順序尺度の相関係数（ポリコリック相関係数）について

小杉考司

1　導入

社会調査において、三件法、五件法で得られたデータを因子分析していたりするけど、実は理論的にはマズイことがある。統計学者に言わせると、連続変数と見なしていいのは七件法からで (吉野・千野・山崎,2007)、それ以下のものはせいぜい順序尺度レベルであるからだ。順序尺度レベルということは、加減乗除の対象にならないってことで、平均をとるのは勿論、相関係数を求めて因子分析するなんて、もってのほかということになる。

しかしまぁ、今までは慣例的に行われてきている。心理学で調査研究をやった人であれば、むしろ五件法なら因子分析せよ、と教わったはずだ。しかしその背後には、「他に方法がないんだから仕方がないでしょ」という妥協があったことを知っている人は少ない。

例えば今でこそ、因子分析といえば最尤法プロマックス回転（あるいはクォーティミン回転。少なくとも斜交回転）が主流になってきたが、そうなると主因子法バリマックス回転で因子分析せよと教わったのは何だったのか、という気持ちになるだろう。あれも時代の要請で、当時は統計パッケージに斜交回転が入ってなかったから、仕方がなかった＝妥協の産物だったのだ（著者の記憶では、SPSSver8 ぐらいから斜交になったはずである。1997 年ぐらいの話。）。

同様に、今となっては五件法をそのまま因子分析するのは時代遅れである。統計学的には、早くから正しい方法が示されている。例えば狩野・三浦 (2002) によると、順序尺度を分析するには

1. 連続とみなす
2. 多分相関係数（polychoric correlation coefficient）, 多分系列相関係数 (polyserial correlation coefficient) を使う
3. 多項分布に基づく方法をとる

の三択になるとしている。

最初のは従来通りの妥協案。最後のは、柳井・繁桝・前川・市川 (1999) に書いてあるらしい。筆者も勉強中。

本稿では、二番目の多分相関係数（ポリコリック相関係数）や多分系列相関係数（ポリシリアル相関係数）について述べる。これらは最近、M-plus や R のパッケージの一つとして算出される**順序尺度の相関係数**であり、これを使えば冒頭に述べたような「妥協」について解決されるのである。

(余談) あくまでも厳密さの程度問題であり、ポリコリックを使ったからといって、劇的に因子構造が変わるというようなことはない。しかし、あくまでも統計的正当性をもとめるのであれば、徹底するべきだと筆者は考えます。

1.1 名称について

聞き慣れない言葉が多く出てくると思われるので、ここで名称の整理をしておく。ポリコリック、ポリシリアル、テトラコリックという三つの言葉についての解説である。

ポリコリック相関係数 Polychoric Correlation は、上述の通り「**多分相関係数**」と訳される。順序尺度と順序尺度の相関係数である。

ポリシリアル相関係数 Polyserial Correlation は、同様に「**多分系列相関係数**」、あるいは「**重双相関係数**」と訳される。順序尺度と連続尺度の相関係数である。例えば項目反応理論の一部、段階反応モデル Graded Response Model や部分採点モデル Partial Credit Model で、順序尺度と見なされる各項目と、間隔尺度と見なされるテストの合計得点は、ポリシリアル相関係数で求める。

テトラコリック相関係数 Tetrachoric Correlation は**四分相関係数**と訳される。四分は 2×2、つまり二値データ同士の相関係数である。これはポリコリック相関係数の特殊な場合である、と考えればよい。基本的には項目反応理論の、正誤データの変数間相関はテトラコリック相関係数になる。各項目とテストの合計得点との相関は、各項目をダミー変数と見なしてピアソンの積率相関係数を求めればよいので、さほど問題にされることはない。

ポリコリック相関係数は、もとのデータを順序尺度と見なして、反応カテゴリ間の距離 (例えば「はい」と「どちらでもない」の距離) を閾値で調整することができる。このため、天井効果・床効果がみられるような反応項目であっても、相関係数を正しく見積もってくれる。実際に算出してみた所感としては、順序尺度を間隔尺度だと見なしてピアソンの積率相関係数を出すより、ポリコリック相関係数のほうが大きい値が得られることが多い。

2 基本的な考え方

基本的な考え方は、データの表現形が順序尺度であって、その背後に連続的なものがある、とするものである。

社会的態度でも何でもいいが、$-\infty$ から ∞ までなめらかに変化する量があって、それが表に出てくるときに「賛成・わからない・反対」の三段階とか、「非常に賛成〜非常に反対」の五段階になってしまうという考え方である。

