i を用意し、テーブルデータとして保存する方法であろう。例えばサイコロの場合、出る目とその確率値は表 2 のように表現することができる。高等教育で学んだ確率は確か離散変数で、私たちも表 2 のようなテーブルデータを思い浮かべながら問題を解いていたはずである。テーブルデータによる表現は非常にシンプルで、計算機にとって扱いやすい。しかしながら確率変数の取り得る値 n が大きくなると、当然それに要するメモリも多くなる。実現値と対応する確率値を合わせて保存しなければならないため、個数 n に対してテーブルデータの数値の個数は $2n$ になる。場合によってはメモリの制約上、テーブルデータによる表現方法を避けなければならない。

もう一つの確率分布の表現方法はパラメトリック確率分布と呼ばれている。これは、表 2 のようなデータを関数でフィッティングする方法と思えばよい。たとえば単変数かつ離散確率変数において有名なパラメトリック確率分布にポアソン分布

$$p(X) = \frac{e^{-\lambda}\lambda^X}{X!}$$

がある (ポアソン分布の X の取り得る値は非負の整数のみ)。ポアソン分布はパラメータ λ を持ち、対象の確率変数に対してフィッティングしなければならない (もちろん理論的観点からパラメータの適切な値が求まり、フィッティングを必要としない場合もある)。フィッティングである以上誤差があるかもしれないが、上手く近似できた場合、確率分布情報の保存はパラメータ分だけのメモリで済む。ポアソン分布の場合 λ の数値のみ保存すればよいことになるので、n の数によっては大幅なメモリ節約になる。

パラメトリック確率分布に対して、テーブルデータによる表現方法をノンパラメトリック確率分布と言う。個人的にはテーブルデータによる表現が好きだが、パラメトリック確率分布と合わせるために

$$p(X) = \sum p_i \delta(X - x_i) \tag{20}$$

のように書くことも多い (確率の公理より $\sum p_i = 1$)。このような表現ゆえに、ノンパラメトリック確率分布は初学者にとって難しいとされている。しかしながら手計算とはいえ高等教育で扱っていたことを思い出すと、その認識は大げさな気もする。また、数値計算との相性も悪くないので、あまりノンパラメトリック確率分布を毛嫌いすべきでないようにも思える。

さて、繰り返しになるがパラメトリック確率分布ではパラメータの調整、つまり学習が必要になる。ポアソン分布の場合の学習対象は λ のみなので、その表現力の乏しさが想像できる (一方のノンパラメトリック確率分布は表現力の点で優れている)。ポアソン分布以外にも数多くのパラメトリック確率分布が提案されているが (本資料では紹介しない)、いずれもパラメータ数は少ない。近年の研究では、これら分布の代わりに深層学習を用いることの方が多い (とはいえデータサイエンティストは今でも伝統的な確率分布を上手く利用している)。

1.4.2　連続確率分布の場合

連続確率変数でもパラメトリック確率分布とノンパラメトリック確率分布がある。パラメトリック確率分布の代表例に正規分布

$$p(X) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right)$$

がある (正規分布のパラメータは μ と σ^2 のみ)。パラメトリック確率分布ゆえにフィッティングさえ上手くいけば、確率分布の情報はパラメータ値に集約される (つまりメモリへの負担が少ない)。

連続確率変数におけるノンパラメトリック確率分布の表現方法は主に 2 通りある。

確率変数空間の離散化

本来なら、連続確率変数の取り得る実現値は無数に存在するため、離散確率変数におけるテーブルデータ (表 2) のような表現方法はできない。そこで、連続確率変数の場合は解析者の手で実現値を離散化する。つまり、無数に取り得た実現値 $\{x|x \in \Omega\}$ を有限個の実現値 $\{x_i|i=1,...,n\}$ で近似してしまう。さらに、各実現値 x_i に対して確率密度関数値 $p(x_i) = p_i$ を定義することで、連続確率変数におけるノンパラメトリック確率分布の表現を得る。これは CAE における離散化の考えと非常に似ている (CAE も時空間を離散化し、連続体力学の物理場を有限次元の状態ベクトルで表現してしまう)。離散確率変数におけるノンパラメトリック確率分布は決して近似的表現方法でない一方、連続確率変数におけるそれは近似的である。しかしながら、CAE との類似性から分かるように、離散数 n を増やすほど近似による誤差は小さくなっていく。

図 2 に同じ確率分布に対するパラメトリック表現とノンパラメトリック表現の対比を示す。図 2(b) は表 2 のようにテーブルデータで表現することもできるし、式 (20) 同様、

$$p(X) \simeq \sum p_i \delta(X - x_i) \tag{21}$$

のように書き表すこともできる (確率の公理より $\sum p_i = 1$)。ノンパラメトリック確率分布を上式で定義した場合、確率分布の代表値計算は離散確率変数のときのようになる。例えば期待値は式 (13) から、

$$\int_\Omega X p(X) dX = \int_\Omega X \sum p_i \delta(X - x_i) dX = \sum x_i p_i$$

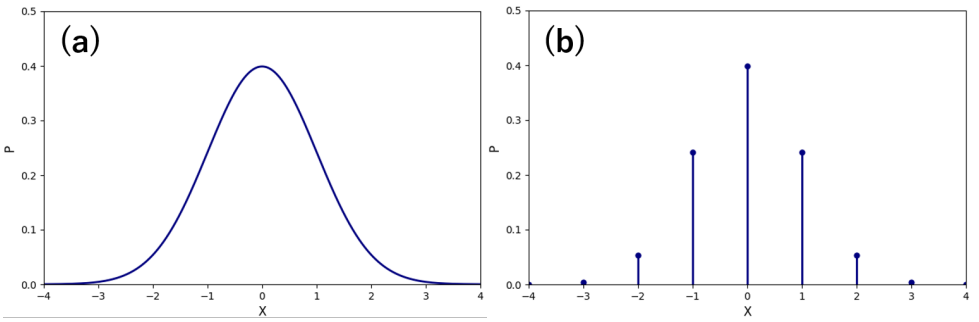

図 2　(a) 連続確率分布のパラメトリック表現 (b) 離散化によるノンパラメトリック表現

となり、式 (12) と同様になる。

アンサンブルによる表現

所望の確率分布 $p(X)$ に従ってサンプリングし、$\{x_i|i=1,...,N\}$ を得たとする。このサンプルデータに対して、重複するものを一つに纏め、新たにデータセット $\{x'_i|i=1,...,n\}(n \leq N)$ を作る。また、$\{x_i|i=1,...,N\}$ の中に x'_i は n_i 個あったとしよう (従って $\sum_i n_i = N$)。すると、x'_i がサンプリングされる確率は n_i/N だと分かるので (頻度主義の確率。頻度主義は 2 章で説明する)、式 (21) より、$p(X)$ を

$$p(X) \simeq \sum_i \frac{n_i}{N}\delta(X - x'_i)$$

と近似できる。ただし、上式は重複を許す形で、

$$p(X) \simeq \sum_i \frac{n_i}{N}\delta(X - x'_i) = \frac{1}{N}\sum_i \delta(X - x_i) \tag{22}$$

と書き直すこともできる (一般的には式 (22) の表記が使われる)。

このように、サンプリング結果の重複度合いで確率密度関数値の大きさを表現する。アンサンブルによる表現の利点はサンプリングの容易さにある。式 (22) の最右辺で表された確率分布に対して、その分布に従うようサンプリングするには、データセット $\{x_i|i=1,...,N\}$ からそれぞれ $1/N$ の確率で選択すればよいことに気付く。頻度主義的に、$p(X)$ の精度はサンプル数とともに増加する。

1.5　正規分布

データ同化で扱う確率変数は専ら連続で、パラメトリック確率分布に正規分布を採用することが多い。本節では正規分布とその特徴を紹介する。

1.5.1　単変数の正規分布

単変数の正規分布は

$$p(X) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right) \tag{23}$$

で表される。正規分布のパラメータは μ と σ^2 で、それぞれ確率分布の期待値および分散と一致する。正規分布は $\mathcal{N}(X|\mu,\sigma^2)$、もしくは単純に $\mathcal{N}(\mu,\sigma^2)$ と表記されることが多い。統計学において、期待値ゼロかつ分散 1 の正規分布 $\mathcal{N}(0,1)$ は非常に重要であり、特別に標準正規分布と呼ばれている。

正規分布の線形変換

確率変数 X は正規分布 $\mathcal{N}(\mu,\sigma^2)$ に従うとする。スカラー a,b で線形変換された確率変数 $aX+b$ も正規分布 $\mathcal{N}(a\mu+b, a^2\sigma^2)$ に従う。

上記より、任意の正規分布 $\mathcal{N}(\mu,\sigma^2)$ は標準正規分布の線形変換

$$X = \sigma Z + \mu$$

より得られる (Z は標準正規分布に従う確率変数)。

正規分布に従う確率変数 X に非線形変換をしたとき、変換後の確率変数は正規分布に従わないと認識している。ただし、「如何なる非線形変換も正規分布を維持できない」と言い切れるのかどうかを、私は知らない。

正規分布の和

$\mathcal{N}(\mu_1, \sigma_1^2)$ に従う確率変数 X と $\mathcal{N}(\mu_2, \sigma_2^2)$ に従う確率変数 Y を考える。このとき、$X+Y$ も正規分布 $\mathcal{N}(\mu_1+\mu_2, \sigma_1^2+\sigma_2^2)$ に従う。

正規分布の積

$\mathcal{N}(\mu_1, \sigma_1^2)$ に従う確率変数 X と $\mathcal{N}(\mu_2, \sigma_2^2)$ に従う確率変数 Y を考える。このとき、XY も正規分布 $\mathcal{N}(\mu, \sigma^2)$ に従う。ただし、

$$\mu = \frac{\sigma_2^2\mu_1 + \sigma_1^2\mu_2}{\sigma_1^2+\sigma_2^2}$$

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}$$

である。

1.5.2 多変数の正規分布

多変数の正規分布は

$$\mathcal{N}(\boldsymbol{X}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{X}-\boldsymbol{\mu})\right) \tag{24}$$

で表される。正規分布のパラメータは $\boldsymbol{\mu}$ と Σ で、それぞれ確率分布の期待値と分散共分散行列と一致する。また、D は確率変数の次元で、演算子 $|\cdot|$ は行列式である。確率分布 $\mathcal{N}(\mathbf{0}, I)$ は特別に標準正規分布と呼ばれている。

正規分布の線形変換

確率変数 $\boldsymbol{X}$ は正規分布 $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ に従うとする。行列 A とベクトル $\boldsymbol{b}$ を用いて、$A\boldsymbol{X}+\boldsymbol{b}$ なる線形変換を施したとき、

$$p(A\boldsymbol{X}+\boldsymbol{b}) = \mathcal{N}(A\boldsymbol{\mu}+\boldsymbol{b}, A\Sigma A^{\mathrm{T}})$$

が成立する。

正規分布の和

多次元正規分布 $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ に従う確率変数 $\boldsymbol{X}$ と $\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$ に従う確率変数 $\boldsymbol{Y}$ を考える。このとき、$X+Y$ も正規分布 $\mathcal{N}(\boldsymbol{\mu}_1+\boldsymbol{\mu}_2, \Sigma_1+\Sigma_2)$ に従う。

正規分布の積

多次元正規分布 $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ に従う確率変数 $\boldsymbol{X}$ と $\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$ に従う確率変数 $\boldsymbol{Y}$ を考える。このとき、XY も正規分布 $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ に従う。ただし、

$$\boldsymbol{\mu} = \Sigma_2(\Sigma_1+\Sigma_2)^{-1}\boldsymbol{\mu}_1 + \Sigma_1(\Sigma_1+\Sigma_2)^{-1}\boldsymbol{\mu}_2$$

$$\Sigma = \Sigma_1(\Sigma_1+\Sigma_2)^{-1}\Sigma_2$$

である。

2 ベイズ機械学習

本資料の冒頭で「データ同化の本質はベイズ機械学習にある」と述べた。したがって、本章の内容は非常に重要になってくる。しかしながらベイズ統計の隅々まで本資料で紹介することは不可能なので、線形回帰における機械学習の紹介に留める。ただし、それだけでもデータ同化の学習に支障はない。なぜならデータ同化において利用する数理モデルは基本的にCAE であり、その他の数理モデル (深層学習など) を深く学ぶ必要はないためである。

ここで、本資料をお読みいただいた皆様の立ち位置を確認しよう。皆様は確率論を学んだに過ぎず、まだ統計学に触れていない。それどころか、実は確率の決め方すら学んでいないことにも気付く。例えばサイコロで 1 が出る確率=1/6 に関して、この 1/6 が何を根拠に決められたのか説明できない。

結論から言うと、確率には頻度主義と呼ばれるものと主観確率と呼ばれるものがある。「確率」という 1 つの言葉に 2 つの見方があるのは、少々奇妙かもしれない。しかしながら、確率が「不確かさ」の定量的表現であったことを思い出そう。この 2 つの見方が不確かさに対して指しているのであれば、なんとなく許容されそうである。一般的な統計学及び機械学習は頻度主義の確率を採用している一方、ベイズ統計及びベイズ機械学習は主観確率を採用している。データ同化はベイズを用

いるが、頻度主義を理解することの恩恵は非常に大きい。そこで本章では、頻度主義の観点と対比させながら、ベイズ機械学習を紹介することにした。

2.1 頻度主義の確率と主観確率

多くの人にとってなじみ深いのは頻度主義の確率であろう。たとえばサイコロの目が 3 となる確率を調べるたいとしよう。このとき、実際にサイコロを n 回振って、そのうち n_3 回 3 が出たならば、3 が出る確率は n_3/n だと推測できる。以下は頻度主義の確率の形式張った説明である。

頻度主義の確率

事象 A を生み得る実験を $n(\to\infty)$ 回行い、実際に A が n_A 回起こったとき、$\Pr(A)=n_A/n$ と定義する。同様に確率変数 X に関して $n(\to\infty)$ 回試行したとき、実現値 x が n_x 回起こったならば、$\Pr(X=x)=n_x/n$ と定義する。

試行回数が $n\to\infty$ となっている点に注意してほしい。当然ながら無限回の試行を実際に行うことは不可能なので、私たちは頻度主義の確率を厳密に求めることはできない。たとえ何万回と試行したとしても、その結果から得られる確率は推測値でしかない。

試行回数があまりにも少ないとき、推測結果はあまり信用できない。大げさな例だが、サイコロを $n=3$ 回振って、全て 6 が出たとする。この結果から「このサイコロは必ず 6 を出す」と見なしてしまうことは、当然ながら危険であろう。したがって、頻度主義の確率では十分な試行回数が必要になる (このことから、頻度主義を基盤にした機械学習がビッグデータを要する理由も見えてくる)。

一方で、主観確率のような考え方は案外私たちの生活の中でよく現れる。たとえば「このビジネスの成功確率は?」と聞かれたとき、私たちは「70% です」と答えたりする。その確率は過去の実績 (つまり実際の試行結果) から算出されたものかもしれないが、単に自分の自信度合いから導出されたものかもしれない。主観確率は後者の自信度合い的な考えに近い。

当然ながら自信度合いなんてものには個人差があり、客観性に欠ける (まさに主観)。数値で表現しているとは言え、主観確率はどこか定性的にも見える。ベイズ統計やベイズ機械学習は、この主観確率を存分に活用しているのだから、今の段階では不思議に思われるかもしれない。実際、数十年前までベイズ統計や主観確率は多くの統計学者から嫌われていた。しかし、今では「厳密ではないかもしれないが、役に立つ」と認識されている。

例えば「30 年後の地球の平均気温が、今と比べて 1 ℃高い確率は?」といった問題を考えてみよう。先ほどのビジネスに関する確率では過去の実績から頻度主義の確率を求められたかもしれないが、今回はそう上手くいかない。未来の話ゆえに一度の試行結果も与えられないためである。他方の主観確率は相変わらず「自信度合い」から導出できる (例えば 30%)。この 30% という数値が、エネルギー問題や世界情勢などを鑑みて導出されたものならば、主観とはいえ信頼できるかもしれない。少なくとも全く手を出せなかった頻度主義よりかは断然ましだろう。

このように、主観確率は客観性を捨てた分、有識者のドメイン知識を組み込みやすくしている。そして数学的厳密さを犠牲にすることで、現実の多くの問題において実利を与えてくれている。以下は主観確率の私なりの解釈である。

主観確率

有識者が主観的に決めた確率値。その正しさは有識者の知識に依存する (見当違いな確率値を設定された場合は、悲惨な結果を招く)。

驚くべきことに、ベイズの定理 (式 (9)(10)) は主観確率 (つまり統計学と関係のないドメイン知識) とデータサイエンスを結びつけてくれる (後述)。

2.2 頻度主義による線形回帰

ここからは線形回帰を題材にして頻度主義による機械学習とベイズ機械学習を比較していく。なお、以後は前者を単に機械学習と呼ぶことにする。

まず機械学習の方を考えよう。いま、データセット $D=\{(x_i,y_i)|i=1,...,N\}$ を持っており、y を $ax+b$ で表したいとする。つまり x を入力として y を推測したい訳であるが、機械学習の分野では入力の代わりに説明変数、推測対象のことを目的変数と言う。

