ノンパラメトリック検定に使用する統計数値表について

青木繁伸

群馬大学社会情報学部

Shigenobu Aoki

Faculty of Social and Information Studies,
Gunma University.

4–2 Aramaki–machi, Maebashi, Gunma 371-0846

2005 年 10 月 18 日

1 はじめに

ノンパラメトリック検定に用いられる統計数値表は，少数例についてのみ用意されている。数値表にない範囲においては，正規近似などを用いれば十分なこともある。

広い範囲の数値表が準備されない一つの理由は，計算所要時間が $n!$ や 2^n に比例するためである。もう一つの理由は，コンピュータで扱える数値の範囲が整数では普通 0 〜 4294967295 (= $2^{32} - 1$) までであることである。

C 言語コンパイラの中には，特別な工夫をしなくても，20 桁くらいの整数を扱うことのできるものもある。しかし，$20! \simeq 2.43 \times 10^{18}$ であることを考えると，アルゴリズムによっては計算の限界は意外に早く訪れる。大規模計算にはメインフレームコンピュータを用いることが考えられるが，通常の利用資格しか持たないユーザが利用可能な時間は限られているので，長時間かかる計算処理は場合によってはワークステーション上で行った方がよい場合もでてくる。また，マイクロコンピュータの発達も著しいので，よいアルゴリズムを用いた場合は速度的にも十分な場合がある。

今回，マイクロコンピュータを用いて，3 種類のノンパラメトリック検定に必要な数値表を得るためのアルゴリズムを見出したので報告する。

2 方法および機器

アルゴリズムの検証のために，C 言語 [Kernighan and Ritchie 1988] によるプログラムを作成し，マイクロコンピュータで実行結果を確認した。

使用したコンパイラは Borland Japan の BC++3.1 版 [Borland International 1991]，マイクロコンピュータ

は，数値演算プロセッサ 80387 を搭載した NEC PC–9801FA を用いた。

3 結果

3.1 Wilcoxon の符号順位検定

この検定は，対応のある 2 変数の観察値の差 $d_i = x_i - x'_i$, $i = 1, 2, \ldots, n$ を取り，差が 0 でないものについて差の絶対値に順位を付け，符号別に順位の和を取ったものを検定統計量とする [Siegel 1956]。帰無仮説 H_1 は，$\sum d_i = 0$ である。

今，差が 0 でない観察値の組が n 個あり，正の符号を持ったものを 1，負の符号を持ったものを 0 と表すと，以下のような n ビットの 2 進数で表せる。

n	$n-1$	$n-2$	$n-3$	$n-2$	$\ldots$	3	2	1
0	0	1	1	0	$\ldots$	1	0	1

右端のビットから順にビット i $(i = 1, 2, \ldots, n)$ として，ビット i に重み i を与えて合計したものが検定統計量になる。これを S_n と表し，統計量の度数分布を $\mathbf{S}_n$ と表すことにする。

上の例では，$S_n = 1 + 3 + \ldots + (n-3) + (n-2)$ である。

ここで，ビット n は 0 か 1 であるが，これを除いたビット $n-1, \ldots, 2, 1$ で決まる統計量 S_{n-1} を考えよう。

n	$n-1$	$n-2$	$n-3$	$n-2$	$\ldots$	3	2	1
x	0	1	1	0	$\ldots$	1	0	1

x が 0 の場合は，$S_n = S_{n-1}$ である。このときの統計量の度数分布を ${}_0\mathbf{S}_{n-1}$ と表現する。

x が 1 の場合は，$S_n = S_{n-1} + n$ になる。このときの統計量の度数分布を ${}_1\mathbf{S}_{n-1}$ と表現する。この度数分布は，${}_0\mathbf{S}_{n-1}$ を n だけ平行移動したものである。

ところで，$\mathbf{S}_n$ は，${}_0\mathbf{S}_{n-1}$ と ${}_1\mathbf{S}_{n-1}$ の合成であるから，度数分布の合成を表す演算子を $\oplus$ とすると，$\mathbf{S}_n = {}_0\mathbf{S}_{n-1} \oplus {}_1\mathbf{S}_{n-1}$ と表現できる。

したがって，初期状態として $n = 1$ の場合の統計量の度数分布が決められれば，$n \geq 2$ の場合の統計量の度数分布は順次決定できる。

- $n = 1$ の場合
  $S_1 = 0$ と $S_1 = 1$ が，それぞれ度数 1 である。
- $n = 2$ の場合
  ビット 2 が 0 のときは，${}_0\mathbf{S}_2$ は，$\mathbf{S}_1$ と同じであるので，${}_0S_2 = S_1 = 0$ と ${}_0S_2 = S_1 = 1$ が，それぞれ度数 1 である。
  ビット 2 が 1 のときは，${}_1S_2 = S_1 + 2 = 2$ と ${}_1S_2 = S_1 + 2 = 3$ であり，それぞれの度数はビット 2 が 0 の場合の度数と同じ (1) である。