どこに意見の変わり目（＝閾値）があるのかはわからないけど、背後に連続量を仮定しようというわけである。

さて、変数 x と y がそれぞれ順序尺度で得られたとしよう。この二変数の相関係数を考えるにあたって、まず二変数のクロス集計表を書いてみるところから始めてみよう。x と y それぞれが、s 段階、r 段階に分割されているとしよう。このときクロス集計表は、表 1 のようになる。さて、x と

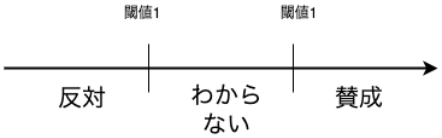

図 1　連続体と表現形。ある量 X1 を超えたら、態度は「反対」から「わからない」に変わる。さらに X2 を超えたら「賛成」に変わる。

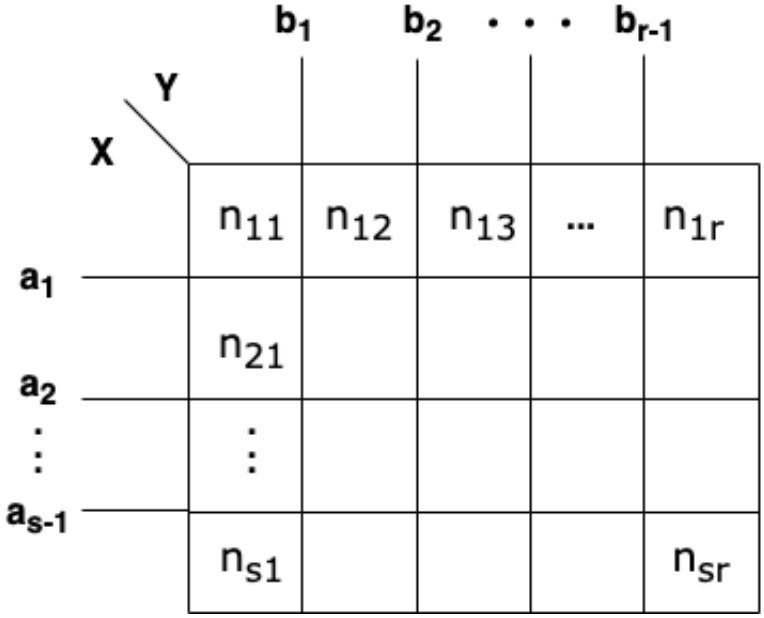

図 2　表 1: 素データの一般的な形。x と y のクロス集計表。ここで a_i や b_j は閾値を表し、$a_0 = b_0 = -\infty$、$a_s = b_r = +\infty$ である。

y の奥に潜在変数 ξ と η があり、それらは正規分布していると仮定するのであった。となれば、変数 x と ξ の関係は次のように書ける。

$$\begin{array}{lll} x = 1 & if & \xi < a_1 \\ x = 2 & if & a_1 \leq \xi < a_2 \\ x = 3 & if & a_2 \leq \xi < a_3 \\ \vdots & & \vdots \\ x = s & if & a_{s-1} \leq \xi \end{array} \tag{1}$$

y についても同様である。パラメータ a_1 が閾値、つまり意見の変わり目である。問題は、ξ と η の相関である ρ を、表 1 から求めたいということである。

3　二変数正規分布

ここで、我々が求めようとしている相関係数 ρ の姿をイメージしておこう。ある変数 X が正規分布する、というのは図にすると図 3 のようなものだ。

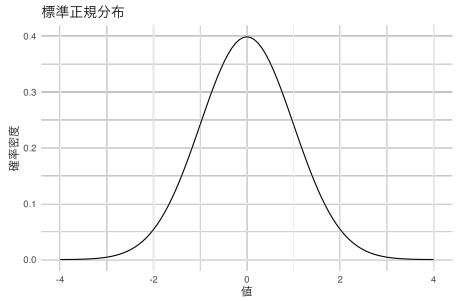

図 3　一変数の正規分布

これが二変数になると、図が立体になる。例えば、全く無相関であれば、図 4 のようになる。

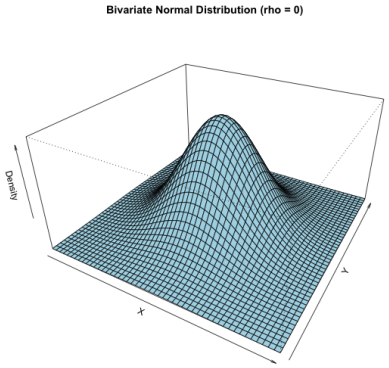

図 4　二変数正規分布、相関 $\rho = 0.0$

徐々に相関係数が大きくなっていけば、立体部が細く尖っていくのが見て取れるだろう。

二つの変数が同時に正規分布するときの確率密度関数 (bivariate normal density function) は、次のような式で表される。ちなみに図 4〜7 はこの式を ggplot2 でプロットしたものである。

$$f(x, y; rho) = \frac{1}{2\pi\sqrt{1-\rho_{xy}^2}} exp\left\{-\frac{1}{2(1-\rho_{xy}^2)}(x^2 - 2\rho_{xy} + y^2)\right\} \tag{2}$$