さて、データセットは実験やセンシングなどで集められた結果だが、そこにはノイズが混在してしまう。一般的にノイズは目的変数にのみ含まれる。例えば実験に関するデータであれば、説明変数は実験条件で目的変数は実験結果になる。また、センシングデータであれば、説明変数は時刻で目的変数はセンサ値かもしれない。いずれの説明変数も設定しやすく、あまりずれることはない (図 3)。本資料でも誤差やノイズは目的変数に対してのみ考慮する。

機械学習の目的は説明変数と目的変数を繋ぐ数理モデルの学習であり、今回の場合はパラメータ a と b の学習である。皆様はすでに最小二乗法をご存じかもしれないが、本資料では確率論的に議論することにしよう。

まず、次節の表記と合わせるために、$\boldsymbol{x}=(x,1)^{\mathrm{T}}$ と $\boldsymbol{w}=(a,b)^{\mathrm{T}}$ なるベクトルを定義する。すると $ax+b=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$ となるので、線形回帰はベクトル $\boldsymbol{w}$ を推定する問題だと考えられる。ただしいかなる $\boldsymbol{w}$ においても、y と $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$ が全てのデータで一致することはなく、

$$y_i=\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}+\epsilon_i \tag{25}$$

をもって等式が成立する。ここで ϵ_i は y_i と $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}$ の差分だが、機械学習では目的変数の誤差と見なす。ただし、あくまで差分だったことは覚えておいてほしい。例えば $\boldsymbol{x}=\boldsymbol{x}_i$ のときの、真の値を y_t とし、そこに誤差 ξ_i が含まれた結果 $y_i=y_t+\xi_i$

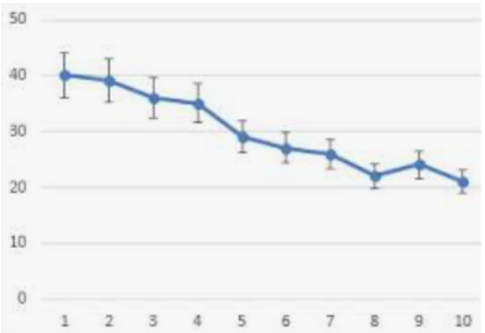

図 3　適当に探してきた図。一般的に横軸 (説明変数) のバラつきは考慮されないことがよく分かる。

が得られたとする。この ξ_i は真の値からのずれなので、センサノイズなどに相当する真の誤差と言える。一方の ϵ_i は式 (25) より、

$$\epsilon_i = (y_t - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}) + \xi_i \tag{26}$$

となる。つまり、ϵ_i は本来の誤差 ξ_i に数理モデルのずれ $(y_t - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w})$ が加わったものと言える。

ξ_i は確率変数 ξ のサンプリング結果であるとしよう。また、ξ の確率分布を $\mathcal{N}(0, \sigma^2)$ とする。ここで σ^2 はセンサノイズに関する分散である。センサであれば σ^2 はカタログ等から推定可能だが、未知であることもある。一方で ϵ_i も確率変数 ϵ の実現値であるとし、その確率分布を $\mathcal{N}(0, \sigma'^2)$ と仮定する。式 (26) より、y_t と $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$ が近い値をとれば、つまり数理モデルが正しければ、ϵ_i と ξ_i の差はなくなる。したがって、学習が上手くいくと考え、ϵ に対して ξ と似た確率分布を仮定することは問題にならない。

目的変数のバラつきを表すために、y に関する確率変数 Y を考える。式 (25) より、確率変数 Y の確率分布 $p(Y|x, \boldsymbol{w})$ は $\mathcal{N}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}, \sigma'^2)$ になる (正規分布の線形変換)。なお、Y の分布は x と $\boldsymbol{w}$ に依存するため条件付確率 $p(Y|x, \boldsymbol{w})$ とした。

ここまでの仮定より、データセット D は確率分布 $\mathcal{N}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}, \sigma'^2)$ からサンプリングされた結果ということになる。各データが独立にサンプリングされたとすると、D が得られる確率 L は

$$L = \prod_i \mathcal{N}(Y = y_i|\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}, \sigma'^2) = \prod_i \frac{1}{\sqrt{2\pi\sigma'^2}} \exp\left(-\frac{(y_i - \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})^2}{2\sigma'^2}\right) \tag{27}$$

となる。この L は D が得られる尤もらしさという意味で、尤度と呼ばれる。ただしこの尤度には未知数 $\boldsymbol{w}$ と σ'^2 が含まれている。したがって、ここからは尤度関数 $L(\boldsymbol{w}, \sigma'^2)$ と呼ぶ。

当然ながら尤度関数の値は $\boldsymbol{w}$ と σ'^2 に依存する。低い尤度を与えるパラメータの場合、D が得られたという結果は稀であることを示す。逆に高い尤度の場合、D なる結果と辻褄が合う。したがって、尤度関数を最大とする $\boldsymbol{w}$ と σ'^2 を是とすると良さそうである。機械学習は詰まる所尤度関数の最大化問題と言える。ただし実際の計算では L の代わりに $-\ln L$ の最小化を考える。式 (27) より $-\ln L$ は

$$-\ln L(\boldsymbol{w}, \sigma'^2) = \frac{N}{2}\ln\sigma'^2 + \frac{N}{2}\ln(2\pi) + \frac{1}{2\sigma'^2}\sum_i (y_i - \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})^2 \tag{28}$$

と求まる。上式が最小値となるとき、$-\ln L$ のパラメータに関する偏微分はゼロになる。そこで式 (28) を見返すと、第三項のみ $\boldsymbol{w}$ に依存することに気付く。そのため、$\boldsymbol{w}$ の最適値探査は、

$$E = \frac{1}{2}\sum_i (y_i - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w})^2 \tag{29}$$

の最小化問題で済むことが分かる。これは正に二乗和誤差で、最小二乗法で利用されている評価関数である。最小二乗法では「y と $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$ のずれを最小化させる」という幾何的解釈で説明されるが、その根底に正規分布という確率的仮定があったことは興味深い。また、式 (29) で考える最小二乗法は暗に σ'^2 の存在を無視していたことになる。場合によってはショックを受けたかもしれないが、実はそれほど罪深くはない。というのも最小二乗法に限らず頻度主義の機械学習の多くはバラつき σ'^2 の存在を無視して議論しており、データと数理モデルが平均的にみて一致することのみ要求する。ちなみに式 (29) に対して σ'^2 の最適値を探査すると、結果は

$$\sigma'^2 = \frac{1}{N}\sum_i (y_i - \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})^2$$

となる。これは線形回帰の結果 $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}$ を平均値に見立てたデータ y_i の分散であり、パラメータ $\boldsymbol{w}$ が推定された今となっては月並みな結果に過ぎない。

しかしながら、上記のような形で推測値のバラつきを評価できることは理解しておくべきであろう。世間では、「バラつき評価はベイズ機械学習の専売特許」と言う人が多いが、それは誤解である。単に、わざわざ頻度主義でバラつき評価をしないだけである。ただ、ベイズ機械学習の方がバラつき評価に向いていることは事実である。例えば図 4 のようにバラつきが x に依存するような場合、話は複雑になってくる。ここまでの議論では、σ^2 や σ'^2 は x に依存しないと考えてきた。依存するとなった場合、議論は初めからやり直しになる。また計算も複雑で、あまりやる気が起きない (私は経験したことがない)。

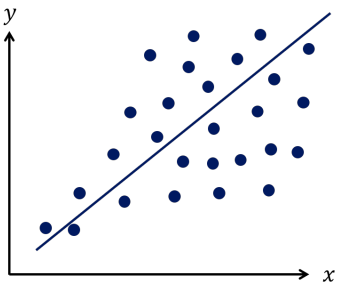

図 4　バラつきが説明変数に依存する例。

一方で、この後すぐ紹介するベイズ機械学習では図 4 のようなバラつきであっても自然に対応可能である。それゆえバラつき評価も必要な場合はベイズ機械学習を利用することが一般的になった。

さて、ご存じの通り機械学習ではビッグデータが好まれる。これはひとえに式 (29)、つまり誤差関数がデータに依存するためである。データ数が少なくなるにつれ、誤差関数に対する各データの寄与は上がる。仮に少数のデータしか集められず、かつそのデータに外れ値が含まれている場合、誤差関数は的外れな指標となってしまい、学習に失敗する。

同様のことを式 (27) からも読み解ける。式 (27) の最大化は、いわばデータセットが作った分布にフィッティングさせる処理である。データセットが作る分布は正に頻度主義的な確率分布であるため、データ数が少ないと信憑性は低い。安心して頻度主義確率を利用できるだけの、ビッグデータが必要になる訳である。

2.3　ベイズ機械学習による線形回帰

2.3.1　ベイズ機械学習の考え方

頻度主義の線形回帰では、学習パラメータ $\boldsymbol{w}$ は確定値として扱われていた。データのばらつきは誤差として認め、$\mathcal{N}(\boldsymbol{x}^{\mathrm{T}}, \sigma'^2)$ に従うと考えた。ただし結局は式 (29) の最小化問題に帰着させるので、分散を意識することは稀である。特別な処理を追加しない限り、機械学習は平均的に y が $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$ で書き表されるということを教えてくれるのみである。

一方のベイズ機械学習では学習パラメータを確率変数と考える。例えば $\boldsymbol{w}$ が $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ に従う 2 次元確率変数である場合、y も確率変数で表され、その確率分布 $p(Y|x)$ は

$$p(Y|x) = \mathcal{N}(\mu_y, \sigma_y^2), \quad \mu_y = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\mu}, \quad \sigma_y^2 = \boldsymbol{x}^{\mathrm{T}}\Sigma\boldsymbol{x}$$

となる (正規分布の線形変換)。誤差項をもって不確かさを表現していた前節の機械学習に対し、ベイズ機械学習は不確かさをパラメータに内包させる (したがって σ'^2 の推定が不要になる)。そして、ある説明変数 x のときの真の値を y_t としたとき、データは $\mathcal{N}(y_t, \sigma^2)$ に従ってばらつく訳であるが、ベイズ機械学習は μ_y を y_t に、σ_y^2 を σ^2 に一致するよう学習する。複雑な考察を行う分、ベイズ機械学習は上式のように y の不確かさも容易に扱える。当然ながらパラメータに関する確率分布は前もって与えられない。ベイズ機械学習の目的は正にパラメータの確率分布の推定である (図 5)。

機械学習はデータセットの分布に一致するようパラメータを推定していた。その点パラメータの確率分布自体を推定することは、特徴的であると言える。しかしながら、私たちは如何にしてパラメータの確率分布を得られるだろうか。手元にあるのはデータセット D のみで、パラメータのサンプルデータなんてものはない (あるはずがない)。頻度主義アプローチで確率を求めることは不可能と言える。

そこで登場するのが主観確率である。ベイズ機械学習ではドメイン知識を駆使して学習パラメータの確率分布 $p(\boldsymbol{w})$ を決めてしまう。しかしながら、これだけだとベイズ機械学習の価値はない。単に他の学問の知識を活用しただけであって、データサイエンス的な要素が全くない。ドメイン知識を活用しつつ、データセットを根拠に推定されたパラメータの確率分布、つまり $p(\boldsymbol{w}|D)$ こそ私たちは知りたい。

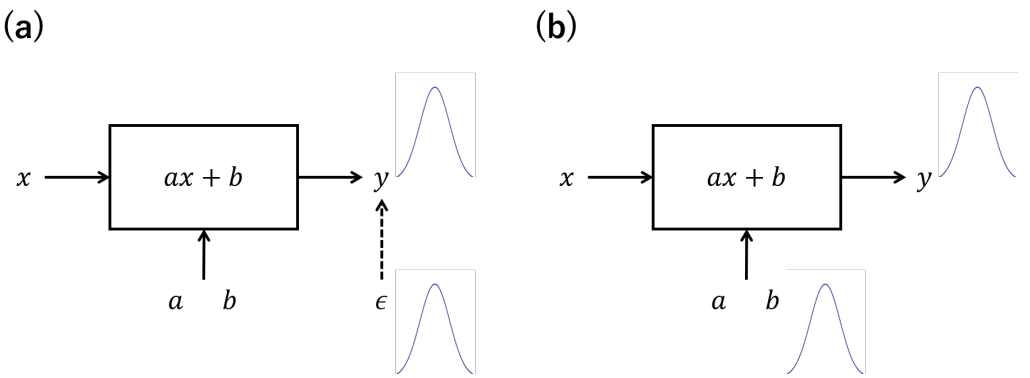

図 5　線形回帰における頻度主義的機械学習とベイズ機械学習の解釈の違い。(a) 頻度主義 (b) ベイズ機械学習。

以上を踏まえてベイズの定理を思い出そう。便宜的に再掲させて頂くと、ベイズの定理は

$$p(\boldsymbol{w}|D) \propto p(D|\boldsymbol{w})p(\boldsymbol{w})$$

のように書き表される。ここで $p(\boldsymbol{w})$ は主観確率、$p(\boldsymbol{w}|D)$ はデータセットを根拠に含む確率であったが、以後はベイズ機械学習の慣習に従った呼び方を使う。まず $p(\boldsymbol{w})$ を統計的プロセスを施す前に与えられる確率という意味で、事前分布と呼ばれている。一方の $p(\boldsymbol{w}|D)$ は事前分布に演算を施した結果という意味で、事後分布と呼ばれている。

残った $p(D|\boldsymbol{w})$ だが、これは以下のように展開できる。

$$p(D|\boldsymbol{w}) = p(\{(x_i, y_i)|i = 1, ..., N\}|\boldsymbol{w}) = \prod_i p(x_i, y_i|\boldsymbol{w}) = \prod_i p(y_i|x_i, \boldsymbol{w})p(x_i) = \prod_i p(y_i|x_i, \boldsymbol{w}).$$

なお、今回説明変数側の不確かさは無視しているので、$p(x_i) = 1$ とした。上式は正にデータセットに対する尤度に相当する (式 (27))。

以上より、条件付確率から機械的に導出されたベイズの定理だが、ベイズ機械学習では事前分布、事後分布並びに尤度で構成されていることが分かった。

ベイズ機械学習の計算手順は以下の通りである。まず、$\boldsymbol{w}$ の事前分布を決め、$\boldsymbol{w}$ の取り得る値全てに対し尤度を計算する。そして $\boldsymbol{w}$ の値毎に事前確率と尤度の積を計算すれば事後分布が得られる (最後は忘れずに事後分布を規格化させること)。ただし、式 (27) では σ'^2 が使われており、ベイズ機械学習では考えていなかった。σ'^2 は元々のバラつき σ^2 に、真の値 y_t と数理モデルの差異が加わったものになる。仮に学習が成功し、後者が完全に一致するならば、σ'^2 と σ^2 も一致する。そこで本資料では、式 (27) の σ'^2 を σ^2 に代えた尤度

$$L = \prod_i \mathcal{N}(Y = y_i|\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}, \sigma^2) = \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})^2}{2\sigma^2}\right) \tag{30}$$

を使用する。また、σ^2 はセンサノイズに相当するため、データ同化の問題では既知であることが多い。本資料でも既知であると考える。

2.3.2 式展開を観察する

まずはパラメータ $\boldsymbol{w}$ の事前分布を

$$p(\boldsymbol{w}) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{w} - \boldsymbol{\mu})\right)$$

と設定する。これは平均 $\boldsymbol{\mu}$ の正規分布であるから、事前に $\boldsymbol{w}$ の値が $\boldsymbol{\mu}$ 付近であると考察したことに相当する。出力が分散 σ^2 の正規分布に従ってばらつくとき、データセットの尤度 $p(D|\boldsymbol{w}) = L$ は式 (30) になる。したがってベイズの定理より、事後分布は

$$p(\boldsymbol{w}|D) = \frac{C}{2\pi|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{w} - \boldsymbol{\mu})\right) \frac{1}{(2\pi\sigma^2)^{N/2}} \prod_i \exp\left(-\frac{(y_i - \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})^2}{2\sigma^2}\right) \tag{31}$$

と求まる。ここで C は規格化係数である。

パラメータの確率分布を調べたいというベイズ機械学習の目的は、これで達成したことになる。しかしながら、「どの値のときに事後分布が最大となるか」という情報は気になることもある。一般的に事後分布の最大値のことを MAP と言い、そのパラメータを推定することを MAP 推定と言う。

MAP 推定は当然ながら式 (31) の最大値探査問題である。ただし、前節同様 $-\ln p(\boldsymbol{w}|D)$ の最小化問題を考える。細かい計算は省くが、$-\ln p(\boldsymbol{w}|D)$ のうち $\boldsymbol{w}$ に依存する項のみ残した評価関数 J は

$$J = \frac{1}{2}(\boldsymbol{w} - \boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{w} - \boldsymbol{\mu}) + \frac{1}{2\sigma^2}\sum_i (y_i - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w})^2 \tag{32}$$