この手順の最初の部分は，表 1 に示すように逐次的に進められる。

なお，上では $n = 1$ からスタートしたが，$n = 0$ のときに $S_0 = 0$ が度数 1 であると考えてスタートしてもよい。以上のアルゴリズムをプログラム化したものを，付録 付録 A に示す。

3.2 Mann–Whitney の U 検定 (Wilcoxon の順位和検定)

この検定は，ケース数が n_1, n_2 である 2 つの群の観察値を小さい順に並べて，群ごとの順位の和を検定統計量にするものである [Siegel 1956]。

今，観察値が n $(= n_1 + n_2)$ 個あり，第 1 群の観察値を 1，第 2 群の観察値を 0 と表すと，n ビットの 2 進数で表せる。ビット列中に 1 が n_1 個，0 は n_2 個あることになる。

右端のビットから順にビット i $(i = 1, 2, \ldots, n)$ として，ビット i に重み i を与えて合計したものは Wilcoxon

の順位和検定の検定統計量である。Mann–Whitney の U 検定はこれを最小値が 0 (最大値は $n_1 \times n_2$) になるように平行移動した検定統計量を用いる。したがって，両者は全く等価な検定である。以下では，U 検定統計量 $S^{n_1}_{n_2}$ に基づいて述べる。また，$\mathbf{S}^{n_1}_{n_2}$ をその度数分布を表すものとする。

検定統計量は，ビットが 1 であるものの順位を合計する場合 (これを $S^{n_1}_{n_2}$ と表記する) と，ビットが 0 であるものの順位を合計する場合 (これを $\tilde{S}^{n_1}_{n_2}$ と表記する) がある。両者は，何れを第 1 標本にするかの違いでもあるが，両者の間には，$\tilde{S}^{n_1}_{n_2} = n_1 \times n_2 - S^{n_1}_{n_2} = S^{n_2}_{n_1}$ の関係がある。また，n_1 と n_2 は対称であるので，$n_1 \geq n_2$ である場合に限定してよい。

ここで，3.1 節のときと同様に，ビット n は 0 か 1 であるが，これを除いたビット $n-1,\ldots,2,1$ を考えよう。それぞれの場合の統計量を，${}_0S^{n_1}_{n_2}$，${}_1S^{n_1}_{n_2}$，また，その度数分布を ${}_0\mathbf{S}^{n_1}_{n_2}$，${}_1\mathbf{S}^{n_1}_{n_2}$ と表記する。

- ビット n が 0 の場合
  ビット n を除いた残りのビット列中には，1 が n_1 個，0 が $n_2 - 1$ 個あることになる。
  したがって，${}_0\mathbf{S}^{n_1}_{n_2} = \mathbf{S}^{n_1}_{n_2-1}$ である。
- ビット n が 1 の場合
  ビット n を除いた残りのビット列中には，1 が $n_1 - 1$ 個，0 が n_2 個あることになる。これは，ビット n が 0 の場合の ビット n を除いたビット列中の 0 と 1 を反転させたものの度数分布すなわち，n_1 と n_2 を入れ替えた場合の度数分布と同じであることになる。したがって，${}_1\mathbf{S}^{n_1}_{n_2} = \tilde{\mathbf{S}}^{n_1-1}_{n_2} = \mathbf{S}^{n_2}_{n_1-1}$ である。

$\mathbf{S}^{n_1}_{n_2} = {}_0\mathbf{S}^{n_1}_{n_2} \oplus {}_1\mathbf{S}^{n_1}_{n_2} = \mathbf{S}^{n_1}_{n_2-1} \oplus \tilde{\mathbf{S}}^{n_1-1}_{n_2} = \mathbf{S}^{n_1}_{n_2-1} \oplus \mathbf{S}^{n_2}_{n_1-1}$ であるから，$n_1 = n_2 = 1$ の場合の統計量の度数分布が分っていれば，これより大きな観察数の場合の統計量の度数分布は全て決定できる。なお，n_1 か n_2 が 0 の場合には統計量が 0 となる場合が度数 1 だけ存在するとすればよい。