作画のコードは次のようなものである。

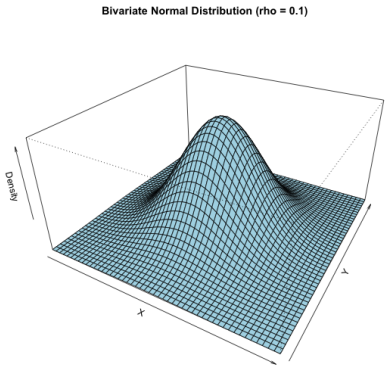

図 5　二変数正規分布、相関 $\rho = 0.1$

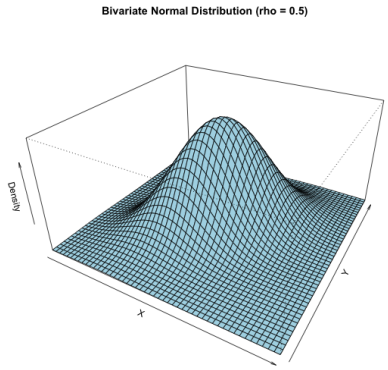

図 6　二変数正規分布、相関 $\rho = 0.5$

```r
# 基本的な設定
library(ggplot2)
x ← seq(-3, 3, length=50)
y ← seq(-3, 3, length=50)

# 二変数正規分布の確率密度関数
f ← function(x, y, rho) {
  1/(2*pi*sqrt(1-rho^2)) * exp(-(1/(2*(1-rho^2)))*(x^2 + y^2 - 2*rho*x*y))
}

# 相関係数ごとにプロット
rho_values ← c(0, 0.1, 0.5, 0.9)
for (rho in rho_values) {
```

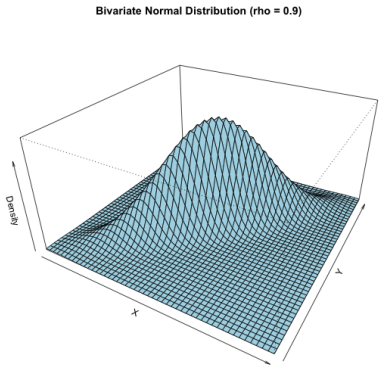

図 7　二変数正規分布、相関 $\rho = 0.9$

```
  z ← outer(x, y, function(x,y) f(x, y, rho))
  df ← expand.grid(X = x, Y = y)
  df$Z ← c(z)

  p ← ggplot(df, aes(x = X, y = Y, fill = Z)) +
    geom_raster() +
    scale_fill_viridis_c() +
    theme_minimal() +
    labs(
      title = paste('Bivariate Normal Distribution (rho =', rho, ')'),
      x = 'X',
      y = 'Y',
      fill = 'Density'
    ) +
    coord_fixed()

  ggsave(
    paste0('bivariate_normal_rho', sprintf('%02d', rho*10), '.png'),
    p,
    width = 6,
    height = 6
  )
}
```

さてここで，二次元クロステーブル，表 3 があったとしよう

この分割表から，図 8 のような 3D ヒストグラムをイメージしてもらいたい。3D グラフは集計などには向いてないが，今回は有用なので心を鬼にして表計ソフトで描いた。さて，これのうえにふわりとシーツを被せたとしよう。それが二次元正規分布のイメージだ。その二次元正規分布の相

	A	B	C	合計
1	10	26	8	44
2	20	58	12	90
3	9	21	9	39
合計	39	105	29	173

関係数, ρ の値はどれぐらいが妥当なのだろうか？ と考えるのがポリコリック相関係数の最尤推定量ということになる。

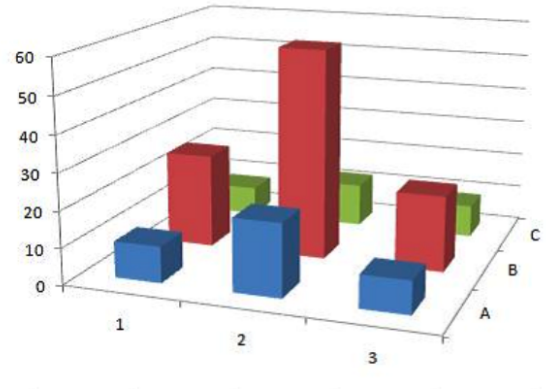

図 8　度数分布と二次元ヒストグラム

4　最尤推定の二つの方法

ポリコリック相関係数を推定するとき、推定すべきパラメータは、相関係数 ρ だけではなく、変数 x に含まれる閾値 $a_1, a_2, ..., a_s$ と変数 y に含まれる閾値 $b_1, b_2, ..., b_r$ も、ということになる。