となる。第二項は σ^{-2} の重み付き二乗和誤差になっている。一方で第一項は事前分布時に設定した $\boldsymbol{\mu}$ から $\boldsymbol{w}$ が離れるほど大きな値を取るようになっている (このような測度をマハラノビス距離と言う)。そのため、MAP 推定は二乗和誤差と事前分布からのずれを最小化させる最適値探査問題だと考えられる。

一部の方は事前分布 (つまり主観確率) の利用にまだ抵抗があるかもしれない。主観確率の決め方によって解が変わってしまうのは、ときに信用できなく思えてしまう。確かに事前分布の設定が的外れであれば、第一項は足を引っ張ることになる。しかしながら、データ数 N が多くなるにつれ第二項の割合は大きくなる。ビッグデータを用意できるならば、ベイズ機械学習の結果は事前分布に影響されにくくなるので、安心してもよい。逆に事前分布が的を得ている場合、第一項は学習を助けてくれる。もしもデータ数が少なく、しかもデータの中に外れ値が含まれている場合、頻度主義的に第二項は学習を妨げる。そのとき第一項は外れ値に敏感に反応しないよう働きかけてくれる。

この議論は MAP 推定だけでなく確率分布自体にも言える。つまり、データ数が多くなれば事後分布は事前分布の影響を受けにくい。また、データ数が少ないとき、事後分布は事前分布のおかげで外れ値に対してロバストになる。

2.3.3 式展開を観察する 2

先ほどの例をここでは別の視点で観察する。式 (31) は正規分布の積なので、事後分布も正規分布となる。そこで式 (31) を展開し、まずは

$$\frac{1}{2\pi|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{\mu})^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{w}-\boldsymbol{\mu})\right)\frac{1}{(2\pi\sigma^2)^{1/2}}\exp\left(-\frac{(y_i-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})^2}{2\sigma^2}\right)$$

を計算してみる。これは 1 つ目のデータのみを学習に利用したときの事後分布に相当する。導出は省略するが、この結果は平均 $\boldsymbol{\mu}_1$ 及び分散共分散行列 Σ_1 が

$$\boldsymbol{\mu}_1=\Sigma_1\left(\Sigma^{-1}\boldsymbol{\mu}+\frac{1}{\sigma^2}y_1\boldsymbol{x}_1\right)$$

$$\Sigma_1=\left(\Sigma^{-1}+\frac{1}{\sigma^2}\boldsymbol{x}_1\boldsymbol{x}_1^{\mathrm{T}}\right)^{-1}$$

で与えられる。この確率分布 $\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1)$ を利用すると、式 (31) の事後分布は

$$p(\boldsymbol{w}|D)=\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1)\frac{C}{(2\pi\sigma^2)^{(N-1)/2}}\prod_{i=2}^{N}\exp\left(-\frac{(y_i-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})^2}{2\sigma^2}\right)$$

と書き換えられる。上式の i が 2 から開始していることに注目してほしい。上式は正に、事前分布を $\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1)$ とし、$i=2,...,N$ のデータで学習したときの事後分布と捉えられる。従って、「初めに用意された事前分布から、一つのデータのみ用いて事後分布を計算し、それを次の学習の際に事前分布として利用する」、そういった逐次的なデータ処理が可能なことが分かる。この逐次的処理は主に 2 つの点で魅力的である。まず、データセット D を集めきるのに時間を要する場合、得られたデータから先んじて処理を施すことができる。データを集めきる前に分布の遷移を確認できることは、多くの場面で便利になってくる (ただしこのような計算手法は頻度主義の機械学習でも可能である。逐次的処理がベイズ機械学習の専売特許だと考えることは誤解である)。もう一つが、学習に利用したデータは消去しても構わない点である。$p(\boldsymbol{W}|D)$ とう表記のせいか、「ベイズ機械学習は常に学習データを保存しておかなければならない」と誤解している人がたまにいる。しかしながら、上記の $\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1)$ が得られれば、最早データ (x_1,y_1) は不要で、メモリ上から消去しても構わない。

逐次的処理でデータ i まで学習に利用し、正規分布 $\mathcal{N}(\boldsymbol{\mu}_i,\Sigma_i)$ が得られているとする。次のデータ (x_{i+1},y_{i+1}) を消費したとき、上記と同様に正規分布の平均及び分散は

$$\boldsymbol{\mu}_{i+1}=\Sigma_{i+1}\left(\Sigma_i^{-1}\boldsymbol{\mu}_i+\frac{1}{\sigma^2}y_{i+1}\boldsymbol{x}_{i+1}\right) \tag{33}$$

$$\Sigma_{i+1}=\left(\Sigma_i^{-1}+\frac{1}{\sigma^2}\boldsymbol{x}_{i+1}\boldsymbol{x}_{i+1}^{\mathrm{T}}\right)^{-1} \tag{34}$$

で更新される。このように、尤度関数と事前分布が正規分布で、パラメータに対する演算も線形演算子のみである場合、事後分布も必ず正規分布となる。この条件を満たすベイズ機械学習では確率的考察を全くせず、式 (33)(34) を用いて只々機械的に学習を進めていくことができる (それゆえコーディングが容易)。

2.3.4 python コード例

式 (33)(34) による線形回帰の python コードを実装していく。まず、データセットを以下のように用意した。

```
import numpy as np
import math

sigma2 = 0.1
data_x = np.random.rand(100)
data_y = 2.*data_x + 1.

data_y += np.random.normal(0., math.sqrt(sigma2), size = 100)
```

ここで sigma2 は本資料中の σ^2 に相当する。data_x は説明変数 (x) であり、今回は [0,1] の一様乱数から個数 100 だけ生成した。data_y は目的変数 (y_t) である。したがって、今回の例では $y=2x+1$ を推定できれば学習成功ということになる。最後の行は目的変数にノイズを加える処理 $(y_t+\xi)$ である。

次に事前分布を設定する。事前分布を正規分布とし、その平均と分散共分散行列を以下のように仮定する。

```
mu = np.array([[0.], [0.]])
Sigma = np.eye(2)
```

ここで mu[0] は $y = ax + b$ における a、並びに mu[1] は b を指す。上記コードより、事前分布は $\mathcal{N}(\mathbf{0}, I)$ となる。
準備が整ったので、ここからは式 (33)(34) に従いデータを処理していく。

```
for x, y in zip(data_x, data_y):
        bmx = np.array([[x], [1.]])
        new_Sigma = np.linalg.inv(np.linalg.inv(Sigma) + (bmx@bmx.T)/sigma2)
        new_mu = new_Sigma@(np.linalg.inv(Sigma)@mu+y*bmx/sigma2)

        Sigma = new_Sigma
        mu = new_mu
```

ここで bmx は本資料中の $\boldsymbol{x}$ に相当する。式 (34) より new_Sigma を、式 (33) より new_mu を計算した。最後に Sigma と mu の定義を更新している。これをデータの数だけ繰り返した。
以下は全コードを纏めたものである。これを実行したところ、最終的に

$$\text{mu} = \begin{bmatrix} 1.91 \\ 1.07 \end{bmatrix}, \quad \text{Sigma} = \begin{bmatrix} 0.0108 & -0.00566 \\ -0.00566 & 0.00395 \end{bmatrix}$$

を得た。これは平均的に見て $y = 1.91x + 1.07$ と推定したことになるので、ある程度学習に成功したと言える。

```
import numpy as np
import math

sigma2 = 0.1
data_x = np.random.rand(100)
data_y = 2.*data_x + 1.
data_y += np.random.normal(0., math.sqrt(sigma2), size = 100)

mu = np.array([[0.], [0.]])
Sigma = np.eye(2)

for x, y in zip(data_x, data_y):
        bmx = np.array([[x], [1.]])
        new_Sigma = np.linalg.inv(np.linalg.inv(Sigma) + (bmx@bmx.T)/sigma2)
        new_mu = new_Sigma@(np.linalg.inv(Sigma)@mu+y*bmx/sigma2)

        Sigma = new_Sigma
        mu = new_mu
```

3 ノンパラメトリックベイズ機械学習

3.1 離散化されたノンパラメトリック確率分布の利用

前章では事前分布と尤度に正規分布を採用していた。事前分布と尤度の積も正規分布に従うため、式 (33)(34) のような正規分布のパラメータ ($\boldsymbol{\mu}$ と Σ) の更新アルゴリズムとして線形回帰を考えることができた。尤度が正規分布に従うことは多くの場合に言えることである (例えばセンサノイズ)。しかしながら、事前分布を正規分布と仮定することは如何であろうか。正規分布と仮定した結果、最終的に得られるパラメータのバラつきも正規分布と仮定したことになる。式 (33)(34) の更新則が利用できるとはいえ、ときにこの仮定は学習失敗の原因となる。
それゆえ、「正規分布に限らず他のパラメトリック確率分布も吟味すべきである」は結論の一つとして言える (ただし正規分布と正規分布以外の積は正規分布にならないので、式 (33)(34) の更新則は使えなくなることに注意)。しかしながら、一般的なパラメトリック確率分布のパラメータ数は少なく、分布の表現力は乏しい (1.4 節)。この問題に関する解決策は様々提案されているが、本資料ではノンパラメトリック確率分布による解法を紹介する。
ではまず初めに、離散化されたノンパラメトリック確率分布を考えよう。これは式 (21) のように書き表されるが、前章を踏まえた表記は

$$p(\boldsymbol{w}) = \sum_i^M p_i \delta(\boldsymbol{w} - \boldsymbol{w}_i)$$

の通りである。これは、$\boldsymbol{w}$ の候補として $\{\boldsymbol{w}_i | i = 1, ..., M\}$ を選定し、それぞれの確率を $\{p_i | i = 1, ..., M\}$ としたことに相当する (図 2(b) のイメージ)。
各 $\boldsymbol{w}_i$ の尤度は前章と変わらない。つまり式 (30) を使うことができ、

$$L = \prod_j \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_j - \boldsymbol{x}_j^{\mathrm{T}} \boldsymbol{w}_i)^2}{2\sigma^2} \right)$$

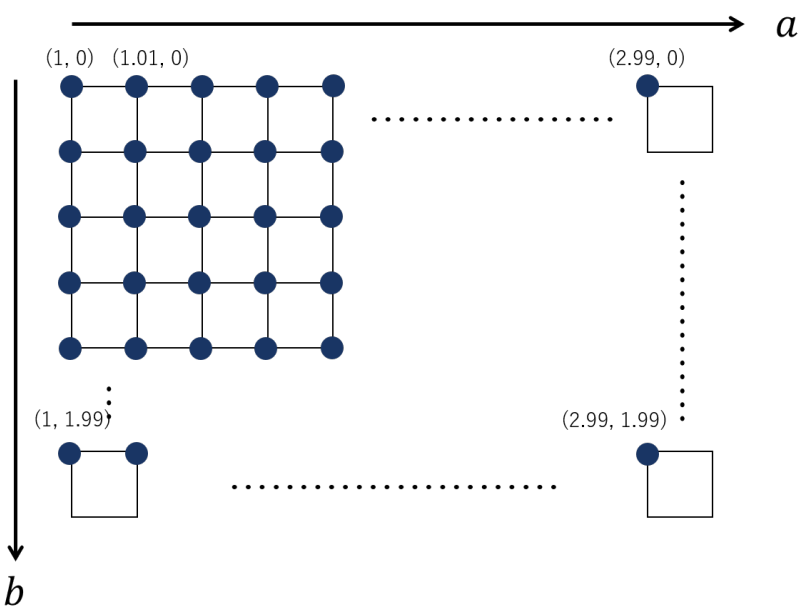

図 6　$\boldsymbol{w}$ の離散化の様子。

となる。したがって、$\boldsymbol{w}_i$ に関する事後分布は

$$p(\boldsymbol{w}_i|D) = p_i C \prod_j \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_j - \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{w}_i)^2}{2\sigma^2}\right) \tag{35}$$

と求まる (ここで C は規格化係数)。これを離散化した各 $\boldsymbol{w}_i$ に対して求めればよい。

3.1.1　python コード例

では、上記の計算を python で実装してみよう。前章と同様に $y = 2x + 1$ の学習を考える。

まず、$\boldsymbol{w}$ 空間を離散化する。a の範囲を [1,3)、b の範囲を [0,2) に限定し、それぞれ 0.01 の幅で離散化する。すると、$\boldsymbol{w}$ の取り得る値は 4 万個になる (図 6)。

```
import numpy as np

a = np.arange(1., 3., 0.01)
b = np.arange(0., 2., 0.01)

w = []
for ia in a:
        for ib in b:
                w.append(np.array([ia, ib]))
w = np.stack(w, axis = 0)
p = np.ones(len(w))/len(w)
```

上記コード中の w はパラメータのリストであり、図 6 中の青点に相当する。また、p は各パラメータの事前分布で本資料中の p_i に相当する。今回は事前分布に一様分布を採用したため、`p = np.ones(len(w))/len(w)` とした次第である。

次に、式 (35) に従ってデータ毎に尤度を掛けていく。

```
for x, y in zip(data_x, data_y):
        y_pred = w[:,0]*x + w[:,1]
        p *= np.exp(-(y-y_pred)**2/2/sigma2)/math.sqrt(2*np.pi*sigma2)

p /= np.sum(p)
```

ここで y_pred は本資料中の $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$ に相当する。また、`np.exp(−(y−y_pred)**2/2/sigma2)/math.sqrt(2*np.pi*sigma2)` は式 (35) の

$$\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_j-\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{w}_i)^2}{2\sigma^2}\right)$$

である。従って、これをデータ毎に掛けられた p は最終的に事後分布になることが分かる (ただし最後の行で規格化する)。

これまでのコードを纏めたものを以下に記す。これを実際に実行すると、w の平均値は

$$\mathrm{w} = \begin{bmatrix}2.14\\0.92\end{bmatrix}$$

となった。

```
import numpy as np
import math

a = np.arange(1., 3., 0.01)
b = np.arange(0., 2., 0.01)

w = []
for ia in a:
        for ib in b:
                w.append(np.array([ia, ib]))
w = np.stack(w, axis = 0)
p = np.ones(len(w))/len(w)

data_x = np.random.rand(100)
sigma2 = 0.1
data_y = 2.*data_x + 1. + np.random.normal(0., math.sqrt(sigma2), size = 100)

for x, y in zip(data_x, data_y):
        y_pred = w[:,0]*x + w[:,1]
        p *= np.exp(−(y−y_pred)**2/2/sigma2)/math.sqrt(2*np.pi*sigma2)

p /=np.sum(p)
```

3.2 アンサンブルによる表現

前述した通りノンパラメトリック機械学習は柔軟な分布の表現を可能にする。しかしながら、前節の手法は計算量の上で問題を抱えることがある。先ほどの例では a と b のパラメータをそれぞれ 200 分割した。それゆえ図 6 より $\boldsymbol{w}$ の組み合わせは全 4 万通りになる。そしてそれぞれの $\boldsymbol{w}$ に関して事後分布の計算を行う。つまり、4 万回の繰り返し計算を要することになる。より一般的に書くと、l 次元パラメータを各要素に対し n 分割すると、離散化されたパラメータは総じて n^l 通りある。事後分布の計算では n^l 回の繰り返し計算を要する。

1.4 節で述べた通り、離散数 n が増加するにつれノンパラメトリック確率分布の表現力は増す (CAE の計算格子との類似性)。また、データ同化では l が非常に大きい値となりやすい。例えば I.C. の状態ベクトル $\boldsymbol{f}$ が不確かな場合、$\boldsymbol{f}$ を学習対

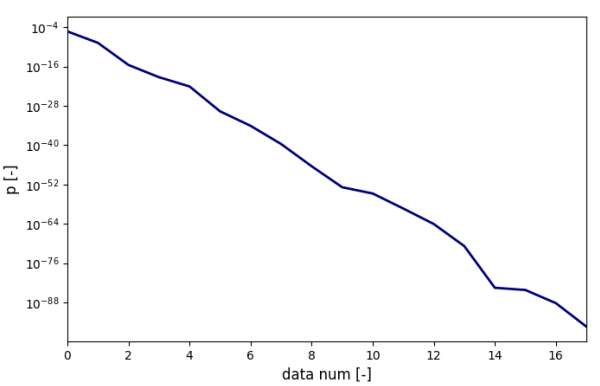

図 7　w=[1., 0.] における事後確率の遷移。

象として扱う訳であるが、このときの l は CAE 計算格子数だけの次元となる。従ってデータ同化の問題では n^l が非常に大きな値となってしまう。本節の目的は、この計算量を減らす方法の紹介である。

そこで、まずは図 7 のグラフを確認してほしい。これは前節の実装例における w=[1., 0.] の事後確率の遷移である。横軸は処理したデータの数であり、横軸がゼロのときの値は事前確率 (つまり 1/40000=2e-5) を示している。私たちは w=[2., 1.] が正解であると知っているため、[1., 0.] は的外れな推定であることも想像できる。そして実際、図 7 に示す通り $p([1.,0.]|D)$ は非常に小さな値になっていく。少しのデータを処理した時点で「もう見込みは無いな」と思えるパラメータは n^l の繰り返し計算から省きたい。つまり、事後確率が低くなったパラメータを計算の途中で除外できるようなアルゴリズムを考えれば、計算コストを削減できそうである。