以上のアルゴリズムをプログラム化したものを，付録 付録 B に示す。付録 B.1 のプログラムは上記のアルゴリズムに基づいているが，再帰呼出しを伴うので n_1, n_2 が大きくなると計算時間がかかり実用的ではない。付録 B.2 のプログラムは再帰呼出しをしないので，結果を出力するのに要する時間に比べると計算時間は無視できる程度である。ただし，より多くのメモリを必要とする。

3.3 Kendall の順位相関係数

この検定では，2 変数 (X, Y) の n 組について，まずそれぞれの変数ごとに順位を付け，片方の変数 (例えば X) の順位の小さい順に並べ変える。次に，もう一方の変数 (Y) の順位 $O_i (i = 1, 2, \ldots, n)$ について，$O_i < O_j$, $(i < j)$ である組合せの数を数え上げたものが，Kendall の順位相関係数 τ の検定統計量 S_n になる [Siegel 1956]。また，$\mathbf{S}_n$ をその度数分布とする。

なお，Kendall は，$S' = n(n-1)/2 - 2S_n$ を表にしている。

また，Kendall の順位相関係数 τ は，(1) 式で得られる。

$$\tau = \frac{2S_n}{n(n-1)/2} - 1 \tag{1}$$

- $n = 1$ の場合
  $S_1 = 0$ が度数 1 だけある。
- $n = 2$ の場合
  Y の順位は 1, 2 の順か 2, 1 の順である 2 通りしかない。それぞれ $S_2 = 1$ および $S_2 = 0$ である。前者は，$n = 1$ の場合の $S_1 = 0$ に順位 2 が加わったため，統計量が 1 増加し $S_2 = 1$ となったものである。後者は，順位 2 が加わったことによる影響を受けない。この 2 つの度数分布を重ねたものが $\mathbf{S}_2$ である。
- $n = 3$ の場合
  S_3 は，$n = 2$ の場合の順位に順位 3 が加わったものである。順位 3 の位置により，$n = 2$ の場合の統計量 S_2 に加えられる増加分と度数の増加には規則性がある。

すなわち，変数の組が n の場合の統計量の度数分布 $\mathbf{S}_n$ は，変数の組が $(n-1)$ の場合の統計量の度数分布

$\mathbf{S}_{n-1}$ が分っていれば決定できる。統計量 S_n のそれぞれの値について，S_{n-1} と比べた増分は 0 〜 $n-1$ である。

これをプログラム化したものを，付録 付録 C に示す。

4 考察

付録 付録 A に示したプログラムでは，$n \leq 68$ までの Wilcoxon 統計量の分布を求めることができる。付録 B.2 に示したプログラムでは，$27 \geq n_1 \geq n_2$ までの Mann–Whitney 統計量の分布を求めることができる。付録 付録 C に示したプログラムでは，$n \leq 20$ までの Kendall 統計量の分布を求めることができる。

これらの上限は，多倍長演算クラスライブラリを使用することによって，容易に拡張できる。多倍超演算に必要な計算時間は，アルゴリズムの高速性でカバーされる。

以下に，これらの計算結果の応用例を示す。

4.1 対応のあるデータの中央値の差の信頼区間

同一個体の繰り返し測定，またはマッチングされたケース–コントロールの比較などにより得られる対応のあるデータの差の信頼区間を求める場合を考える。この場合，2 つの分布は位置の母数以外は同じであり，対応付けられたデータの差の分布は対称であると仮定する。

標本の大きさを n とし，対応づけられた測定値の差を $d_1, d_2, \ldots, d_n$ とする。2 つの母集団の差および差の信頼区間は，あらゆる 2 つの測定値の差の組合せについての $n(n+1)/2$ 個の平均値 $(d_i + d_j)/2$ に基づいて求められる。ほぼ $100(1-\alpha)\%$ の信頼区間を求めるには，まず Wilcoxon 統計量の $100\alpha/2\%$ 点 $W_{\alpha/2}$ を求める。$W_{\alpha/2} = K^\star$ としたとき，$n(n+1)/2$ 個の測定値の差の平均値の小さい方から数えて $K^\star$ 番目と，大きい方から数えて $K^\star$ 番目の値が信頼限界値である [Gardner and Altman 1989]。

Gardner は，$n \leq 50$ までの場合の，$\alpha = 0.10, 0.05, 0.01$ については数表を与えている。$n > 50$ のときは，

$$K^\star = \frac{n(n+1)}{4} - N_{1-\alpha/2} \times \sqrt{\frac{n(n+1)(2n+1)}{24}} \tag{2}$$

で，近似できるとしている。

表 3 に，(2) 式による近似効率を検討した結果を示す。$K^\star_{approx}$ が，(2) 式によるもので，approx level は，それが実際に与える信頼率である。