具体的な方法としては二種類ある。一つは、全てのパラメータを同時に求める方法で、これは理論的に正しいが偏微分連立方程式を解くことになるので面倒である。もう一つの方法は、閾値を周辺度数からもとめちゃって、ρ についての方程式一つにしちゃう、というものである。後者の方法を、特に二段階最尤推定 twe-step ML 法という。実際、アプリケーションに実装する上では二段階最尤推定法が主に用いられているようである。

ところで、この最尤推定アプローチには、標準誤差が対数尤度の二次導関数の逆数として簡単に得られるというメリットもある。本稿では、以下 Olsson(1979) に沿って議論を進めていくが、標準誤差の推定など、より詳細は原典を当たって欲しい。

4.1 尤度方程式の導出

データが観測度数 n_{ij} から構成されているとする。ここで、表 1 の通り $i = 1, 2, \cdots, s, j = 1, 2, \cdots, r$ である。

ここで、観測度数がセル (i,j) に落ち込む確率を π_{ij} とすれば、そのサンプルの尤度は

$$L = C \cdot \prod_{i}^{s} \cdot \prod_{j}^{r} \pi_{ij}^{n_{ij}} \tag{3}$$

である。ここで C は定数。対数をとると、

$$l = lnL = \ln C + \sum_{i=1}^{s} \sum_{j=1}^{r} n_{ij} \ln \pi_{ij}. \tag{4}$$

x における閾値は a_i で表され、$i = 0, 1, \cdots, s$ である。y における閾値は b_j で、$j = 0, 1, \cdots, r$ である。ここで、$a_0 = b_0 = -\infty$、$a_s = b_r = +\infty$ である。従って、

$$\pi_{ij} = \Phi_2(a_i, b_j) - \Phi_2(a_{i-1}, b_j) - \Phi_2(a_i, b_{j-1}) + \Phi_2(a_{i-1}, b_{j-1}) \tag{5}$$

となる。ここで Φ_2 は、相関係数 ρ のときの二変数正規分布関数 bivariate normal distribution function with ρ である。

4.2 手法 1: 全てのパラメータを同時に推定する

推定すべきパラメータは、$\rho, a_1, \cdots, a_{s-1}, b_1, \cdots, b_{r-1}$ である。l についての偏微分をそれぞれのパラメタで行うと、

$$\frac{\partial l}{\partial \rho} = \sum_{i=1}^{s} \sum_{j=1}^{r} \frac{n_{ij}}{\pi_{ij}} \frac{\partial \pi_{ij}}{\partial \rho} \tag{6}$$

$$\frac{\partial l}{\partial a_k} = \sum_{i=1}^{s} \sum_{j=1}^{r} \frac{n_{ij}}{\pi_{ij}} \frac{\partial \pi_{ij}}{\partial a_k} \tag{7}$$

$$\frac{\partial l}{\partial b_m} = \sum_{i=1}^{s} \sum_{j=1}^{r} \frac{n_{ij}}{\pi_{ij}} \frac{\partial \pi_{ij}}{\partial b_m}. \tag{8}$$

となる。ここで、ϕ_2 を二変数正規密度関数とすると、$\partial \Phi_2(u, v)/\partial \rho = \phi_2(u, v)$ である。すると

$$\frac{\partial \pi_{ij}}{\partial \rho} = \phi_2(a_i, b_j) - \phi_2(a_{i-1}, b_j) - \phi_2(a_i, b_{j-1}) + \phi_2(a_{i-1}, b_{j-1}). \tag{9}$$

であり、式 6 は次のように書き直せる。

$$\frac{\partial l}{\partial \rho} = \sum_{i=1}^{s} \sum_{j=1}^{r} \frac{n_{ij}}{\pi_{ij}} \left\{ \phi_2(a_i, b_j) - \phi_2(a_{i-1}, b_j) - \phi_2(a_i, b_{j-1}) + \phi_2(a_{i-1}, b_{j-1}) \right\}. \tag{10}$$

式 7 においては、次のことが明らかである。

$$\frac{\partial \pi_{ij}}{\partial a_k} = \begin{cases} 0 & i \neq k \text{かつ} i \neq k+1 \text{であれば、} \pi_{ij} \text{の式に} a_k \text{を含まないので} \\ \frac{\partial \Phi_2(a_k, b_j)}{\partial a_k} - \frac{\partial \Phi_2(a_k, b_{j-1})}{\partial a_k} & k = i \text{のとき} \\ -\frac{\partial \Phi_2(a_k, b_j)}{\partial a_k} + \frac{\partial \Phi_2(a_k, b_{j-1})}{\partial a_k} & k = i-1 \text{のとき} \end{cases} \tag{11}$$

