Pythonによる出生データの数理データサイエンス入門

東京理科大学 理学部第一部 応用数学科 教授　橋口（はしぐち）　博樹（ひろき）

はじめに

筆者の所属する応用数学科（数理情報科学科時代を含め）では，伝統的に統計学や最適化法，数値解析の研究・教育に力を入れており，データサイエンスの数理を学ぶ，あるいは，研究するといった土壌は昔からありました。本稿では，人口動態統計の出生に関するデータを解析対象として，データサイエンスの数理「数学の知恵を理解し，使う」ということと，この解析に関するPythonによる実装方法を２つの題材を通して解説したいと思います。

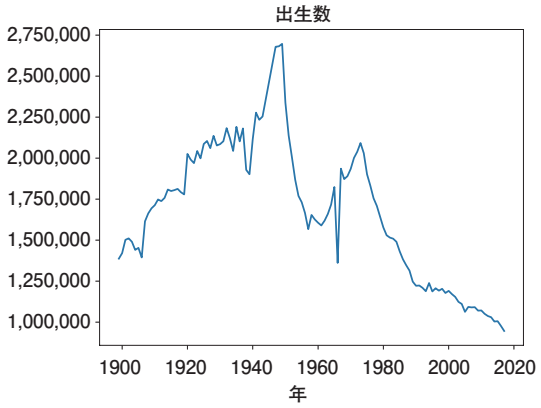

図１　出生数の推移（厚生労働省人口動態統計より）

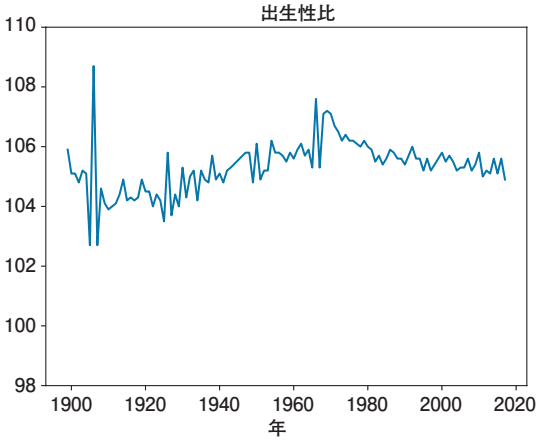

図２　出生性比の推移（厚生労働省人口動態統計より）

男女は同じ割合で生まれるのか？

厚生労働省の人口動態統計のホームページからデータをダウンロードし，Pyhonでデータを読み込めるように加工して，出生数の経年変化を折れ線で示したものが**図１**です。データが取られている1899年から2017年までの間でのピーク（最頻値）は1949年であって，200万人を超えていました。出生数の減少傾向には歯止めがかからず，2016年にはとうとう100万人を下回り，ピーク時の半分以下です。2017年も100万人を下回っています。

図２を見ると出生性比は，だいたい105程度になっています。平均値は105.3で標準偏差は0.88です。この出生性比は，女子100人に対して男子が何人生まれるかという指標で，100ならば五分五分で，男女の割合は0.5となります。105ということは，男子が女子より５％ほど生まれやすいということになり，男子の生まれる割合も同様に105/(100＋105)＝0.512となり，0.5より少し多くなります。2017年の場合，94万6,065人生まれて，男子が48万4,449人，女子が46万1,616で2万3,000人ほど男子が多くなっています。出生性比は104.9です。この男子が多く生まれるという結果は，統計的に有意な差があること

が，統計的検定を行うことで結論づけられます。

この検定方法の原理は，次のような考え方です。公平なコイン（表が0.5の確率で，裏も0.5の確率で出る）を94万6,065回投げて表（男子）が出る数を数えたとします。再び投げて表の数を数えます。94万6,065回投げて表を数えることを100回繰り返したとして，表が出る回数が48万4,449回以上となることは，公平なコインでは100回中１回もありません。94万6,065回中で48万4,449回表が出たのは，そもそもコインが公平ではなかったと結論づけます。同様に男女の生まれる割合も0.5ではなく，男子の方が多く生まれるということが統計的に結論づけられます。

誕生日問題

誕生日問題は，「あるグループの中に，同じ誕生日をもつ人が２人以上存在する確率が50％を超えるためには，グループに少なくとも何人必要であるか？」という問題です。この問題は，以下のように解かれます。グループの人数をk人として，k人の中で同じ誕生日をもつ人が少なくとも２人以上存在する確率を$p(k)$とします。また，その余事象である，k人すべてが異なる誕生日をもつ確率を$\overline{p}(k)$とします。誕生日問題は，$p(k)$が0.5を初めて超えるときのkの値を見つければよいわけです。まず$\overline{p}(k)$を計算してみます。２人目が１人目と異なる誕生日である確率は364/365です。次に，３人目が１人目２人目と異なる誕生日である確率は363/365です。これを続けていくとk人目は$(365-k+1)$/365となります。まとめると，$\overline{p}(k)$は