$n = 15$ の時点でも，信頼率が 90.0, 95.0, 99.0 % の場合は，近似効率はかなり良いことが分かる。しかし，信頼率が 99.9% の場合については，かならずしもよいとはいえず，これは $n = 100$ になっても改善されていない。

対応のあるデータ 50 組の差が平均値 0，分散 1 になるような 10 セットの仮想データを用いて実際に得られる信頼限界を表 4 に示す。$K^\star_{exact}$ と $K^\star_{approx}$ の大きさはある意味では小さいかもしれないが，信頼限界値にはかなりの差があるようにみえる (誇張されていると思われる)。

なお，付録 付録 A に示したプログラムを若干修正して，$n = 240$ までの表 3 に示したような完全な表を求めるのには 250 秒を要しただけである。

4.2 対応のないデータの中央値の差の信頼区間

必ずしも正規分布にしたがわない 2 つの母集団から，標本の大きさ n_1, n_2 の データを抽出し，それぞれ $x_1, x_2, \ldots, x_{n_1}$ ，および $x'_1, x'_2, \ldots, x'_{n_2}$ とする。母集団の中央値 (平均値) および差の信頼区間は，データのあらゆる組合せ ($n_1 \times n_2$ 個) の差 $x_i - x'_j, (i = 1, 2, \ldots, n_1; j = 1, 2, \ldots, n_2)$ の中央値から推定される。

ほぼ $100(1-\alpha)\%$ の信頼区間を求めるには，Mann-Whitney 統計量の $100\alpha/2\%$ 点を $K = W_{\alpha/2}$ とすると，$n_1 \times n_2$ 個の観測値間の差の，小さい方から数えて K 番目と，大きい方から数えて K 番目の値が信頼限界値である [Gardner and Altman 1989]。

Gardner は，$50 \geq n_1 \geq n_2$ までの場合の，$\alpha = 0.10, 0.05, 0.01$ については数表を与えている。n_1, $n_2 > 25$ の

ときは，

$$K = \frac{n_1 n_2}{2} - N_{1-\alpha/2} \times \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}} \tag{3}$$

で，近似できるとしている。

表 5 に，(3) 式による近似効率を検討した結果を示す。$K^{\star}_{approx}$ が，(3) 式によるもので，approx level は，それが実際に与える信頼率である。

$n = 15$ の時点でも，信頼率が 90.0, 95.0, 99.0 % の場合は，近似効率はかなり良いことが分かる。しかし，信頼率が 99.9% の場合については，かならずしもよいとはいえず，これは $n_1 = n_2 = 50$ になっても改善されていない。

母平均値が 0 と 1 で，母分散が共に 1 にしたがう 2 つの母集団から大きさ 50 の標本を取り出し，実際に得られる信頼限界を表 6 に示す。K_{exact} と K_{approx} の大きさはある意味では小さいかもしれないが，信頼限界値にはかなりの差があるようにみえる (誇張されていると思われる)。

なお，付録 B.2 に示したプログラムを若干修正して，$n_1 = n_2 = 60$ までの表 5 に示したような完全な表を求めるのには 115 秒を要しただけである。

4.3　有意確率の多項式近似

計算によって得られる統計表は膨大なので省略するが，例えば，表 7 は得られた統計表を元にして，Kendall の順位相関係数の有意確率を求めるための近似多項式を検討した例である。$11 \leq n \leq 30$ の，$0.1 \leq p \leq 0.0001$ の範囲について，小数点以下 4 桁程度まで正確な値が得られる。

5　まとめ

3 種のノンパラメトリック検定の統計量の分布を，マイクロコンピュータで求めるためのアルゴリズムを考案し，実際にマイクロコンピュータ上で実行可能なプログラムを作成した。

それらは，十分な実行速度を示し，従来統計学の教科書に掲載されていた数値表の範囲を遥かに拡張できた。

また，有意性検定を行う上では正規近似がかなり有効であるが，信頼限界を求める場合には，特に α が小さい場合には，正規近似の妥当性に疑問があることを示した。

参考文献

[Borland International 1991] Borland International (翻訳 ボーランド株式会社): Borland C++ Version 3.0 Programmer's Guide.

[Kernighan and Ritchie 1988] Kernighan, B. W. and Ritchie, D. M.: The C Programming Language. Second Edition, Prentice Hall.
石田晴久 訳: プログラミング言語 C 第 2 版 ANSI 規格準拠, 共立出版株式会社, 東京.

[Gardner and Altman 1989] Gardner, M. J. and Altman, D. G.: Statistics with Confidence — Confidence intervals and statistical guidelines. British Medical Journal, London.