だから、式 7 は i を k から $k+1$ まで変化させれば十分である。故に、式 7 は

$$\begin{aligned} \frac{\partial l}{\partial a_k} &= \sum_{j=1}^{r} \frac{n_{ij}}{\pi_{kj}} \left\{ \frac{\partial \Phi_2(a_k, b_j)}{\partial a_k} - \frac{\partial \Phi_2(a_k, b_{j-1})}{\partial a_k} \right\} \\ &+ \frac{b_{k+1j}}{\pi_{k+1j}} \left\{ -\frac{\partial \Phi_2(a_k, b_j)}{\partial a_k} + \frac{\partial \Phi_2(a_k, b_{j-1})}{\partial a_k} \right\} \\ &= \sum_{j=1}^{r} \left(\frac{n_{kj}}{\pi_{kj}} - \frac{n_{k+1j}}{\pi_{k+1j}} \right) \left\{ \frac{\partial \Phi_2(a_k, b_j)}{\partial a_k} - \frac{\partial \Phi_2(a_k, b_{j-1})}{\partial a_k} \right\}. \end{aligned} \tag{12}$$

また、もし ϕ_1 と Φ_1 を、それぞれ一変数についての生起確率密度と正規分布関数とすれば、

$$\frac{\partial \Phi_2(u, v)}{\partial u} = \phi_1(u) \cdot \Phi_1 \left\{ \frac{(v - \rho u)}{(1 - \rho^2)^{1/2}} \right\} \tag{13}$$

となる (Tallis,1962,p.346)。さすれば式 7 は、次のようになる。

$$\frac{\partial l}{\partial a_k} = \sum_{j=1}^{r} \left(\frac{n_{kj}}{\pi_{kj}} - \frac{n_{k+1j}}{\pi_{k+1j}} \right) \cdot \phi_1(a_k) \cdot \left[\Phi_1 \left\{ \frac{(b_k - \rho a_k)}{(1 - \rho^2)^{1/2}} \right\} - \Phi_1 \left\{ \frac{(b_{j-1} - \rho a_k)}{(1 - \rho^2)^{1/2}} \right\} \right]. \tag{14}$$

同様に、

$$\frac{\partial l}{\partial b_m} = \sum_{j=1}^{r} \left(\frac{n_{im}}{\pi_{im}} - \frac{n_{im+1}}{\pi_{im+1}} \right) \cdot \phi_1(b_m) \cdot \left[\Phi_1 \left\{ \frac{(a_i - \rho b_m)}{(1 - \rho^2)^{1/2}} \right\} - \Phi_1 \left\{ \frac{(a_{i-1} - \rho b_m)}{(1 - \rho^2)^{1/2}} \right\} \right]. \tag{15}$$

これらの式 10,14,15 が、対数尤度の一次導関数を構成することになる。

4.3 方法 2:Two-Step 最尤推定法〜閾値を周辺度数から算出する場合〜

この場合、方程式は次のようにたてられる。

$$\frac{\partial l}{\partial \rho} = \sum_{i=1}^{s} \sum_{j=1}^{r} \frac{n_{ij}}{\pi_{ij}} \left\{ \phi_2(a_i, b_j) - \phi_2(a_{i-1}, b_j) - \phi_2(a_i, b_{j-1}) + \phi_2(a_{i-1}, b_{j-1}) \right\} = 0. \tag{16}$$

$$a_i = \Phi_1^{-1}(P_{i.}) \tag{17}$$

$$b_j = \Phi_1^{-1}(P_{.j}) \tag{18}$$

ここで P_{ij} はセル (i, j) の観測度数であり、$P_{i.}$ と $P_{.j}$ は観測された累積周辺分布である。つまり、

$$P_{i.} = \sum_{k=1}^{i} \sum_{j=1}^{r} P_{kj} \tag{19}$$

かつ

$$P_{\cdot j} = \sum_{i=1}^{s} \sum_{k=1}^{j} P_{ik} \tag{20}$$

である。

5 具体的例

ここでは、二段階最尤推定法によるポリコリック相関係数を，R で求めるコードを書いてみよう。

まず、式 17 や式 18 にみられる一変数正規分布の逆関数だが、これは数値計算的にはあるパーセンタイル点の上側確率として知られているものである。まあ面倒なことを考えなくても，R には確率分布関数が既に入っているから，qnorm(x, mean , sd) と書くだけで OK だ。