そこで登場するのがアンサンブルによって表現されたノンパラメトリック確率分布である。いま事前分布が式 (22) のように

$$p(\boldsymbol{w}) = \frac{1}{M}\sum_i^M \delta(\boldsymbol{w} - \boldsymbol{w}_i)$$

で表されているとする。ここで $\boldsymbol{w}_i$ はサンプル値で、M は重複を許すサンプル数である。ただし、以後はデータ同化の慣習に合わせて、各サンプルのことを粒子と呼ぶことにする。

粒子 $\boldsymbol{w}_i$ に対するデータ (x_j, y_j) の尤度を

$$\beta_{ij} = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_j - \boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{w}_i)^2}{2\sigma^2}\right)$$

と定義する。各粒子の確率は $1/M$ であるため、データ (x_j, y_j) を処理した後の事後分布は $\beta_{ij}C/M$ となる。ここで C は規格化係数であるが、$\sum_i \beta_{ij}C/M = 1$ より、直ぐに $C = M/\sum_j \beta_{ij}$ だと分かる。よって、$\beta_{ij}/\sum_j \beta_{ij}$ が低い場合というのは、図 7 のような今後見込みの無い粒子ということになる。従って今後はあまり計算したくない粒子ということになるが、何か $\beta_{ij}/\sum_j \beta_{ij}$ に閾値を設けて継続可否を判断することはしない。その代わりに、M 個の粒子を $\beta_{ij}/\sum_j \beta_{ij}$ に従って再サンプリングする。このサンプリング結果は当然ながら事後分布に従う。ただし、有限個のサンプリングである以上、たとえ事後確率がゼロでなくても、一度もサンプリングされなかった粒子も現れる。そのため、事後確率の低い (つまり見込みの無い) 粒子は優先的に無視されることになる。このようにして、アンサンブルによる確率分布は計算の省略を行うことができる。

以上の説明を聞いて、進化計算のアルゴリズムを思い出した方がいるかもしれない。確かに本手法は進化計算と非常に似ており、データ同化の論文中で度々比較されている。以下に、逐次的データ処理における事後分布の更新アルゴリズムを記す。

アンサンブル表現によるノンパラメトリックベイズ機械学習

1. 所望の事前確率分布 $p(\boldsymbol{w})$ に従い、M 個の粒子 $\{\boldsymbol{w}_i|i = 1, ..., M\}$ をサンプリングする。
2. 各粒子 $\boldsymbol{w}_i$ に対して、一つのデータ (x, y) に関する尤度 β_i を計算する。
3. 粒子を $\beta_i/\sum_i \beta_i$ に従い M 個だけサンプリングする (このとき、同じ粒子を複数回サンプリングしてもよい)。このサンプリング結果は事後分布のアンサンブル $\{\boldsymbol{w}_i|i = 1, ..., M\}$ になる。
4. 2.-3. を繰り返す。

しかしながら、このアルゴリズムだと上手く行かないことがある。アルゴリズムを眺めると、1. で生成された粒子 $\{\boldsymbol{w}_i|i = 1, ..., M\}$ 以外は、未来永劫登場しないことが分かる。3. は粒子の多様性を減らす一方で、今まで見たことのないようなパラメータ値 $\boldsymbol{w}$ を生成することはない (粒子の多様性が酷く低下することを退化と呼ぶ)。図 8 のような場合を考えよう。2 次元のパラメータ空間に対して、正解は赤丸の値であったとする。一方で事前分布のサンプリング結果が青丸であった場合、上記アルゴリズムだと満足できる結果は得られないであろう。それゆえ各粒子がシフトできるような外部作用が欲しい。

そのための手法として、よく分布に正規分布のノイズが付加される。つまり、分布を構成する各粒子 $\boldsymbol{w}_i$ に対して $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \boldsymbol{\epsilon}$ の計算をする。ここで $\boldsymbol{\epsilon}$ は多次元の正規分布である。このノイズ付加の結果、分布は変化してしまう。せっか

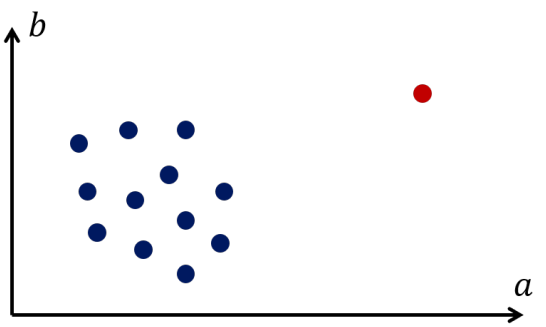

図 8　ノイズ付加がないと上手くいかない場合。赤丸は正解値。青丸は事前分布。

くベイズ機械学習を根拠に導かれた分布にも関わらず、それを変形させてしまうことは理にかなってないようにも思える。確かにその指摘は尤もである。しかしながら、数値計算の都合上 (つまり無限のサンプリングができないゆえに)、粒子の多様性を維持するためにどうしても必要なノイズと思って頂きたい (個人的に、このノイズは進化計算の突然変異に相当すると考えている)。せめて分布の期待値までも変わってしまわないように、ノイズの正規分布の平均はゼロと設定するのが一般的である。

ノイズを付加させた場合のアルゴリズムを以下に記す。

アンサンブル表現によるノンパラメトリックベイズ機械学習

1. 所望の事前確率分布 $p(\boldsymbol{w})$ に従い、M 個の粒子 $\{\boldsymbol{w}_i | i = 1, ..., M\}$ をサンプリングする。
2. $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \boldsymbol{\epsilon}$。
3. 各粒子 $\boldsymbol{w}_i$ に対して、一つのデータ (x, y) に関する尤度 β_i を計算する。
4. 粒子を $\beta_i / \sum_i \beta_i$ に従い M 個だけサンプリングする (このとき、同じ粒子を複数回サンプリングしてもよい)。このサンプリング結果は事後分布のアンサンブル $\{\boldsymbol{w}_i | i = 1, ..., M\}$ になる。
5. 2.-4. を繰り返す。

3.2.1 python コード例

上記の計算を python で実装してみる。前章と同様に $y = 2x + 1$ の学習を考える。

まず、$\boldsymbol{w}$ の事前分布に、前章と同様の一様分布を採用する (つまり a は [1,3] の一様分布、b は [0,2] の一様分布)。粒子数 M=1000 としたときの、事前分布に従う粒子生成を以下に記す。

```
import numpy as np
import math

M = 1000
a = 1.+2.*np.random.rand(M)
b = 2.*np.random.rand(M)
w = np.stack((a, b), axis = 1)
```

次に尤度計算とサンプリングのコードを示す。

```
for x, y in zip(data_x, data_y):
        w += np.random.normal(0., 0.01, (M, 2))

        y_pred = w[:,0]*x + w[:,1]
        beta = np.exp(-(y-y_pred)**2/2/sigma2)/math.sqrt(2*np.pi*sigma2)

        p = beta/np.sum(beta)
        w_index = np.random.choice(M, M, p = p)
        w = w[w_index]
```

上記 beta は各粒子の尤度を纏めた配列である。それゆえ p は各粒子の事後確率であり、`np.random.choice(M, M, p = p)` で事後分布に従ったサンプリングを行っている。ただしサンプリング結果は粒子のインデックス番号なので、次の行で該当するパラメータを抽出した。また、繰り返し計算の最初に、w にノイズを加えている。

これまでのコードを纏めたものを以下に記す。これを実行したところ、w の平均は

$$\mathrm{w} = \begin{bmatrix} 1.88 \\ 1.07 \end{bmatrix}$$

となった。

```
import numpy as np
import math

M = 1000
a = 1.+2.*np.random.rand(M)
b = 2.*np.random.rand(M)
w = np.stack((a, b), axis = 1)

data_x = np.random.rand(100)
sigma2 = 0.1
```

```
data_y = 2.*data_x + 1. + np.random.normal(0., math.sqrt(sigma2), size = 100)

for x, y in zip(data_x, data_y):
        w += np.random.normal(0., 0.01, (M, 2))

        y_pred = w[:,0]*x + w[:,1]
        beta = np.exp(-(y-y_pred)**2/2/sigma2)/math.sqrt(2*np.pi*sigma2)

        p = beta/np.sum(beta)
        w_index = np.random.choice(M, M, p = p)
        w = w[w_index]
```

4 データ同化の前準備

本章以降からようやくデータ同化の話に進む。ベイズ機械学習 (2-3 章) では線形回帰にしか触れていないが、それでも学ぶことは多かった。線形回帰以外の数理モデルであっても、2-3 章の議論は転用できるためである。しかしながら、このまま CAE を用いたベイズ機械学習 (つまりデータ同化) に進んでしまうのは不親切であるため、ここでベイズ機械学習の一般性のある表記を紹介しておく。

ベイズ機械学習の一般的表記

説明変数を $\boldsymbol{x} = (x_1, x_2, ..., x_n)^{\mathrm{T}}$、目的変数を $\boldsymbol{y} = (y_1, y_2, ..., y_m)^{\mathrm{T}}$ とし、数理モデル $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{w})$ を考える。ここで $\boldsymbol{w}$ は関数 f のパラメータをベクトルで纏めたものである。データセット $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)|i = 1, ..., N\}$ があるとき、パラメータ $\boldsymbol{w}$ を学習したい。
そこでベイズ機械学習の利用を考える。$\boldsymbol{w}$ の事前分布として $p(\boldsymbol{w})$ を用意する。多次元の目的変数はノイズを受け、分散共分散が Σ の正規分布に従ってばらつく。各データが独立にサンプリングされたとき、D に関する尤度関数 $p(D|\boldsymbol{w})$ は

$$p(D|\boldsymbol{w}) = \prod_i \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_i - f(\boldsymbol{x}_i, \boldsymbol{w}))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_i - f(\boldsymbol{x}_i, \boldsymbol{w}))\right)$$

となる。従って事後分布はベイズの定理より

$$p(\boldsymbol{w}|D) = Cp(\boldsymbol{w}) \prod_i \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_i - f(\boldsymbol{x}_i, \boldsymbol{w}))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_i - f(\boldsymbol{x}_i, \boldsymbol{w}))\right)$$

と求まる (ここで C は規格化係数)。

2-3 章と比べ、線形回帰の部分が $f(\boldsymbol{x}, \boldsymbol{w})$ に代わったこと、並びに尤度関数が多次元正規分布の積になったこと以外は同じであることが分かる。なお、多次元正規分布が使われるようになった理由は、目的変数 $\boldsymbol{y}$ が多次元になったためである。

4.1 システムモデルと観測モデル

4.1.1 システムモデル

ここで具体的な話に戻し、データ同化における $f(\boldsymbol{x}, \boldsymbol{w})$ の在り方を議論していく。ところで、CAE は支配方程式を満たす関数 (連続体力学では場と言う) の求解といえる。つまり、位置座標 $\boldsymbol{r}$ と時刻 t における物理量 $u(\boldsymbol{r}, t, \boldsymbol{w})$ を CAE では求めている。ここで $\boldsymbol{w}$ は B.C. などの CAE に必要なパラメータであり、CAE 実行時にいくつかパラメータを設定すると思われるが、それらをベクトルとして纏めたものと考えて頂ければ結構である (「B.C. はノイマンもしくはディリクレ」といった一見数値で表せないような場合でも、「ノイマン=1、ディリクレ=0」のように離散値でラベル付けすればよい)。

CAE は格子法と粒子法に大別できるが、どちらも物理場 $u(\boldsymbol{r}, t, \boldsymbol{w})$ を離散化して求めている。つまり、$u(\boldsymbol{r}, t, \boldsymbol{w})$ を陰関数の形ではなく、状態ベクトルとして求める。例えば格子法の場合、格子毎に物理量 u_{ij} が定義されているが (図 9)、これを一つに纏めたベクトル

$$\boldsymbol{u} = (u_{11}, u_{12}, ..., u_{1N}, u_{21}, ..., u_{2N}, ..., u_{MN})^{\mathrm{T}}$$

は、正にパラメータ $\boldsymbol{w}$ における時刻 t の物理場と言える (このような状態ベクトルによる表現は粒子法でも可能である)。なお、複数の物理量 (u と v) を扱う場合は、

$$\boldsymbol{u} = (u_{11}, v_{11}, u_{12}, v_{12}, ..., u_{1N}, v_{1N}, u_{21}, v_{21}, ..., u_{2N}, v_{2N}, ..., u_{MN}, v_{MN})^{\mathrm{T}}$$

のように重ねればよい。

当然ながら、CAE は時間方向に対しても離散化する。時刻 t の状態ベクトル $\boldsymbol{u}_t$ を与えたとき、次時間ステップの状態 $\boldsymbol{u}_{t+\Delta t}$ は CAE のアルゴリズムより求まる。したがって、CAE のことを「パラメータ $\boldsymbol{w}$ に従って、現時刻の状態ベクトル (説明変数) から次時刻の状態ベクトル (目的変数) を求める数理モデル」と定義できる。そこで、この機能を

$$\boldsymbol{u}_{t+\Delta t} = f(\boldsymbol{u}_t, \boldsymbol{w}) \tag{36}$$

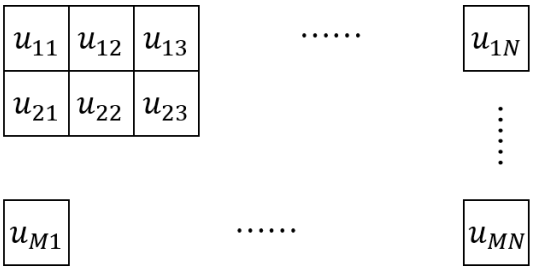

図 9　格子法における物理場の表現。それぞれの四角は計算格子で、u_{ij} はその計算格子における物理量を表している。

で表現できることにする。

式 (36) こそ正に CAE の数理モデル表現である。しかし残念ながら、データ同化においてこの表現はあまり用いられない。例えばパラメータが所与のとき、式 (36) による $\boldsymbol{u}_t$ から $\boldsymbol{u}_{t+\Delta t}$ の更新は確定的である。同様に I.C. として $\boldsymbol{u}_0$ を与えたとき、次時刻の状態ベクトル $\boldsymbol{u}_1$ も一つの値に決まる。また、次の状態 u_2 やその次の u_3 も一つの値に決まる。つまり I.C. の $\boldsymbol{u}_0$ が与えられれば、任意のステップの状態ベクトル $\boldsymbol{u}_{n\Delta t}$ は一意に決まり、式 (36) より

$$\boldsymbol{u}_{n\Delta t} = f(\boldsymbol{u}_{(n-1)\Delta t}, \boldsymbol{w}) = f(f(...f(\boldsymbol{u}_0))\boldsymbol{w}) = F(\boldsymbol{u}_0, \boldsymbol{w})$$

のような数理モデルで書き表すことができる。しかしながら、データ同化を用いる問題では、I.C. も未知であることが多い。そのため上式の $\boldsymbol{u}_0$ もパラメータとして扱う必要がある。

そこで $\boldsymbol{u}_0$ と $\boldsymbol{w}$ を改めて $\boldsymbol{x}_0$ に纏め、上式を $\boldsymbol{u}_{n\Delta t} = F(\boldsymbol{x}_0)$ に書き換える。更に、状態ベクトル $\boldsymbol{u}$ とパラメータ $\boldsymbol{w}$ を纏めたベクトルを $\boldsymbol{x}$ と定義すれば、上式は $\boldsymbol{x}_{n\Delta t} = F(\boldsymbol{x}_0)$ と書き換えられる。ただし、

$$\boldsymbol{x}_{n\Delta t} = F(\boldsymbol{x}_0) = \begin{bmatrix} f(f(...f(\boldsymbol{u}_0), \boldsymbol{w}) \\ \boldsymbol{w} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_{n\Delta t} \\ \boldsymbol{w} \end{bmatrix}$$

であり、$\boldsymbol{w}$ は何ら作用を受けないとする。

このように、物理量とパラメータを纏めたベクトルをデータ同化では好んで利用する。この慣習に倣えば、式 (36) の数理モデルは

$$\boldsymbol{x}_{t+\Delta t} = F(\boldsymbol{x}_t)$$

に書き換えられる。ちなみに $\boldsymbol{x}$ という表記は多くのデータ同化の書籍に倣った次第である。これまで説明変数に $\boldsymbol{x}$ を使ってきた分、以後は意味が変わるこに注意頂きたい。上式中の $\boldsymbol{x}$ は数理モデルのパラメータ $\boldsymbol{w}$ も含んでいる。そのため、これを説明変数と呼ぶことは難しいかもしれない。実際、データ同化の書籍では説明変数や目的変数といった言葉を用いない。単純に状態ベクトル $\boldsymbol{x}$ の更新として扱う (無理に言うなら $\boldsymbol{x}_t$ が説明変数で $\boldsymbol{x}_{t+\Delta t}$ が目的変数)。

さて、繰り返しになるが上式の F は CAE が担う。データ同化は最終的に実観測データとすり合わせを行うが、たとえ現在の状態ベクトル $\boldsymbol{x}_t$ が正しかったとしても (つまり正しい物理場とパラメータを用意できたとしても)、解析結果 $\boldsymbol{x}_{t+\Delta t}$ が正しいとは限らないことを、私たちは知っている。このずれは、支配方程式や解析手法そのものに問題があるときに発生する (データ同化は気象学の領域から発展してきたが、この分野では特にあり得る)。したがって、データ同化では上式の代わりに以下の数理モデルを考える。これをデータ同化の分野では特別にシステムモデルと呼ぶ。