$$\overline{p}(k)=\frac{364}{365}\cdot\frac{363}{365}\cdot\frac{362}{365}\cdots\frac{365-k+1}{365}$$

$$=\frac{365!}{365^k(365-k)!} \qquad (1)$$

となり，したがって，$p(k)$は

$$p(k)=1-\overline{p}(k)=1-\frac{365!}{365^k(365-k)!} \qquad (2)$$

となります。kが大きくなるにつれて$p(k)$は大きくなる，つまり，単調増加であることは容易に分かりますので，$p(1), p(2), \ldots$と順番に計算していき，$p(22)\approx 0.476$と$p(23)\approx 0.507$となる$k=23$が，誕生日問題の解となります。さらに，$k=70$では，$p(70)\approx 0.999$となり，70人程度で，同じ誕生日の人が２人以上存在する確率が99.9％を超えることが分かります。

拡張誕生日問題

前節で扱った誕生日問題では，人はどの日でも1/356の確率で生まれること，つまり，誕生日の分布が一様であることを前提としています。でも実際のところは，誕生日の分布は一様ではないので，この場合の誕生日問題を考えてみたいと思います。まず，k人の中で同じ誕生日をもつ人が少なくとも２人以上存在する事象をA_kとし，k人すべてが異なる誕生日をもつ事象を$\overline{A}_k$とします。また，$n=365$とし，何月何日を単にjの記号で表し，$j=1,\ldots, n$とします。j日が誕生日である確率をp_j，つまり，p_jはjが誕生日の人の割合とし，それらを集めたベクトルを

$$\boldsymbol{p}=(p_1, p_2, \ldots, p_j, \ldots, p_n)$$

とします。このとき，$S_{n,k}(\boldsymbol{p})$を

$$S_{n,k}(\boldsymbol{p})=\sum_{1\le i_1<\ldots<i_k\le n} p_{i_1}\cdots p_{i_k}$$

として，分布$\boldsymbol{p}$のもとで，k人すべてが異なる誕生日をもつ確率は

$$P(\overline{A}_k;\boldsymbol{p})=k!S_{n,k}(\boldsymbol{p}) \qquad (3)$$

と表すことができます。$S_{n,k}(\boldsymbol{p})$は$(1+p_1t)\cdot(1+p_2t)\cdots(1+p_{k-1}t)\cdot(1+p_nt)$の展開における$t^k$の係数にもなります。通常の誕生日問題では，すべてのjに対して$p_j=1/n=1/365$であり，

$$P\{\overline{A}_k;(1/n,\ldots,1/n)\}=k!\sum_{1\le i_1<\ldots<i_k\le n}(1/n)^k$$

表1　各年の日別出生率のもとでの$P(A_k;\mathbf{p})$の値

k	2005年	2010年	2015年	一様分布
1	0	0	0	0
2	0.00280	0.00281	0.00282	0.00274
22	0.48259	0.48397	0.48585	0.47570
23	0.51439	0.51580	0.51774	0.50730
68	0.99888	0.99891	0.99895	0.99873
69	0.99910	0.99912	0.99915	0.99896
70	0.99927	0.99929	0.99932	0.99916

$$= k!\frac{{}_nC_k}{n^k} = \overline{p}(k) \qquad (4)$$

となり，$\overline{p}(k)$と一致することが分かります。しかし，$(p_1, ..., p_n)$が同じでないような一般の場合には，$k=23$とすると，和$\sum_{1\le i_1<\ldots<i_k\le n}$を取る組合せの総数は，${}_{365}C_{23}\approx 1.632\times 10^{36}$であり，すべてを列挙することは到底不可能です。仮に1秒間に100万個列挙したとしても10^{22}年以上かかり，この年数は地球誕生から現在までの年数よりはるかに長い年数です。そこで，${}_nC_k = {}_{n-1}C_k + {}_{n-1}C_{k-1}$が成り立つことの組合せ論的根拠，つまり，特定の一人に着目してその人を含む含まないかで，この漸化式が成り立つことと同様の考え方によって，$S_{n,k}(\boldsymbol{p})$は

$$S_{n,k}(\boldsymbol{p}) = S_{n-1,k}(p_1, ..., p_{n-1}) + p_n S_{n-1,k-1}(p_1, ..., p_{n-1})$$

の漸化式を満たします。ただし，初期値は，$S_{j,1}(p_1, ..., p_j) = \sum_{i=1}^{j} p_i$，$S_{j,j} = p_1\cdots p_j$です。このことと(3)から，$P\{\overline{A_k};\boldsymbol{p}\}$は

$$P\{\overline{A_k};\boldsymbol{p}\} = kP\{\overline{A_k};(p_1, ..., p_{n-1})\} + kp_nP\{\overline{A_k};(p_1, ..., p_{n-1})\} \qquad (5)$$