とはいえ，その関数でどういう計算をしているのか気になるというチョーマニアックな人もいるかもしれない。旧版では C 言語での実装を想定していたからしていたから，近似計算法を調べたんですよ。そしたらあるものですな。こんな感じです (パソコン統計解析ハンドブック 1 基礎統計編より。有効桁数下三桁)。これを R に書き直すと次のようになる[*1]。

```
normal_percent ← function(val) {
    # 係数の定義
    b0 ← 0.1570796288 * 10
    b1 ← 0.3706987906e-1
    b2 ← -0.8364353589e-3
    b3 ← -0.2250947176e-3
    b4 ← 0.6841218299e-5
    b5 ← 0.5824238515e-5
    b6 ← -0.1045274970e-5
    b7 ← 0.8360937017e-7
    b8 ← -0.3231081277e-8

    # 0.5より大きい場合は1から引く
    v ← ifelse(val > 0.5, 1 - val, val)

    # 計算式
    y ← -log(4 * v * (1 - v))

    # 多項式の計算
    sum ← b0 + (b1 * y) + (b2 * y^2) + (b3 * y^3) + (b4 * y^4) +
          (b5 * y^5) + (b6 * y^6) + (b7 * y^7) + (b8 * y^8)
    sum ← sum * y

    # 結果の計算
```

[*1] ちなみに書き写すのは面倒だろうから，Gist にコードをあげた。https://gist.github.com/kosugitti/70299c94b9bbab93c5ba99240cac2574 をみて欲しい。

```
    alpha ← sqrt(sum)
    if (val < 0.5) alpha ← -alpha

    return(alpha)
}
```

試しに実行してみよう。

```
> normal_percent(0.489)
[1] -0.02761213
> normal_percent(0.872)
[1] 1.137076
> normal_percent(0.405)
[1] -0.2396873
> normal_percent(0.943)
[1] 1.578123
```

このように，qnorm 関数と同じような結果が得られることがわかる。
数値例を元に考えてみよう。次のようなデータがあったとする。

表 1　クロス集計表の例 (出典：Encyclopedia of Statistical Sciences;Polychoric and Polyserial Correlations より)

	A	B	C	合計
甲	58	52	1	111
乙	26	58	3	87
丙	8	12	9	29
合計	92	122	13	227

例えば、一列目と二列目の間にある閾値は、qnorm(111/227) とすれば $\hat{a_1}$ が得られる。

$$\hat{a_1} = \Phi^{-1}(111/227) = \Phi^{-1}(0.489) = -0.02761$$

である。以下同様に、qnorm(198/227) とすれば $\hat{a_2}$ が得られる。

$$\hat{a_2} = \Phi^{-1}(198/227) = \Phi^{-1}(0.872) = 1.13708$$

同様に、qnorm(92/227) とすれば $\hat{b_1}$ が得られる。

$$\hat{b_1} = \Phi^{-1}(92/227) = \Phi^{-1}(0.405) = -0.23969$$

最後も同様に、qnorm(214/227) とすれば $\hat{b_2}$ が得られる。

$$\hat{b_2} = \Phi^{-1}(214/227) = \Phi^{-1}(0.943) = 1.57812$$

と推定される。これをもとに、式 16 を解くことになる。

式 16 をみてみると、どうやら二変量正規分布の確率密度関数 (ϕ_2;probability density function;PDF) と累積分布関数 (Φ_2;cumulative distribution function;CDF) が要りそうである。前者は式 2 で求められるが、後者は数値計算による近似計算となる。具体的なプログラムとして、ハル (1994) を参考に，R で書いたものを以下に示す[*2]。

ちなみに，普通の人はこんなことしなくていいのである。R の mvtnorm パッケージには，二次元正規分布バージョンの qnorm 関数である，pmvnorm 関数があるからだ。どうしても車輪を再発明したい人はどうぞ。

```
qBiNormal ← function(a, b, rho) {
    if (is.na(a) || is.na(b) || is.na(rho)) {
      warning("NA values detected in qBiNormal inputs")
      return(NA)
    }
    # support function
    f ← function(x, y, aprime, bprime, rho) {
      exp(aprime * (2 * x - aprime) + bprime * (2 * y - bprime) +
        2 * rho * (x - aprime) * (y - bprime))
    }

    # special case : Inf/-Inf
    if (is.infinite(a) && is.infinite(b)) {
      if (a > 0 && b > 0) {
        return(1)
      }
      if (a < 0 && b < 0) {
        return(0)
      }
      return(0)
    }

    # othre special case of Inf/-Inf
    if (is.infinite(a) || is.infinite(b)) {
      if (any(c(a, b) == -Inf)) {
        return(0)
      }
      #
      if (is.infinite(a)) {
        return(pnorm(b))
      } else if (is.infinite(b)) {
        return(pnorm(a))
      }
```