データ同化のシステムモデル

$$\boldsymbol{x}_{t+\Delta t} = F(\boldsymbol{x}_t) + \boldsymbol{v}_t \tag{37}$$

ここで $\boldsymbol{v}_t$ はシステムノイズと言い、前述のずれに相当する。$\boldsymbol{v}_t$ の値は理論的に解くことができそうだが、一般的に不確かなものとして確率変数で考える (それゆえノイズという言われ方をしている)。個人的に、気象学の色が濃く出た結果、システムノイズを確率変数と考えるようになったと思っている。流体力学によると、ある程度微視的に見た流れの中には、様々な方向の渦が多数存在している。大きな計算格子ではこれら渦の作用を表せないので、特にラグランジュ形式の CFD ではランダムな作用を別途与える。これは式中で見れば正にシステムノイズに相当する。

とは言えデータ同化は CFD 以外でも利用する訳なので、そのときもランダムなシステムノイズを設定することに抵抗を感じるかもしれない。しかしながら、私はシステムノイズを加えるならいつでもランダムな確率変数にすることをお勧めする。このシステムノイズは解析値のずれと説明したが、別の見方をすれば 3.2 節の後半で登場したノイズに相当する。つまり、粒子の多様性を維持するのにシステムノイズは上手く働いてくれる。後ほど紹介する粒子フィルタやそれに近い手法では、逆にこういった利点を求めてシステムノイズを与えることが多い。

さて、システムノイズのバラつき加減は、流体力学の例のように根拠をもって設定されることもあるが、実際は解析者が恣意的に決めることの方が多い。バラつき加減 (例えば分散値) をハイパーパラメータとして考え、データ同化の成否をもとに調整することになる (なお、システムノイズの確率分布には正規分布がよく採用されている)。

4.1.2　観測モデル

私達が物理実験をするとき、まずは興味のある物理システムを実験装置内で構築し、センシングするのが一般的である。この実験装置内は物理場 $u(\boldsymbol{r}, t)$ なる状態にあるかもしれないが、この全ての情報をセンシングできることは稀である。その

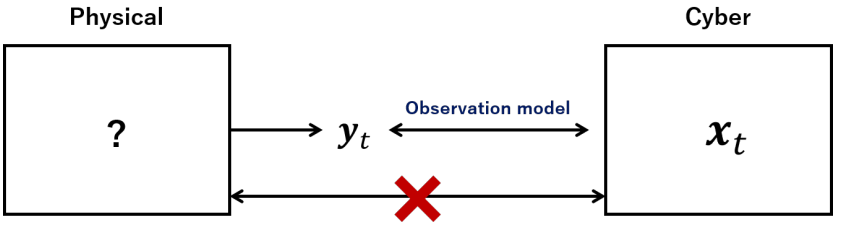

図 10　観測モデルについて。

ため、センサデータと物理場の情報には乖離がある。情報という観点では「センサデータは物理場に加工が施されたもの」とも言える。

本資料を通して教師データの代表例にセンサデータを挙げていた。また、当然ながら CAE は物理場の解析を行う。つまり、センサデータ $\boldsymbol{y}$ と状態ベクトル $\boldsymbol{x}$ の間にも乖離があるため、両者を繋ぐ (いわゆる加工を施す) モデルが必要になる (図 10)。このようなモデルを観測モデルと言う。

観測モデルの例 1

CAE 側の状態ベクトルを $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, w_1, w_2)^{\mathrm{T}}$ とする。ここで x_i は物理量、w_i はパラメータとする。従って、これは計算格子数が 5 個の解析に相当する。このうちセンサは x_2 と x_4 を測るものとする $(y = (x_2, x_4)^{\mathrm{T}})$。スパースな測定ゆえに状態ベクトルとセンサデータには乖離がある。

$\boldsymbol{x}$ から $\boldsymbol{y}$ を得るには以下のように線形変換を施せばよい。

$$\boldsymbol{y} = H\boldsymbol{x}, \quad H = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

上式は観測モデルのほとんどを成している。しかしながら、実際のセンサデータにはノイズが含まれていること、並びに CAE 結果である $\boldsymbol{x}$ が必ずしも正しいとは言えないことより、両辺にずれが存在する。ずれ $\boldsymbol{w}$ を含めた表記

$$\boldsymbol{y} = H\boldsymbol{x} + \boldsymbol{w}$$

を観測モデルと言う。また、この $\boldsymbol{w}$ を観測ノイズと言う。これは式 (26) の ϵ に相当する。従ってこの観測ノイズも後ほど確率変数として扱われる。式 (26) 中の ξ がセンサノイズに相当することから、観測ノイズのバラつき加減にセンサノイズの誤差を設定することは、厳密にいえば誤りである。しかしながら、CAE 結果が正しいと仮定して、観測ノイズ=センサノイズとすることも多い (実際に 2 章のベイズ機械学習では $\sigma'^2 \leftarrow \sigma^2$ としている)。

観測モデルの例 2

CAE の状態ベクトルを $\boldsymbol{m} = (m_1, m_2, m_3, m_4, m_5, w_1, w_2)^{\mathrm{T}}$ とする。ここで m_i は各計算格子の質量 (密度と計算格子体積の積) である。また、センサデータ y は物理システムの総重量であるとする。このとき、y と $\boldsymbol{m}$ の関係は、観測ノイズを含めた形で

$$y = H\boldsymbol{x} + \boldsymbol{w}, \quad H = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0]$$

と表記できる。このように、センサデータがシステム全体に関する情報であってもよい。

観測モデルの例 3

CAE 側の状態ベクトルを $\boldsymbol{v} = (v_1, v_2, v_3, v_4, v_5, w_1, w_2)^{\mathrm{T}}$ とする。ここで $\boldsymbol{v}_i$ は各計算格子の速さである。一方のセンサデータ y は計算格子 3 の運動エネルギーを測定する場合、両者の関係は

$$y = H(\boldsymbol{v}) + \boldsymbol{w}, \quad H(\boldsymbol{v}) = \frac{1}{2} m v_3^2$$

となる。このように、非線形な変換を観測モデルとして採用してもよい。

以上が観測モデルの例である。なお、例 1 や 2 では要素の抽出として線形変換を明記していたが、例 3 のように省略して書いてしまうことも多い。以下は観測モデルの一般的な表記である。

データ同化の観測モデル

$$\boldsymbol{y}_t = H(\boldsymbol{x}_t) + \boldsymbol{w}_t \tag{38}$$

4.2 非逐次型と逐次型のデータ同化

前節でシステムモデルと観測モデルを学んだ。そこで、本章の初めにベイズ機械学習の一般的表記を紹介したが、これをシステムモデルと観測モデルによる表現に書き換えてみよう。ただし、データ同化には非逐次型と逐次型の 2 種類に大別できる。詳細は後述するが、今は「データを得る度に確率分布を更新する方を逐次型と言う」だけの認識で十分である。つまり、逐次型は式 (33)(34) の更新的考えに似ている。

4.2.1 非逐次型データ同化

非逐次型データ同化の例は 5 章で紹介する。以下は非逐次型データ同化におけるベイズ機械学習の考え方である。なお、非逐次型データ同化ではシステムノイズを無視することが多い。本資料でもそれに倣うことにした。

非逐次データ同化

状態ベクトルの時系列を、$\{\boldsymbol{x}_t|t=1,...,T\}$、同時刻のセンサデータを $\{\boldsymbol{y}_t|t=1,...,T\}$ で表す。ここで $\boldsymbol{x}_t$ はシステムモデル

$$\boldsymbol{x}_{t+1} = F(\boldsymbol{x}_t)$$

に従い時間発展し、$\boldsymbol{y}_t$ と $\boldsymbol{x}_t$ の間には

$$\boldsymbol{y}_t = H(\boldsymbol{x}_t) + \boldsymbol{w}_t$$

の関係がある。パラメータを要素に含む $\boldsymbol{x}_t$ はベイズ機械学習において確率変数として扱われる。また、$\boldsymbol{y}_t$ も観測ノイズがあるため確率変数である。

初期時刻の $\boldsymbol{x}$ を $\boldsymbol{x}_0$、$\boldsymbol{x}_0$ の初期分布 (つまり事前分布) を $p(\boldsymbol{x}_0)$ とする。$\boldsymbol{x}_0$ が所与のとき $\{\boldsymbol{x}_t|t=1,...,T\}$ が得られたとするならば、センサデータ $Y=\{\boldsymbol{y}_t|t=1,...,T\}$ の尤度 $p(Y|\boldsymbol{x}_0)$ は

$$p(Y|\boldsymbol{x}_0) = \prod_t \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))\right)$$

となる。したがって $\boldsymbol{x}_0$ の事後分布は

$$p(\boldsymbol{x}_0|Y) = Cp(\boldsymbol{x}_0)\prod_t \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))\right) \tag{39}$$

で求まる (ここで C は規格化係数)。

上記の $\boldsymbol{x}_0$ には B.C. や物性値、並びに I.C. の情報が含まれている。つまり、表 1 の学習対象の全てを表すベクトルな訳であり、その事後分布 (式 (39)) は正に所望の学習結果と言える。

このように、非逐次型データ同化は一般的なベイズ機械学習となんら変わらない。ただし、$\boldsymbol{x}_0$ の次元は一般的な機械学習において見られないほど大きいこともあり、学習結果の精度や計算コストの面で苦労することがある。

4.2.2 逐次型データ同化

逐次型データ同化の例は 6 章で紹介する。以下は逐次型データ同化におけるベイズ機械学習の考え方である。なお、逐次型データ同化には平滑化という技術があるが、本資料では紹介しない。今後も平滑化なる手法が無いものとした説明になっていることにご留意頂きたい。

逐次型データ同化では次時刻のセンサデータまでしか考えない。初期時刻の状態ベクトルを $\boldsymbol{x}_0$ とし、次時刻の状態ベクトルを $\boldsymbol{x}_1$ とする。また、時刻 1 のときのセンサデータを $\boldsymbol{y}_1$ とする。

$\boldsymbol{x}_0$ が所与のとき、$\boldsymbol{x}_1$ の条件付確率 $p(\boldsymbol{x}_1|\boldsymbol{x}_0)$ はシステムモデル $\boldsymbol{x}_1 = F(\boldsymbol{x}_0) + \boldsymbol{v}_1$ より求まる (例えば $\boldsymbol{v}_t$ が $\mathcal{N}(\boldsymbol{0},\Sigma_v)$ に従うとき、$p(\boldsymbol{x}_1|\boldsymbol{x}_0)$ に関する確率分布は $\mathcal{N}(F(\boldsymbol{x}_0),\Sigma_v)$ となる)。したがって、$\boldsymbol{x}_0$ に関する確率分布 $p(\boldsymbol{x}_0)$ を用意すれば、$\boldsymbol{x}_0$ と $\boldsymbol{x}_1$ の同時確率分布 $p(\boldsymbol{x}_0,\boldsymbol{x}_1)$ は $p(\boldsymbol{x}_1|\boldsymbol{x}_0)p(\boldsymbol{x}_0)$ より求まる。この確率を更に $\boldsymbol{x}_0$ に関して積分すれば、$\boldsymbol{x}_1$ に関する周辺確率 $p(\boldsymbol{x}_1)$ が得られる。逐次型データ同化ではこの $p(\boldsymbol{x}_1)$ を事前分布として扱う。そうすれば状態ベクトル $\boldsymbol{x}_1$ に対する $\boldsymbol{y}_1$ の尤度 $p(\boldsymbol{y}_1|\boldsymbol{x}_1)$ は

$$p(\boldsymbol{y}_1|\boldsymbol{x}_1) = \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_1 - H(\boldsymbol{x}_1))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_1 - H(\boldsymbol{x}_1))\right)$$

より求まり、$\boldsymbol{x}_1$ の事後分布は

$$p(\boldsymbol{x}_1|\boldsymbol{y}_1) = Cp(\boldsymbol{x}_1)\frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_1 - H(\boldsymbol{x}_1))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_1 - H(\boldsymbol{x}_1))\right)$$

となる (ここで C は規格化係数)。

次の時刻のセンサデータ $\boldsymbol{y}_2$ が得られたとき、$\boldsymbol{x}_1$ の分布に上式の $p(\boldsymbol{x}_1|\boldsymbol{y}_1)$ を利用し、同様に $p(\boldsymbol{x}_2)$ の計算、そして事後分布 $p(\boldsymbol{x}_2|\boldsymbol{y}_2)$ を計算する。これを繰り返す処理が正に逐次型データ同化である。

逐次型データ同化

状態ベクトル $\boldsymbol{x}_{t-1}$ に関する確率分布を $p(\boldsymbol{x}_{t-1})$ とする。$\boldsymbol{x}_{t-1}$ が所与のときの $\boldsymbol{x}_t$ に関する確率分布 $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ をシステムモデル

$$\boldsymbol{x}_t = F(\boldsymbol{x}_{t-1}) + \boldsymbol{v}_t$$

より求める。すると $\boldsymbol{x}_{t-1}$ と $\boldsymbol{x}_t$ の同時確率 $p(\boldsymbol{x}_t, \boldsymbol{x}_{t-1})$ を周辺化することで、$p(\boldsymbol{x}_t)$ が求まる。これを事前分布として利用する。

時刻 t におけるセンサデータを $\boldsymbol{y}_t$ とする。この尤度 $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ は

$$p(\boldsymbol{y}_t|\boldsymbol{x}_t) = \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))\right)$$

であるため、$\boldsymbol{x}_t$ の事後分布は

$$p(\boldsymbol{x}_t|\boldsymbol{y}_t) = Cp(\boldsymbol{x}_t) \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_t - H(\boldsymbol{x}_t))\right) \tag{40}$$

となる (ここで C は規格化係数)。式 (40) を利用して更に $p(\boldsymbol{x}_{t+1})$ を計算し、上記の処理を繰り返す。

上記処理より、最終的に学習された確率分布 $p(\boldsymbol{x}_T)$ が得られる。式 (40) の事後分布を次時刻で事前分布として扱う点は、2 章で紹介した更新的アルゴリズムに似ている。

逐次型データ同化で最も特徴的なのは、やはり更新があることだろう。発展的な手法の多くは、更新のタイミングに追加の処理を施すことで、データ同化の性能を上げている。こういった余地がある分、逐次型データ同化の方が柔軟的だと個人的に思っている。

しかしながら、一度にデータを処理しないゆえの欠点も存在する。式 (39) と (40) を見比べたとき、非逐次型データ同化は初期の $\boldsymbol{x}_0$ の学習をしていること、そして逐次型データ同化は最終時刻 $\boldsymbol{x}_T$ の学習をしていることに気付く。システムモデルを用いれば $\boldsymbol{x}_0$ の確率分布から $\boldsymbol{x}_T$ の確率分布は求まるので、$\boldsymbol{x}_T$ に関してしか分からない点で逐次型データ同化は劣っていると言える。逐次型データ同化は I.C. の不確かさに関しては何も教えてくれないことになるので、問題によっては不便に感じるかもしれない。また、逐次型データ同化では時間経過とともにパラメータ $\boldsymbol{w}$ の分布も変わることになる。そのため、推測された物理量 $\boldsymbol{u}_t$ の時間発展も物理的に不自然な挙動を示す (図 16)。最終時刻の $\boldsymbol{x}_T$ が正しければ問題ないことも多いが、この不自然さはときに逐次型データ同化を選ばない理由となる。

4.3 Lorenz63 モデルの学習データ

次の章からデータ同化について本格的に議論していく。具体的な理解を促すため、python コードによる実装例を紹介することにした。そこで、具体的なシミュレーション問題が必要になる訳だが、本資料では Lorenz63 を採用する。

Lorenz63 モデルは 3 つの変数 (x, y, z) と 3 つのパラメータ (p, r, b) からなる、

$$\frac{dx}{dt} = -p(x - y)$$

$$\frac{dy}{dt} = -xz + rz - y$$

$$\frac{dz}{dt} = xy - bz$$

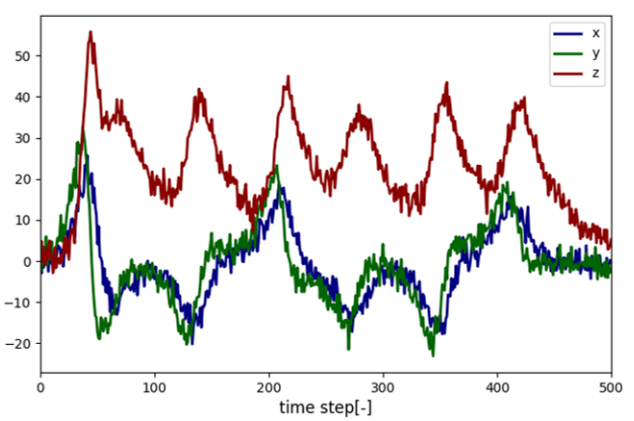

図 11　Lorenz63 モデルの生成データ (観測ノイズ込み)。

で表されるモデルである。ここで、本資料では原著に従い $p = 28$、$r = 10$、$b = 8/3$ とした。そして時間方向に関して離散化し、

$$x_t = x_{t-\Delta t} - p(x_{t-\Delta t} - y_{t-\Delta t})\Delta t$$

$$y_t = y_{t-\Delta t} + (-x_{t-\Delta t}z_{t-\Delta t} + rz_{t-\Delta t} - y_{t-\Delta t})\Delta t$$

$$z_t = z_{t-\Delta t} + (x_{t-\Delta t}y_{t-\Delta t} - bz_{t-\Delta t})\Delta t$$