の漸化式を満たし，初期値は

$$P\{\overline{A_k};(p_1, ..., p_j)\} = \sum_{i=1}^{j} p_i,$$

$$P\{\overline{A_k};(p_1, ..., p_j)\} = j!p_1\cdots p_j = jp_jP\{\overline{A_k};(p_1, ..., p_{j-1})\} \qquad (6)$$

となります。人口動態統計をもとに，日本における2005年，2010年，2015年の出生分布のもとで$P(A_k;\mathbf{p}) = 1 - P(\overline{A_k};\mathbf{p})$の値を計算すると，**表1**のようになります。一様分布のときと同じ$k=23$が答えとなることが分かります。

Pythonによる出生データの解析と誕生日問題の解の計算

出生データの解析のファイルの読み込みや最頻値計算などの基本的な分析はpandasを使いました。また，図1の描画にはmatplotlibを使いました。pandas，matplotlibの解説は書籍やWebページで多く見られるので，そちらを参照してください。

(1)式をみると，$\overline{p}(k)$と$\overline{p}(k-1)$の漸化式が

$$\overline{p}(k) = \left(1 - \frac{k-1}{365}\right)\overline{p}(k-1), k \ge 2$$

$\overline{p}(1) = 1$となることが分かります。この漸化式をPythonでコーディングすると

```
def pbar(k):
    pi=1
    for  i in range(1,k):
        pi *=(1-i/365)
    return pi

def pbirthday(k):
    return 1-pbar(k)
if __name__ == '__main__':
    for k in range(1,100):
        print(k, round(pbirthday
        (k),3))
```

のようになります。

拡張誕生日問題の計算(5)と(6)はちょっと厄介です。関数non uniform birthday problem non_unif_bp(k,n, memo, data)は，グループ人数kと，$n=356$，memo，dataの4つの引数からなります。dataに1からnまでの確率p_jの値をリストで代入し，memoは$P\{\overline{A_k};(p_1, ..., p_j)\}$の値を格納しています。一度計算した値をmemoに記憶させておきPythonの辞書機能を使って，memo[memo_key]で

参照することで，２回以上同じ計算を繰り返さない工夫がされています。さらに，もし参照すべき値がなかったら通常はエラーを返すので，それを逆手にとって，新しい値の計算

```
memo[memo_key]=non_unif_bp(k,n-1,
memo, data)+k*data[n-1]*non_unif_
bp(k-1,n-1, memo, data)
```

を行います。これはtry and exceptの機能で実現しています。このようなkとnの２次元の漸化式に基づく再帰的な計算方法は動的計画法と呼ばれる最適化技法の一つで，フィボナッチ数列の計算が代表例です。どことなく，$P\{\overline{A_k};(p_1, ..., p_j)\}$の計算もフィボナッチ数列の計算に似ていますね。

```
def non_unif_bp(k,n, memo, data):
    if k==1:
        memo_key=str(k)+','+str(n)
        s=sum(data[:n])
        memo[memo_key]=s
        return s
    if k==n:
        memo_key=str(k)+','+str(n)
        try:
            return memo[memo_key]
        except KeyError:
            memo[memo_key]=k*data[k-
            1]*non_unif_bp(k-1,n-1,
            memo, data)
            return memo[memo_key]
    else:
        memo_key=str(k)+','+str(n)
        try:
            return memo[memo_key]
        except KeyError:
            memo[memo_key]=non_unif_
            bp(k,n-1, memo, data)+k*
            data[n-1]*non_unif_bp(k-
            1,n-1, memo, data)
            return memo[memo_key]
```

試しにnon_unif_bpを一様分布の下で計算したのが次のプログラムです。

```
if __name__ == '__main__':
# Uniform
    memo={}
    days=365
    data=[1/days]*days
    k=100
    n=len(data)
    non_unif_bp(k,n,memo,data)
    for i in range(1,k+1):
        for j in range(k,n+1):
            non_unif_bp(i,j,memo,data)
    for ik in range(1,k+1):
        key=str(ik)+','+str(n)
        print(ik,round(1-memo[key],4))
```

おわりに

データサイエンスの世界ではPython言語はもっともよく使われる言語ではないかと思います。Pythonは，それまでに開発されていた手続き型言語（C,Fortran），関数型言語（Lisp），スクリプト言語（perl），オブジェクト指向型言語（C＋＋）などの「いいとこどり」をしている印象があり，いろいろな用途に使えます。また，ライブラリが充実していて，基本的な統計解析から機械学習に至るまでデータサイエンスに必要な機能が備わっています。しかし，ユーザは，中でどんなことをやっているのかの理論を理解するのは難しく，また，新しい方法論が生まれて実装されるまでの時間も短くなっていて，なおさら，よく分からず使っているのが現状ではないでしょうか。筆者も例外ではなく，このスピードにはついていけていません。しかし，理論の根本は大学初年度で習う線形代数と微積分ですので，データサイエンスの数理には高校から大学初年度までの基本的な数学がとても重要であり，その理解には一つ一つを数学的にじっくりと考える姿勢が重要だと思います。