[*2] このコードは、Gist にもある。https://gist.github.com/kosugitti/6f690a6552564c60b15b5950e7bbce93 をみて欲しい。

```
  }

  # main loop
  if (a ≤ 0 && b ≤ 0 && rho ≤ 0) {
    aprime ← a / sqrt(2 * (1 - rho^2))
    bprime ← b / sqrt(2 * (1 - rho^2))

    # const
    A ← c(
      5.54433663102343E-02, 1.24027738987730E-01, 1.75290943892075E-01,
      1.91488340747342E-01, 1.63473797144070E-01, 1.05937637278492E-01,
      5.00270211534535E-02, 1.64429690052673E-02, 3.57320421428311E-03,
      4.82896509305201E-04, 3.74908650266318E-05, 1.49368411589636E-06,
      2.55270496934465E-08, 1.34217679136316E-10, 9.56227446736465E-14
    )

    B ← c(
      2.16869474675590E-02, 1.12684220347775E-01, 2.70492671421899E-01,
      4.86902370381935E-01, 7.53043683072978E-01, 1.06093100362236E+00,
      1.40425495820363E+00, 1.77864637941183E+00, 2.18170813144494E+00,
      2.61306084533352E+00, 3.07461811380851E+00, 3.57140815113714E+00,
      4.11373608977209E+00, 4.72351306243148E+00, 5.46048893578335E+00
    )

    # double summation
    sum ← 0
    for (i in 1:15) {
      for (j in 1:15) {
        sum ← sum + A[i] * A[j] * f(B[i], B[j], aprime, bprime, rho)
      }
    }

    return(sum * sqrt(1 - rho^2) / pi)
  }

  if (a * b * rho ≤ 0) {
    if (a ≤ 0 && b ≥ 0 && rho ≥ 0) {
      return(pnorm(a) - qBiNormal(a, -b, -rho))
    } else if (a ≥ 0 && b ≤ 0 && rho ≥ 0) {
      return(pnorm(b) - qBiNormal(-a, b, -rho))
    } else if (a ≥ 0 && b ≥ 0 && rho ≤ 0) {
      return(pnorm(a) + pnorm(b) - 1 + qBiNormal(-a, -b, rho))
    }
  }

```

```
    if (a * b * rho ≥ 0) {
      denum ← sqrt(a^2 - 2 * rho * a * b + b^2)
      rho1 ← ((rho * a - b) * sign(a)) / denum
      rho2 ← ((rho * b - a) * sign(b)) / denum
      delta ← (1 - sign(a) * sign(b)) / 4

      return(qBiNormal(a, 0, rho1) + qBiNormal(b, 0, rho2) - delta)
    }

    # error case
    return(-99.9)
}
```

ちなみに、このプログラムのメインでは 10^{-15} の精度まで算出できるようになっているが、実際には 10^{-4} ぐらいで十分であろう。その場合は、次の数列を使えばよい。

```
    double A[4] = { 0.3253030,0.4211071,0.1334425,0.006374323 };
    double B[4] = { 0.1337764,0.6243247,1.3425378,2.2626645};
    double sum = 0;
    for( int i = 0 ; i < 4 ; i++ ){
        for( int j = 0; j < 4 ; j++ ){
            sum = sum + A[i] * A[j] * f( B[i], B[j],aprime,bprime,rho);
        }
    }
```

さて数値例を見てみよう。表 1 を用いた結果、相関係数 ρ は 0.419899 と推定される。その場合、この CDF ルーチンを使って算出される、各セル出現期待値 (式 5 の π_{ij}) は次の通りである。これ

I	j	π_{ij}
1	1	0.2651410
1	2	0.2139610
1	3	0.0098850
2	1	0.1199310
2	2	0.2373620
2	3	0.0259676
3	1	0.0202151
3	2	0.0861221
3	3	0.0214161

らの関数を使って、式 16 の方程式の解を求めることになる。

非線形方程式を解くアルゴリズムはニュートン法（あるいはニュートン・ラフソン法）が有名だが、途中でさらに微分した式を必要とするので、筆者には力不足であった・・・というのが以前の