なる更新式を作った。時間刻みは 0.01 とした。

本資料では 3 つのパラメータを既知とし、(x, y, z) の初期値 (1, 0, 0) のみ未知であるとする。また、(x, y, z) を直接観測できるとし、観測ノイズは $\mathcal{N}(\mathbf{0}, 4I)$ に設定した。学習データに関しては以上の条件のもと生成した (図 11)。以下に生成データを出力する関数を示す。引数の noise が False のときは、結果に観測ノイズを加えないことにする。また、関数の出力は 2 次元配列で、時刻 1 以降の結果である (行数は time_step、列数は 3)。

```
import numpy as np

def Lorenz63(x0=1., y0=0., z0=0., time_step = 、500noise = True):
        p = 10.
        r = 28.
        b = 8./3.
        dt = 0.01

        x = np.zeros(time_step+1); x[0] = x0
        y = np.zeros(time_step+1); y[0] = y0
        z = np.zeros(time_step+1); z[0] = z0

        for t in range(1, time_step+1):
                x[t] = x[t-1] + dt*(-p*x[t-1]+p*y[t-1])
                y[t] = y[t-1] + dt*(-x[t-1]*z[t-1]+r*x[t-1]-y[t-1])
                z[t] = z[t-1] + dt*(x[t-1]*y[t-1]-b*z[t-1])

        data = np.stack((x, y, z), axis = 1)
        if noise:
                data += np.random.normal(0., 2., size = data.shape)

        return data[1:]
```

5 非逐次型データ同化

5.1 4 次元変分法

代表的な非逐次型データ同化として 4 次元変分法がある。4 次元変分法の事後分布推定は正に 4.2.1 節の通りである。ただし、4 次元変分法では式 (39) の MAP 推定まで考えることが多い。

そこで、式 (39) の最大値探査問題の代わりに、$-\ln p(\boldsymbol{x}_0|Y)$ の最小化問題を考える。$-\ln p(\boldsymbol{x}_0|Y)$ のうち、$\boldsymbol{x}_0$ に依存する項のみを残した評価関数 $J(\boldsymbol{x}_0)$ は以下の通りになる。

4 次元変分法における MAP 推定

$$J(\boldsymbol{x}_0) = \frac{1}{2}\sum_t (\boldsymbol{y}_t - H(\boldsymbol{x}_t))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y}_t - H(\boldsymbol{x}_t)) - \ln p(\boldsymbol{x}_0) \tag{41}$$

上式の第一項はセンサデータに対する二乗和誤差に似ている。この解釈は正しく、マハラノビス距離における二乗和誤差と言われている。また、第二項は事前確率が高い $\boldsymbol{x}_0$ を是とした評価関数である。この評価関数は式 (32) と非常に似ており、2.3.2 節での議論はここでも使える。つまり、データ数が多くなると事前分布の寄与は小さくなる。一方でデータに外れ値が含まれている場合、事前分布の存在は学習のロバスト性を向上させる。もしも $\boldsymbol{x}_0$ に関して何ら知見がなければ、第一項のみの学習、つまり頻度主義的な機械学習を行えばよい。

式 (41) の最小化は何となく簡単に見えるが、計算コストの面で意外と大変だったりする。一般的に、データ同化で扱うベクトル $\boldsymbol{x}$ は次元が大きい (計算格子数だけ必要になることもある)。従って基本的に最適化には勾配法が使われるが、高次元のベクトルに対する微分を計算するのはときに大変である。勾配を求める際は、アジョイント法もしくは微分可能プログラミングによる実装が必要になるかもしれない。

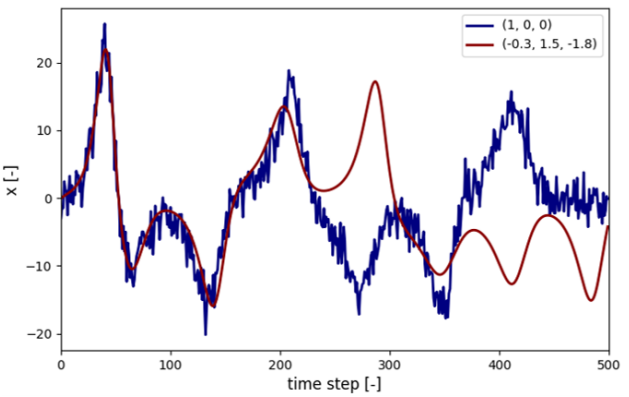

図 12　4 次元変分法の結果と正解値の比較。(1, 0, 0) の結果が正解値 (ただし観測ノイズが含まれている)。(-0.3, 1.5, -1.8) の結果が 4 次元変分法による推定結果。

5.1.1　python コード例

4 次元変分法で Lorenz63 モデルの初期値 $\boldsymbol{x}_0 = (x_0, y_0, z_0)$ を推定する。今回は事前分布に関する知識はなく、式 (41) 中の第一項のみを考えるとする (つまり頻度主義的な学習)。観測ノイズを $\mathcal{N}(\boldsymbol{0}, 4I)$ としたとき、式 (41) は

$$J(\boldsymbol{x}_0) = \frac{1}{8}\sum_i |\boldsymbol{y}_t - \boldsymbol{x}_t|^2$$

に書き換えられる。ここで $\boldsymbol{y}_t$ は時刻 t の観測データ、$\boldsymbol{x}_t$ は初期値を $\boldsymbol{x}_0$ としたときの推定値である。

$\boldsymbol{x}_0$ に関する J の微分 ∇J は差分法で求める。また、連続値最適化には最急降下法を用いる。つまり探索点は $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_0 - \alpha\nabla J$ に従い更新する。ここで α は緩和係数である。

まず上式の J を返す関数を定義した。なお、上式では各データに対し和を取っているが、今回は評価関数値の発散を防ぐために平均値とした。

```
import numpy as np

def J(x, y):
        return np.mean((y-x)**2)/8.
```

次に、学習データを生成する。コード中の y は本資料の $\{y_t | t = 1, ..., T\}$ に相当する。また、pred_init は初期値の推定値にあたる。後の反復計算でこの配列を更新していく。dh は差分計算時のステップ幅で、alpha は緩和係数である。

```
y = Lorenz63()

pred_init = np.array([0., 0., 0.])
dh = 0.001
alpha = 1e-3
```

次に勾配の計算および pred_init 更新部分のコードを示す。今回は最急降下法のイタレーション回数を 10000 回とした。初めの 4 行は pred_init 及びその近傍の Lorenz63 モデル結果を計算している。その後それぞれの差分から勾配を近似計算し、pred_init の更新を行った。

```
for itr in range(10000):
        x = Lorenz63(pred_init[0], pred_init[1], pred_init[2], noise = False)
        x_diff = Lorenz63(pred_init[0]+dh, pred_init[1], pred_init[2], noise = False)
        y_diff = Lorenz63(pred_init[0], pred_init[1]+dh, pred_init[2], noise = False)
        z_diff = Lorenz63(pred_init[0], pred_init[1], pred_init[2]+dh, noise = False)

        gradxJ = (J(y, x_diff)-J(y, x))/dh
        gradyJ = (J(y, y_diff)-J(y, x))/dh
        gradzJ = (J(y, z_diff)-J(y, x))/dh

        pred_init[0] -= alpha*gradxJ
```

```
        pred_init[1] -= alpha*gradyJ
        pred_init[2] -= alpha*gradzJ
```

学習がうまく行けば pred_init=(1, 0, 0) となるはずだが上手くいかず、結果は (-0.3, 1.5, -1.8) となった。ただし、$\boldsymbol{x}_0 = (1, 0, 0)$ と (-0.3, 1.5, -1.8) のときの x の結果を比較したところ、カオス性が増す後半以外は一致している (図 12)。なにより適当な最急降下法や差分計算を考えれば、初期値に敏感な Lorenz63 モデルとしてはそこそこ良いんじゃないかなと思える。

以上のコードを纏めたものを以下に記す。

```
import numpy as np

def J(x, y):
        return np.mean((y-x)**2)/8.

y = Lorenz63()

pred_init = np.array([0., 0., 0.])
dh = 0.001
alpha = 1e-3

for itr in range(10000):
        x = Lorenz63(pred_init[0], pred_init[1], pred_init[2], noise = False)
        x_diff = Lorenz63(pred_init[0]+dh, pred_init[1], pred_init[2], noise = False)
        y_diff = Lorenz63(pred_init[0], pred_init[1]+dh, pred_init[2], noise = False)
        z_diff = Lorenz63(pred_init[0], pred_init[1], pred_init[2]+dh, noise = False)

        gradxJ = (J(y, x_diff)-J(y, x))/dh
        gradyJ = (J(y, y_diff)-J(y, x))/dh
        gradzJ = (J(y, z_diff)-J(y, x))/dh

        pred_init[0] -= alpha*gradxJ
        pred_init[1] -= alpha*gradyJ
        pred_init[2] -= alpha*gradzJ
```

5.2 MCMC 法による事後分布推定

本節では一風変わったデータ同化を紹介する。これは式 (39) に従って $\boldsymbol{x}_0$ をサンプリングする手法である。つまり、式 (39) の分布を式 (22) の形で表現する。ただし、$p(\boldsymbol{x}_0|Y)$ の確率分布を求めてからそれに従ってサンプリングするような、二度手間なことはしない。それよりも効率よくサンプリングするために、マルコフ連鎖モンテカルロ法 (MCMC 法) を利用する。MCMC 法はベイズ統計でよく用いられており、様々なサンプリング手法を一括りにした総称である (MCMC 法だけで一冊の本が書ける)。本資料では最も簡単な MCMC 法の一つ、メトロポリス法を紹介する。

5.2.1 メトロポリス法

確率分布 $p(x)$ からサンプリングする方法を考える。ここで x は連続確率変数であり、任意の $p(x)$ を把握もしくは計算できるものとする (上記で $p(\boldsymbol{x}_0|Y)$ を把握せずにサンプリングするよう述べたので矛盾に感じるかもしれないが、一旦忘れてほしい。最終的に解決させる)。

メトロポリス法は焼きなまし法に似ている。まず、適当な初期位置 x^0 から初め、$p(x^0)$ を計算する。次に x^0 からランダム変数 ϵ だけ移動し $x^1 = x^0 + \epsilon$ の確率値 $p(x^1)$ を計算する。ただし、$p(x^0) > p(x^1)$ ならば $1 - p(x^1)/p(x^0)$ の確率で x の更新を取り消し、$x^1 = x^0$ とする。

これを $x^2, x^3 ...$ と繰り返したときのデータセット $\{x^i | i = 0, ..., N\}$ について、図 13 を使って考えていこう。図 13(a) に示す通り、$p(x^0) \leq p(x^1)$ ならばデータ x^t は無条件に更新される。一方で確率値が減少する方向の場合、確率 $1 - p(x^1)/p(x^0)$ でその場に留まる。したがってこのサンプリング方法は高確率な部分を優先的に選んでいる。一方で確率 $p(x^1)/p(x^0)$ で確率値の低いデータも許すため、極大値のみサンプリングしすぎるようにはなっていない。このようなサンプリング手法をメトロポリス法と言う。

しかしながら、この方法は初期位置 x^0 に依存する。図 13(b) のように初期位置が分布の裾にある場合、高確率な位置に移動するまである程度タイムステップを要する。それまでのデータは確率値の低いところにあるため、確率分布に従ったサンプルとは言えない。妥当なサンプリングになるまでに要する時間のことをバーンイン期間と言う。したがって、メトロポリス法を用いたのち、バーンイン期間のデータを捨てたもの $\{x_i | i = b, ..., N\}$ をサンプリング結果として使う (ここで b はバーンイン期間後のタイムステップ値)。残念ながらバーンイン期間の良い見積もり方法を私は知らない。

最後に、メトロポリス法のアルゴリズムを記す。

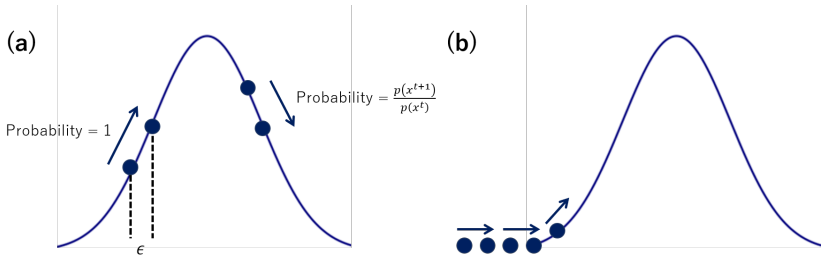

図 13　(a) メトロポリス法のサンプリング方法。(b) バーンイン期間のイメージ。

メトロポリス法

1. 初期値 x を設定する。
2. x からランダム変数 ϵ だけ移動させる。$x' \leftarrow x + \epsilon$。
3. 2. に関して $p(x') < p(x)$ の場合、確率 $1 - p(x')/p(x)$ で x' の移動を取り消す。$x' \leftarrow x$。
4. $x \leftarrow x'$。
5. 2.-5. を繰り返す。各ステップの x を保存。
6. バーンイン期間の x を取り除く。

5.2.2　python コード実装例

メトロポリス法を用いて標準正規分布 $\mathcal{N}(0,1)$ からのサンプリングを試みる。まず、x の初期値を [-1,1] の一様乱数で生成する。バーンイン期間を 100 とし、10000 のサンプルを生成することにした。サンプルデータはリスト X に保存していく。

```
import numpy as np

burn_in = 1000
num = 10000
x = 2.*(np.random.rand()−0.5)

X = []; X.append(x)
```

次にメトロポリス法に従いサンプリングしていく。x を [-0.1, 0.1] の一様乱数で更新しコード中の x_next を得る。x と x_next の確率値を比較し、X に格納する方を決める。これをバーンイン期間と所望のサンプル数だけ繰り返す。最後に X=X[burn_in+1:] とすることでバーンイン期間のデータを捨てる。

```
for itr in range(burn_in+num):
        x_next = x + 0.2*(np.random.rand()−0.5)

        p = np.exp(−x**2 / 2) / np.sqrt(2 * np.pi)
        p_next = np.exp(−x_next**2 / 2) / np.sqrt(2 * np.pi)

        if p_next < p:
                r = 1. − p_next/p
                if np.random.rand() < r:
                        X.append(x)
                else:
                        X.append(x_next)
        else:
                X.append(x_next)

        x = X[−1]

X = X[burn_in+1:]
```

以下はこれまでのコードを纏めたものである。また、図 14 にサンプリング結果を示す。

```
import numpy as np

burn_in = 1000
num = 10000
x = 2.*(np.random.rand()—0.5)

X = []; X.append(x)

for itr in range(burn_in+num):
        x_next = x + 0.2*(np.random.rand()—0.5)

        p = np.exp(—x**2 / 2) / np.sqrt(2 * np.pi)
        p_next = np.exp(—x_next**2 / 2) / np.sqrt(2 * np.pi)

        if p_next < p:
                r = 1. — p_next/p
                if np.random.rand() < r:
                        X.append(x)
                else:
                        X.append(x_next)
        else:
                X.append(x_next)

        x = X[—1]

X = X[burn_in+1:]
```

5.2.3 事後分布に従うサンプリング方法

ここからはデータ同化に戻り、式 (39) に従うサンプリングを考える。ただし、繰り返しになるが確率分布 $p(\boldsymbol{x}_0|Y)$ を調べてからサンプリングするような、二度手間なことはしない。

一方で事前分布 $p(\boldsymbol{x}_0)$ に関しては、任意の $\boldsymbol{x}_0$ に対して確率値を把握しているとする。そこでメトロポリス法の利用を検討する。

まず $\boldsymbol{x}_0$ の初期値 $\boldsymbol{x}^0$ を設定する。$\boldsymbol{x}^0$ のときの尤度 $p(Y|\boldsymbol{x}^0)$ はこれまで通り求めることができる。したがって $\boldsymbol{x}^0$ のときの事後確率は

$$p(\boldsymbol{x}^0|Y) = Cp(\boldsymbol{x}^0)p(Y|\boldsymbol{x}^0)$$

となる (ここで C は規格化係数)。そして $\boldsymbol{x}^0$ をランダムに移動させた $\boldsymbol{x}^1 = \boldsymbol{x}^0 + \boldsymbol{\epsilon}$ に関しても同様に事後確率

$$p(\boldsymbol{x}^1|Y) = Cp(\boldsymbol{x}^1)p(Y|\boldsymbol{x}^1)$$

を計算することができる (なお、ベイズの定理より規格化係数は両式とも同じ値である)。確率値の比

$$\frac{p(\boldsymbol{x}^1|Y)}{p(\boldsymbol{x}^0|Y)} = \frac{p(\boldsymbol{x}^0)}{p(\boldsymbol{x}^1)}\frac{p(Y|\boldsymbol{x}^1)}{p(Y|\boldsymbol{x}^0)}$$

をもとに $\boldsymbol{x}$ の位置を決める。

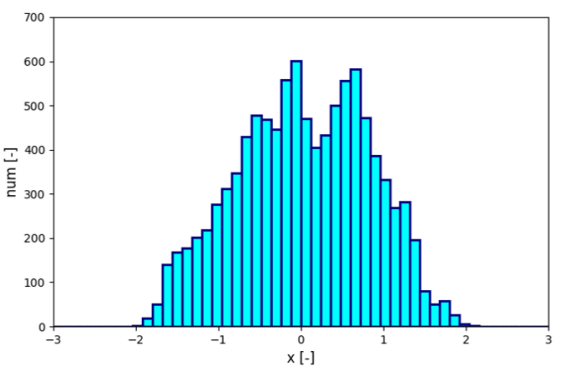

図 14　メトロポリス法によるサンプリング結果。

これを繰り返した結果 $\{\boldsymbol{x}^i | i = b, ..., N + b\}$ は事後分布に関するサンプルデータとなる。従って事後分布は式 (22) より

$$p(\boldsymbol{x}_0|Y) \simeq \frac{1}{N - b + 1} \sum_{i=b}^{N} \delta(\boldsymbol{x}_0 - \boldsymbol{x}^i)$$

と推定される。

5.2.4 python コード例

MCMC 法を用いて Lorenz63 モデルの初期位置 $\boldsymbol{x}_0$ の事後分布を求める。下記コードの通り、学習データ y は 100 タイムステップまでとした。またバーンイン期間を 1000 ステップ、サンプル数を 10000 とした。$\boldsymbol{x}_0$ の事前分布は $\mathcal{N}(\boldsymbol{\mu}, I)$、$\boldsymbol{\mu} = (0.5, 0., 0.)^{\mathrm{T}}$ とした。コード中にある通り、メトロポリス法の初期位置 x0 を事前分布に従って抽出した。また、リスト X でサンプルデータを格納していく。

```
import numpy as np

time_step = 100
y = Lorenz63(time_step = time_step)

burn_in = 1000
num = 10000

x0 = np.random.normal(0., 1., size = 3) + np.array([0.5, 0., 0.])
X0 = []; X0.append(x0)
```

メトロポリス法のため、バーンイン期間と所望のサンプル数だけの繰り返し計算を行っていく。下記コード中の x0_next はメトロポリス法における更新点である。また、Hx0(Hx0_next) は初期位置が x0(x0_next) のときの Lorenz63 モデルである。

```
for itr in range(burn_in+num):
        x0_next = x0 + 0.2*(np.random.rand(3)—0.5)

        Hx0 = Lorenz63(x0[0], x0[1], x0[2], noise = False, time_step = time_step)
        Hx0_next = Lorenz63(x0_next[0],
                x0_next[1], x0_next[2], noise = False, time_step = time_step)
```

次に事後分布の比率を計算する。まず尤度の比率は下記の関数 ratioLik より求めた。なお、観測ノイズは $\mathcal{N}(\boldsymbol{0}, 4I)$ である。また、式 (39) より $1/(2\pi)^{3/2}/|\Sigma|^{1/2}$ は両者で共通するため、尤度の比率計算では $\exp(\cdot)$ のみ考慮すればよい。また、尤度が非常に小さい場合、丸め誤差が問題になってくる。そのため、下記関数のようにタイムステップ毎に尤度の比を計算することで、丸め誤差を防いだ。

```
def ratioLik(y, Hx, Hx_next):
        r = 1.
        for iy, iHx, iHx_next in zip(y, Hx, Hx_next):
                Lx = np.exp(—np.sum((iy—iHx)**2)/8.)
                Lx_next = np.exp(—np.sum((iy—iHx_next)**2)/8.)

                r *= Lx_next/Lx

        return r
```

尤度の比を先ほどの for ループの中で受け取る (コード中 rL)。また、事前分布の比 (rP) は正規分布の比より求まる。ただし、こちらも $1/(2\pi)^{3/2}$ は共通すること、並びに事前分布中の x、y、z は独立であることから、このような書き方になっている。両者の積より事後分布の比 r を求めた。

```
for itr in range(burn_in+num):
        rL = ratioLik(y, Hx0, Hx0_next)
        rP = np.exp(—((x0_next[0]—0.5)**2+x0_next[1]**2+x0_next[2]**2) / 2) / \
                np.exp(—((x0[0]—0.5)**2+x0[1]**2+x0[2]**2) / 2)
        r = rL*rP
```

最後にメトロポリス法に従いサンプリングを行う。サンプリング過程及びこれまでのコードを纏めたものを以下に示す。

```
import numpy as np

def ratioLik(y, Hx, Hx_next):
        r = 1.
        for iy, iHx, iHx_next in zip(y, Hx, Hx_next):
                Lx = np.exp(-np.sum((iy-iHx)**2)/8.)
                Lx_next = np.exp(-np.sum((iy-iHx_next)**2)/8.)

                r *= Lx_next/Lx

        return r

time_step = 100
y = Lorenz63(time_step = time_step)

burn_in = 1000
num = 10000

x0 = np.random.normal(0., 1., size = 3) + np.array([0.5, 0., 0.])
X0 = []; X0.append(x0)

for itr in range(burn_in+num):
        x0_next = x0 + 0.2*(np.random.rand(3)-0.5)

        Hx0 = Lorenz63(x0[0], x0[1], x0[2], noise = False, time_step = time_step)
        Hx0_next = Lorenz63(x0_next[0],
                                x0_next[1], x0_next[2], noise = False, time_step = time_step)

        rL = ratioLik(y, Hx0, Hx0_next)
        rP = np.exp(-((x0_next[0]-0.5)**2+x0_next[1]**2+x0_next[2]**2) / 2) / \
                np.exp(-((x0[0]-0.5)**2+x0[1]**2+x0[2]**2) / 2)
        r = rL*rP

        if r < 1.:
                if np.random.rand() < (1.-r):
                        X0.append(x0_next)
                else:
                        X0.append(x0)
        else:
                X0.append(x0_next)

        x0 = X0[-1]
```

本コードを実行したところ、$\boldsymbol{x}_0$ の各要素の分布は図 15 のようになった (各要素毎にヒストグラムを作成しているので、これは $\boldsymbol{x}_0$ の同時確率分布ではなく各要素の周辺確率を見ていることに注意)。正解がそれぞれ (1, 0, 0) であることを考えるとあまり良くないかもしれない。ただし Lorenz63 モデルは初期値にかなり敏感なので、非逐次型データ同化で解こうとすると、こんなものだと思われる。

6 逐次型データ同化

6.1 カルマンフィルタ (KF)

本章から逐次型データ同化に進む。逐次型データ同化で基礎にあたるのはカルマンフィルタ (KF) であろう。

私たちは 2.3.3 節でベイズ機械学習の更新アルゴリズム的観点を学んだ。式 (33)(34) のように計算がシンプルになった理由は、事前分布と尤度を正規分布で統一したこと、並びにパラメータに対して線形変換しか施さなかったことにある。KF もこの特性を存分に利用する。

まず 4.2.2 節の $p(\boldsymbol{x}_{t-1})$ と尤度、並びにシステムノイズ $\boldsymbol{v}_t$ も正規分布と仮定する。また、それだけではなくシステムモデルも線形変換であると仮定する。そのため、式 (37) の代わりに行列 F を用いて

$$\boldsymbol{x}_t = F\boldsymbol{x}_{t-1} + \boldsymbol{v}_t$$

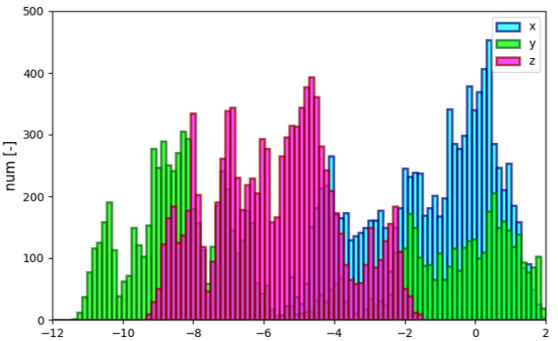

図 15　メトロポリス法でサンプリングされた $\boldsymbol{x}_0$ の事後分布。

なるシステムモデルを考える。$\boldsymbol{x}_{t-1}$ は正規分布に従うので $\boldsymbol{x}_t$ も正規分布に従う (正規分布の線形変換)。システムノイズの確率分布を $\mathcal{N}(\boldsymbol{0}, V)$、$p(\boldsymbol{x}_{t-1})$ を $\mathcal{N}(\boldsymbol{\mu}_{t-1}, S_{t-1})$ としたとき、$p(\boldsymbol{x}_t)$ は

$$p(\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, S_t), \quad \boldsymbol{\mu}_t = F\boldsymbol{\mu}_{t-1}, \quad S_t = FS_{t-1}F^{\mathrm{T}} + V \tag{42}$$

となる。

KF はこれだけでなく、観測モデルに対しても線形であることを仮定する (4.1.2 節の例 1 と 2 が線型な観測モデルに当てはまる)。つまり式 (38) の代わりに、行列 H を用いた

$$\boldsymbol{y}_t = H\boldsymbol{x}_t + \boldsymbol{w}_t$$

なる観測モデルを考える。導出は省略するが、このときの事後分布 $p(\boldsymbol{x}_t|\boldsymbol{y}_t)$ も正規分布になる。観測ノイズ $\boldsymbol{w}_t$ が $\mathcal{N}(\boldsymbol{0}, \Sigma)$ に従うとしたとき、$p(\boldsymbol{x}_t|\boldsymbol{y}_t)$ は

$$p(\boldsymbol{x}_t|\boldsymbol{y}_t) = \mathcal{N}(\boldsymbol{\mu}'_t, S'_t), \quad \mu'_t = \mu_t + K_t(\boldsymbol{y}_t - H\boldsymbol{\mu}_t), \quad S'_t = (I - K_tH)S_t \tag{43}$$

となる。ここで K_t はカルマンゲインと言い、

$$K_t = S_tH^{\mathrm{T}}(HS_tH^{\mathrm{T}} + \Sigma)^{-1} \tag{44}$$

である。

このように KF は式 (42)(43) の繰り返し計算をするだけで学習できるため、計算コストが少なく済む。しかしながら、CAE を用いたデータ同化ではあまり用いられない。線型なシステムモデルという仮定はときに大胆すぎる。非線形な流体現象の場合、線型なサロゲートモデルがない限り KF の利用は難しい。

ならば音響などの線型問題ならばどうかと言うと、これも難しいときがある。KF が線型な物理現象ではなく、ベクトル $\boldsymbol{x}_t$ に対する線型性を仮定していることに注意してほしい。4.1.1 で示した通り、この $\boldsymbol{x}$ には物理場 $\boldsymbol{u}$ だけでなくパラメータ $\boldsymbol{w}$ も含まれている。音響解析における $\boldsymbol{w}$ として音速や壁面インピーダンスなどが挙げられるが、それらはシステムモデルの行列 F の中にも含まれている。このような場合は $\boldsymbol{x}$ から見た音響解析は非線型であり、KF の仮定を満たさない。

以上の理由から、CAE を用いたデータ同化で KF を利用することは稀である。

6.2　アンサンブルカルマンフィルタ (EnKF)

KF の考え方を踏襲しつつ、非線型なシステムモデルに対応可能な手法としてアンサンブルカルマンフィルタ (EnKF) がある。EnKF でも確率分布は正規分布しか現れない。非線型なシステムモデル $\boldsymbol{x}_t = F(\boldsymbol{x}_{t-1}) + \boldsymbol{v}_t$ を考える場合、たとえ $p(\boldsymbol{x}_{t-1})$ が正規分布に従うとしても $p(\boldsymbol{x}_t)$ は正規分布に従わない。しかしながら、EnKF は近似的に正規分布であることを維持しようとする。

まず初期時刻の $\boldsymbol{x}_0$ に関して、事前分布 $p(\boldsymbol{x}_0)$ に従い N 個サンプリングする。このときのアンサンブルを $\{\boldsymbol{x}_0^i | i = 1, ..., N\}$ で表す。次に各粒子に対してシステムモデルを施し、

$$\boldsymbol{x}_1^i = F(\boldsymbol{x}_0^i) + \boldsymbol{v}_t$$

を得る。このアンサンブルは $p(\boldsymbol{x}_t)$ をよく表していると考えられるだろう。そこで、$p(\boldsymbol{x}_t)$ を正規分布だと仮定し、その分散共分散行列 S_1 を

$$S_1 = \frac{1}{N-1}\sum_i (\boldsymbol{x}_1^i - \boldsymbol{\mu}_1)(\boldsymbol{x}_1^i - \boldsymbol{\mu}_1)^{\mathrm{T}}$$

より推定する。ここで $\mu_1 = \sum_i \boldsymbol{x}_1^i/N$ である。なお上式が N ではなく $N-1$ で割っている理由は、これが分散共分散行列の不偏推定量になるためである (詳しくは統計学の書籍を参照のこと)。最後に、式 (43) に倣って、各粒子を

$$\boldsymbol{x}_1^i \leftarrow \boldsymbol{x}_1^i + K_1(\boldsymbol{y}_1 - H\boldsymbol{x}_1^i)$$

で更新する。ここで K_1 はカルマンゲインであり、式 (44) に倣って

$$K_1 = S_1 H^{\mathrm{T}} (H S_1 H^{\mathrm{T}} + \Sigma)^{-1}$$

と定義する。

最終的に得られたアンサンブル $\{\boldsymbol{x}_1^i | i = 1, ..., N\}$ は正に事後分布、つまり $p(\boldsymbol{x}_1|\boldsymbol{y}_1)$ をよく表している。これを繰り返すことで EnKF は学習していく。以下に EnKF のアルゴリズムを記す。

アンサンブルカルマンフィルタ

1. 事前分布に従いアンサンブル $\{\boldsymbol{x}^i | i = 1, ..., N\}$ を生成する。
2. $\boldsymbol{x}^i \leftarrow F(\boldsymbol{x}^i) + \boldsymbol{v}_0$。
3. アンサンブルから分散共分散行列

$$S = \frac{1}{N-1} \sum_i (\boldsymbol{x}^i - \boldsymbol{\mu})(\boldsymbol{x}^i - \boldsymbol{\mu})^{\mathrm{T}} \tag{45}$$

   を計算する。ここで $\boldsymbol{\mu} = \sum_i \boldsymbol{x}^i / N$。
4. 各粒子を尤度計算に従い、

$$\boldsymbol{x}^i \leftarrow \boldsymbol{x}^i + K(\boldsymbol{y} - H\boldsymbol{x}^i) \tag{46}$$

   で更新する。ここで K は

$$K = S H^{\mathrm{T}} (H S H^{\mathrm{T}} + \Sigma)^{-1} \tag{47}$$

   であり、カルマンゲインと言う。
5. 2.-4. を繰り返す。

非線型なシステムモデルを扱えるようになったが、KF のときと変わらず計算が容易なところも見られる。ただし、各粒子の更新時に粒子の数 N だけのシステムモデル、つまり CAE 解析が必要になる。当然ながら N の数が少ないとアンサンブルによる確率分布の信憑性が低くなる。計算コストと精度を鑑みて粒子数を決めなければならない。

6.2.1 python コード例

EnKF による Lorenz63 モデルのデータ同化を試みる。観測ノイズはこれまで通り $\mathcal{N}(\boldsymbol{0}, 4I)$ とした。従って $\Sigma = 4I$ である。また、観測値 $\boldsymbol{y}$ と状態ベクトル $\boldsymbol{x}$ は同じものなので、$H = I$ となる。以上よりカルマンゲインは $K = S(S + 4I)^{-1}$ となる。

今回は 400 タイムステップまでの解析を行う。これまで同様学習データ y を前もって生成しておいた。また、EnKF の粒子数 N は 100 とした。x は N 行 3 列の配列であり、アンサンブルを表す。今回は事前分布を $\mathcal{N}((-2, 0, 0)^{\mathrm{T}}, I)$ とし、それに従うようサンプリングした。

```
import numpy as np

time_step = 400
y = Lorenz63(time_step = time_step)
R = 4.*np.eye(3)

N = 100
x = np.random.normal(0., 1., size = (N, 3)) + np.array([-2., 0., 0.])
```

アンサンブル x から次時刻の状態に更新するシステムモデルを step という関数で定義した。下記の通り、粒子毎に Lorenz63 の解析をしている。ただし逐次的データ同化なので一度に 400 タイムステップの解析をしていない点に注意してほしい。解析を途中で止め、そのときの事後分布を計算できるようにしている。なお、今回はシステムノイズを標準正規分布としている。

```
def step(x):
        x_next = []
        for ix in x:
                x_n = Lorenz63(ix[0], ix[1], ix[2], noise = False, time_step = 1)[0]
                x_next.append(x_n)

        x_next = np.stack(x_next, axis = 0)
        noise = np.random.normal(0., 1., size = x_next.shape)
        x_next += noise

        return x_next
```