この資料である。あれから時間がずいぶん流れ，私も R もバージョンが上がって，なんとこれを解くコードが書けるようになった。

まず最適化に必要な尤度関数を用意する。ここでは素直に qnorm 関数を使っている。ちなみに pmin 関数は、二つのベクトルの要素ごとの最小値を返す関数である。なぜか Apple シリコンの R だけが，cumsum 関数で 1 を超え，NA を返すので、それを修正するためにはめ込んでいる[*3]。

```
polychoric_likelihood ← function(rho, mat) {
  nr ← nrow(mat)
  nc ← ncol(mat)
  th_r ← rowSums(mat) / sum(mat)
  th_c ← colSums(mat) / sum(mat)
  a ← qnorm(pmin(cumsum(th_r), 1))
  b ← qnorm(pmin(cumsum(th_c), 1))

  exp_probs ← matrix(0, nrow = nr, ncol = nc)
  for (i in 1:nr) {
    for (j in 1:nc) {
      a_low ← if (i > 1) {
        a[i - 1]
      } else {
        -Inf
      }
      a_high ← a[i]
      b_low ← if (j > 1) {
        b[j - 1]
      } else {
        (-Inf)
      }
      b_high ← b[j]

      exp_probs[i, j] ← qBiNormal(a_high, b_high, rho) -
        qBiNormal(a_high, b_low, rho) -
        qBiNormal(a_low, b_high, rho) +
        qBiNormal(a_low, b_low, rho)
    }
  }

  log_lik ← sum(mat * log(pmax(exp_probs, 1e-10)))
  return(-log_lik)
}
```

さらにこれを最適化関数，optim で呼び出す。ここでは二つの引数，x と y を受け取り、欠損を除外してペアワイズベクトルにしてからクロス表行列を作り，最適化関数に放り込むようになって

[*3] 他の環境では発生しないから，バグだろうし，修正されると思う。2025/03/08 現在のお話でした。

いる。

```
polychoric ← function(x, y) {
    x[x == -1] ← NA
    y[y == -1] ← NA
    pairwise ← !is.na(x + y)
    x ← x[pairwise]
    y ← y[pairwise]
    mat ← table(x, y)
    fit  ← optim(
      par = 0,
      fn  = polychoric_likelihood,
      mat = mat,
      method = "Brent",
      lower = -1,
      upper = 1
    )
    if  ( fit $convergence != 0) {
      stop("Failed to converge when calculating polychoric correlation. Try with different  initial values or check your data.")
    }
    cor ← fit$par
    return(cor)
}
```

これらの関数は，筆者が開発した exametrika パッケージに含まれている。もちろん polychor パッケージの polychor 関数のほうが，より高速に計算できるし確実だろう。ここはまあ，「できるようになったこと」を示すためのアップデートなのでお楽しみください。

6　最後の独り言

この資料を最初に作ったのが 2008 年 2 月である。その頃は，R のバージョンが 2.5.1 で，その頃のパッケージは polychor パッケージのみであった。その頃は，こんなコードを書いていたんだなあ，そして optim 関数が使えなくてフルスクラッチでやるのは挫折したんだなあ，と思い出した。

あれから 20 年弱 (正確には 17 年)。統計に関する知識も R に関する知識もだいぶ深まった。人間，何かしら成長するものである。2023 年度の一年間にサバティカルの期間をもらい，その間にテスト理論のパッケージを作り始めた。それを今度は学会のセミナー講師として提供することになり，それじゃあ段階反応モデルも組み込もうか，やっぱりポリコリック相関係数を自前で用意したいなあ・・・という流れで昔の資料を引っ張り出したのだ。生成 AI の助けも少し借りられたが，まあなんとか「最適化関数」が使えるようになったのである。遅いけど，自分で書いたコードなので感慨深い。

段階反応モデルそのものの実装は，ltm パッケージにやや精度は負けるものの，自分で書いて

exametrika パッケージに組み込んだ。

大事なのは，「ほんとうのこと」を「自分が理解する」ことだ。誰かの書いた関数，誰かの書いたパッケージでは，その原理を理解していても本当に分かったと言えるのかどうか。もちろん全てのことについて，全ての原理を知る必要はなかろう。ゆうきまさみが「パトレイバー」のなかで篠原遊馬に言わせたように，車はアクセル踏みゃ走る，水道の蛇口を撚れば水が出る，でもいいのかも知れない。でも，全ての原理を誰かに任せるのではなく，自分の中に「ほんとうのこと」を積み重ねていくことが大事なのだ。李書文先生が (「健児」のなかで) 言うように，「他人が飛んだり跳ねたり，毎日パタパタ打ち合っている間に，絶対的な功夫を養うのだ」。

あー，楽しかった。

7　参考文献

1. Hull,J. Options, futures, and other derivative securities 社団法人金融財政事情研究会 (訳) フィナンシャルエンジニアリング, 株式会社きんざい.
2. 狩野裕・三浦麻子　2002　グラフィカル多変量解析—AMOS、EQS、CALIS による 目で見る共分散構造分析 現代数学社
3. Olsson,U. 1979 Maximum Likelihood Estimation of the Polychoric Correlation Coefficient. Psychometrika,44(4),443–460.
4. 戸川隼人 1992 UNIX ワークステーションによる科学技術計算ハンドブック　サイエンス社
5. 柳井晴夫・繁枡算男・前川眞一・市川雅教　1999　因子分析－その理論と方法－朝倉書店
6. 吉野諒三・千野直仁・山岸侯彦　2007　数理心理学　培風館