下記の通り、400 タイムステップまでの繰り返し計算により解析を逐次的に行っている。x_next は関数 step の出力であり、次時刻のアンサンブルである。今回は 10 タイムステップ毎にデータ同化を行うことにした。`if (t+1)%10 == 0:` とあるように、10 タイムステップの周期で EnKF の処理を行っている。

```
for t in range(time_step):
        x_next = step(x)

        if (t+1)%10 == 0:
                mu = np.mean(x_next, axis = 0)
                Sigma = np.cov(x_next-mu, rowvar = False)

                K = Sigma@np.linalg.inv(Sigma+R)
                x_next += (y[t]-x_next)@K

        x = x_next
```

これまでのコードを纏めたものを以下に示す。

```
import numpy as np

time_step = 400
y = Lorenz63(time_step = time_step)
R = 4.*np.eye(3)

N = 100
x = np.random.normal(0., 1., size = (N, 3)) + np.array([-2., 0., 0.])

def step(x):
        x_next = []
        for ix in x:
                x_n = Lorenz63(ix[0], ix[1], ix[2], noise = False, time_step = 1)[0]
                x_next.append(x_n)

        x_next = np.stack(x_next, axis = 0)
        noise = np.random.normal(0., 1., size = x_next.shape)
        x_next += noise

        return x_next

for t in range(time_step):
        x_next = step(x)

        if (t+1)%10 == 0:
                mu = np.mean(x_next, axis = 0)
                Sigma = np.cov(x_next-mu, rowvar = False)

                K = Sigma@np.linalg.inv(Sigma+R)
                x_next += (y[t]-x_next)@K

        x = x_next
```

Lorenz63 モデルの数値 x について、観測値 y とアンサンブルの平均値を比較した (図 16)。観測データに対して学習結果の平均値はよく一致している。この結果は喜ばしいことだが、平均値が不自然ながたつきを見せていることに注意してほしい。4.2 節で議論した通り、逐次型データ同化は現時刻 (もしくは最終時刻) の尤もらしさにしか興味がない。尤度を高めるためなら途中時刻であっても状態ベクトルの不自然な (支配方程式に従わない) 変化も認めている。そのため学習後に過去を含めた時系列を確かめたとき、このようながたつきが見られる。このようなことは非逐次型データ同化の結果 (図 12) では見られなかった。これは全時刻を俯瞰する非逐次型データ同化の長所と言える。個人的には、オンライン学習の場合は逐次型データ同化の方が良いが、物性値推定のような逆解析的にデータ同化を用いたい場合は非逐次型データ同化を選ぶべきな気がしている。

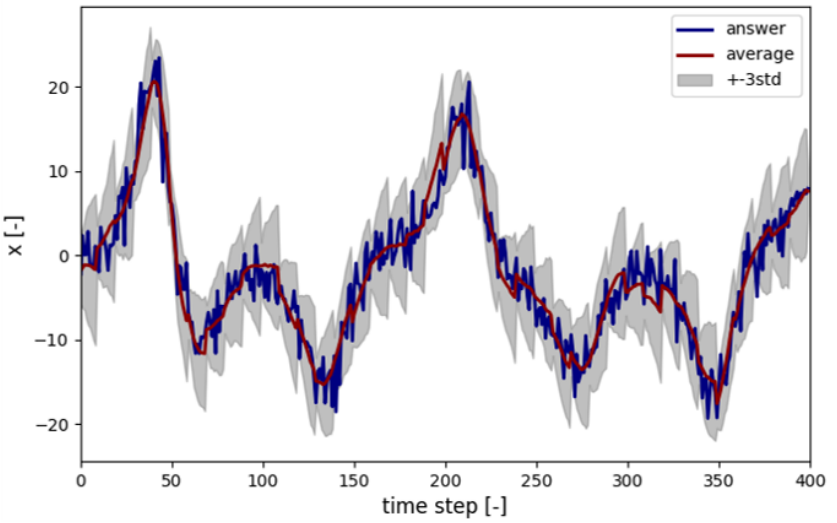

図 16　EnKF の結果。Lorenz63 モデルの数値 x に対し、(answer) 観測データ、(average) アンサンブルの平均値、(+-std) 平均値 $\pm 3\sigma$。σ はアンサンブルにおける x の標準偏差。

6.3　粒子フィルタ (PF)

EnKF にすることで非線形なシステムモデルも扱えるようになった。しかしながら、相変わらず正規分布しか扱えないことと、線型な観測モデルという仮定が残っている。本節で紹介する粒子フィルタ (PF) はこれらの制約を受けない、非常に自由度の高い手法と言える。

粒子という言葉から想像できるように、PF は式 (22) のアンサンブル表現によるノンパラメトリックベイズ機械学習である。以下は PF のアルゴリズムである。

粒子フィルタ

1. 所望の事前分布 $p(\boldsymbol{x})$ に従って粒子をサンプリングする。$\{\boldsymbol{x}_i | i = 1, ..., N\}$。
2. $\boldsymbol{x}_i \leftarrow F(\boldsymbol{x}_i) + \boldsymbol{v}$。
3. 各粒子に対して尤度

$$\beta_i = p(\boldsymbol{y}|\boldsymbol{x}_i) = \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y} - H(\boldsymbol{x}_i))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y} - H(\boldsymbol{x}_i))\right)$$

   を計算する。
4. 各粒子の事前確率は $1/N$ であるため、事後分布は $C\beta_i/N$ となる。ここで C は規格化係数だが、確率の総和が 1 であることから $C = N/\sum_i \beta_i$ であることが分かる。従って、各粒子の事後確率は $\beta_i/\sum_j \beta_j$ と求まる。
5. 各粒子を $\beta_i/\sum_j \beta_j$ の確率で再サンプリングし、事後分布に従うアンサンブルを得る。
6. 2.-5. を繰り返す。

上記の通り、PF は 3.2 節のベイズ機械学習と類似している。3.2 節で見られた多様性維持のためのノイズは、システムノイズが担っている。そのため、PF ではシステムノイズが重要になってくる。

EnKF もアンサンブルを利用するが、確率分布はいつも正規分布であったことを思い出してほしい。一方の PF はノンパラメトリック確率分布であるため、KF のときからあった制約が全てなくなっている。ただし、ノンパラメトリックである分、EnKF のときより多くのサンプル数が必要になる (少ないサンプル数だと確率分布が十分に表現できないことになる)。サンプル数の数だけシステムモデルの処理、つまり CAE 解析が必要になってくるため、PF の計算コストは高い。一回当たりの解析時間が短く、低次元な状態モデルの問題に PF は向いている。

6.3.1　python コード例

PF を用いて Lorenz63 モデルのデータ同化を行う。基本的に EnKF のときと条件は同じだが、粒子数だけ 500 に変えた。

それ以外の EnKF のコードとの差分は下記の事後分布計算の所のみである。コード中 beta は上記アルゴリズムの β_i の配列に相当する。ただし、これまでと同様に $1/(2\pi)^{3/2}/|\Sigma|^{1/2}$ の部分は計算していない。prob は各粒子の事後確率である。`np.random.choice(N, N, p=prob)` でサンプリングする粒子インデックスを決定し、次の行で実際にサンプリングしている。

```
for t in range(time_step):
        if (t+1)%10 == 0:
```

```
                beta = np.exp(-np.sum((y[t]-x_next)**2, axis = 1)/8.)
                prob = beta/np.sum(beta)

                index = np.random.choice(N, N, p=prob)
                x_next = x_next[index]
```

EnKF と共通している部分も含めたコードを以下に記す。また、Lorenz63 モデルの数値 x について、観測値 y とアンサンブルの平均値を比較した (図 17)。

```
import numpy as np

def step(x):
        x_next = []
        for ix in x:
                x_n = Lorenz63(ix[0], ix[1], ix[2], noise = False, time_step = 1)[0]
                x_next.append(x_n)

        x_next = np.stack(x_next, axis = 0)
        noise = np.random.normal(0., 1., size = x_next.shape)
        x_next += noise

        return x_next

time_step = 400
y = Lorenz63(time_step = time_step)
R = 4.*np.eye(3)

N = 500
x = np.random.normal(0., 1., size = (N, 3)) + np.array([-2., 0., 0.])

for t in range(time_step):
        x_next = step(x)

        if (t+1)%10 == 0:
                beta = np.exp(-np.sum((y[t]-x_next)**2, axis = 1)/8.)
                prob = beta/np.sum(beta)

                index = np.random.choice(N, N, p=prob)
                x_next = x_next[index]

        x = x_next
```

6.4 融合粒子フィルタ (MPF)

PF は自由度が高い一方で、多様性の消滅 (つまり退化) が問題になることが多い。退化の対策方法は今でも熱心に研究されており、様々な PF の派生形が提案されている。本資料ではそのうちの一つである融合粒子フィルタ (MPF) を紹介する。

お気付きになられた方もいるかと思うが、PF のアルゴリズムは進化計算と非常に似ている。前節のアルゴリズムの 5. は、進化計算では世代交代と呼ばれている。また、システムノイズは突然変異に相当する。進化計算では突然変異の他に交叉がよく施されるが、MPF は PF に交叉の機能を加えたものと言える。

原著では、粒子のアンサンブル $\{\boldsymbol{x}_i | i = 1, ..., N\}$ から事後確率 $\beta_i / \sum_i \beta_i$ に従い 3 つの粒子 $\{\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3\}$ をサンプリングする。そしてこれらの重み付き和

$$\boldsymbol{x}'_1 = a_1 \boldsymbol{x}^1 + a_2 \boldsymbol{x}^2 + a_3 \boldsymbol{x}^3$$

を新しい粒子として考える。この $\boldsymbol{x}$ は $\{\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3\}$ と異なる値なので、多様性の増加 (もしくは維持) ができそうである。この処理を N 回繰り返し、新しくできた粒子のアンサンブル $\{\boldsymbol{x}'_i | i = 1, ..., N\}$ を事後分布として扱う。以下は MPF のアルゴリズムである。

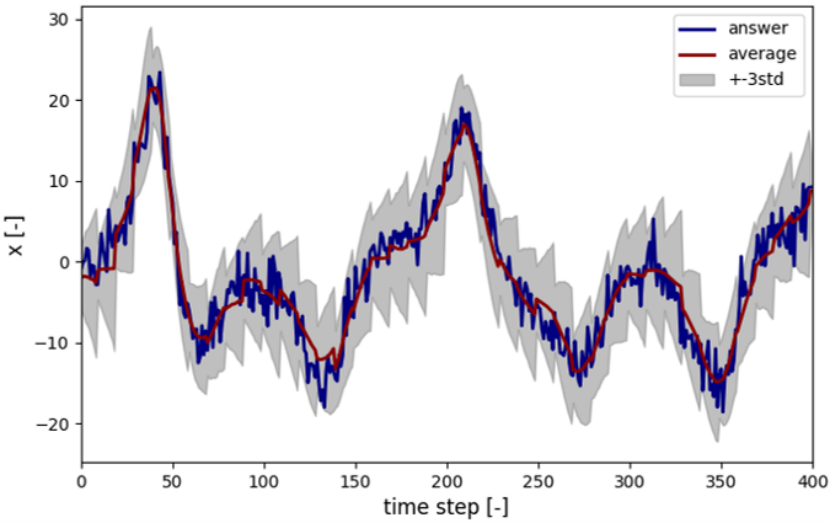

図 17　PF の学習結果。Lorenz63 モデルの数値 x に対し (answer) 観測データ、(average) アンサンブルの平均値、(+-std) 平均値 $\pm 3\sigma$。σ はアンサンブルにおける x の標準偏差。

融合粒子フィルタ

1. 所望の事前分布 $p(\boldsymbol{x})$ に従って粒子をサンプリングする。$\{\boldsymbol{x}_i | i = 1, ..., N\}$。
2. $\boldsymbol{x}_i \leftarrow F(\boldsymbol{x}_i) + \boldsymbol{v}$。
3. 各粒子に対して尤度

$$\beta_i = p(\boldsymbol{y}|\boldsymbol{x}_i) = \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y} - H(\boldsymbol{x}_i))^{\mathrm{T}}\Sigma^{-1}(\boldsymbol{y} - H(\boldsymbol{x}_i))\right)$$

   を計算する。
4. 各粒子の事前確率は $1/N$ であるため、事後分布は $C\beta_i/N$ となる。ここで C は規格化係数だが、確率の総和が 1 であることから $C = N/\sum_i \beta_i$ であることが分かる。従って、各粒子の事後確率は $\beta_i/\sum_j \beta_j$ と求まる。
5. 各粒子を $\beta_i/\sum_j \beta_j$ の確率で N 個だけ再サンプリングする。
6. 5. のアンサンブルから粒子を 3 個サンプリングする。サンプリングされた粒子 $\{\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3\}$ から、新しい粒子

$$\boldsymbol{x} = a_1\boldsymbol{x}^1 + a_2\boldsymbol{x}^2 + a_3\boldsymbol{x}^3 \tag{48}$$

   を生成する。これを N 回繰り返し、新しいアンサンブルを得る。
7. 2.-6. を繰り返す。

式 (48) の係数 a_i は何でもよい訳ではない。たとえば $a_1 = a_2 = a_3 = 0$ の場合、新しい粒子は常に $\mathbf{0}$ になってしまう。これは大げさな例だが、原著では 2 つの制約を提言している。

アルゴリズム中 5. で得られたアンサンブルは事後分布に従っている。事後分布はベイズ機械学習に則って求められたので、それを式 (48) で崩してしまうのはよくない。例えば正規分布は和や線型変換で平均や分散が変わってしまう。このようなことが式 (48) でも起こっていると考えられる。

しかしながら、原著によると、確率分布の期待値、分散、尖度などあらゆる代表値を維持するように、係数 a_i を設定することは不可能らしい。そこで、少なくとも期待値と分散は式 (48) から得られたアンサンブルでも維持されるよう、

$$a_1 + a_2 + a_3 = 1, \quad a_1^2 + a_2^2 + a_3^2 = 1 \tag{49}$$

なる制約式を提言している。

6.4.1　python コード例

MPF で Lorenz63 モデルのデータ同化を行う。係数は

$$a_1 = \frac{2}{3}, \quad a_2 = \frac{2}{3}, \quad a_3 = -\frac{1}{3}$$

とした。MPF と PF の差分はサンプリング部分のみである。PF と同様に事後確率 prob を求め、N 個のサンプリングを 3 種用意する。それぞれのサンプリング結果を a_i の重み付き和で計算すれば、アルゴリズムの 6. が実行されたことになる。

```
for t in range(time_step):
```

```
        if (t+1)%10 == 0:
                index_1 = np.random.choice(N, N, p=prob)
                index_2 = np.random.choice(N, N, p=prob)
                index_3 = np.random.choice(N, N, p=prob)

                x_next = (2./3.)*x_next[index_1] \
                         + (2./3.)*x_next[index_2] \
                         - (1./3.)*x_next[index_3]
```

以下は MPF の全コードである (今回の問題だと、PF とほとんど結果は変わらなかった)。

```
import numpy as np


def step(x):
        x_next = []
        for ix in x:
                x_n = Lorenz63(ix[0], ix[1], ix[2], noise = False, time_step = 1)[0]
                x_next.append(x_n)

        x_next = np.stack(x_next, axis = 0)
        noise = np.random.normal(0., 1., size = x_next.shape)
        x_next += noise

        return x_next

time_step = 400
y = Lorenz63(time_step = time_step)
R = 4.*np.eye(3)

N = 500
x = np.random.normal(0., 1., size = (N, 3)) + np.array([-2., 0., 0.])

for t in range(time_step):
        x_next = step(x)

        if (t+1)%10 == 0:
                beta = np.exp(-np.sum((y[t]-x_next)**2, axis = 1)/8.)
                prob = beta/np.sum(beta)

                index_1 = np.random.choice(N, N, p=prob)
                index_2 = np.random.choice(N, N, p=prob)
                index_3 = np.random.choice(N, N, p=prob)

                x_next = (2./3.)*x_next[index_1] \
                         + (2./3.)*x_next[index_2] \
                         - (1./3.)*x_next[index_3]

        x = x_next
```

さいごに (参考図書紹介)

「統計学入門」東京大学教養学教室 (編)
ベイズ統計ではない、いわゆる一般的な統計学に関する本。序文から内容が素晴らしく、考え方が参考になる。
「パターン認識と機械学習」C.M. ビショップ (著)
頻度主義とベイズを上手く比較した名著。3 章まで読むとよい。
「データ同化流体科学」大林茂ら (著)
データ同化を広く扱った書籍。タイトルに流体とあるが、他分野の人にとっても役に立つ。
「データ同化-観測・実験とモデルを融合するイノベーション-」淡路敏之ら (著)
冒頭から説明がかなり不親切。購入したことを後悔。
「データ同化入門」樋口知之ら (著)
逐次型データ同化に特化した書籍。内容も濃く参考になった。本資料では取り扱っていない平滑化も説明している。