[Siegel 1956] Siegel, S.: Nonparametric Statistics: for the Behabioral Sciences, McGraw Hill Kogakusha, LTD., Japan.

表 1　Wilcoxon の符号順位検定統計量の分布の逐次的な決定

$n=1$			$n=2$			$n=3$			$n=4$		
${}_0S_1$	${}_0S_1$	S_1	${}_0S_2$	${}_1S_2$	S_2	${}_0S_3$	${}_1S_3$	S_3	${}_0S_4$	${}_1S_4$	S_4
0 (1)		0 (1)	0 (1)		0 (1)	0 (1)		0 (1)	0 (1)		0 (1)
	1 (1)	1 (1)	1 (1)		1 (1)	1 (1)		1 (1)	1 (1)		1 (1)
				2 (1)	2 (1)	2 (1)		2 (1)	2 (1)		2 (1)
				3 (1)	3 (1)	3 (1)	3 (1)	3 (2)	3 (2)		3 (2)
							4 (1)	4 (1)	4 (1)	4 (1)	4 (2)
							5 (1)	5 (1)	5 (1)	5 (1)	5 (2)
							6 (1)	6 (1)	6 (1)	6 (1)	6 (2)
										7 (2)	7 (2)
										8 (1)	8 (1)
										9 (1)	9 (1)
										10 (1)	10 (1)

註：カッコ内の数値は，対応する統計量が得られる度数を表す。

表 2　Mann-Whitney の U 検定統計量の分布の逐次的な決定

$n_1=1,\ n_1=1$			$n_1=2,\ n_2=1$			$n_1=2,\ n_2=2$		
${}_0S_0^1$	${}_1S_1^0$	S_1^1	${}_0S_0^2$	${}_1S_1^1$	S_1^2	${}_0S_1^2$	${}_1S_2^1$	S_2^2
0 (1)		0 (1)	0 (1)		0 (1)	0 (1)		0 (1)
	1 (1)	1 (1)		1 (1)	1 (1)	1 (1)		1 (1)
				2 (1)	2 (1)	2 (1)	2 (1)	2 (2)
							3 (1)	3 (1)
							4 (1)	4 (1)

$n_1=3,\ n_2=1$			$n_1=3,\ n_2=2$			$n_1=3,\ n_2=3$		
${}_0S_0^3$	${}_1S_1^2$	S_1^3	${}_0S_1^3$	${}_1S_2^2$	S_2^3	${}_0S_2^3$	${}_1S_3^2$	S_3^3
0 (1)		0 (1)	0 (1)		0 (1)	0 (1)		0 (1)
	1 (1)	1 (1)	1 (1)		1 (1)	1 (1)		1 (1)
	2 (1)	2 (1)	2 (1)	2 (1)	2 (2)	2 (2)		2 (2)
	3 (1)	3 (1)	3 (1)	3 (1)	3 (2)	3 (2)	3 (1)	3 (3)
				4 (2)	4 (2)	4 (2)	4 (1)	4 (3)
				5 (1)	5 (1)	5 (1)	5 (2)	5 (3)
				6 (1)	6 (1)	6 (1)	6 (2)	6 (3)
							7 (2)	7 (2)
							8 (1)	8 (1)
							9 (1)	9 (1)

註：カッコ内の数値は，対応する統計量が得られる度数を表す。

表 3　対応のあるデータの中央値の差の信頼区間の近似効率

n	α	$K^{\star}_{exact}$	exact level	$K^{\star}_{approx}$	approx level
15	0.100	31	0.9053955	32	0.8930054
	0.050	26	0.9520874	26	0.9520874
	0.010	16	0.9916382	15	0.9932861
	0.001	7	0.9991455	3	0.9998169
30	0.100	152	0.9038984	153	0.8996026
	0.050	138	0.9502899	138	0.9502899
	0.010	110	0.9900685	108	0.9912945
	0.001	79	0.9990482	73	0.9994452
50	0.100	467	0.9009141	468	0.8988901
	0.050	435	0.9505536	435	0.9505536
	0.010	374	0.9900357	371	0.9908830
	0.001	305	0.9990129	297	0.9992738
70	0.100	961	0.9009166	962	0.8996990
	0.050	908	0.9504826	908	0.9504826
	0.010	806	0.9900558	803	0.9905684
	0.001	690	0.9990073	681	0.9991890
100	0.100	2046	0.9006407	2047	0.8999276
	0.050	1956	0.9500762	1955	0.9504815
	0.010	1780	0.9900473	1776	0.9904465
	0.001	1579	0.9990110	1568	0.9991417

表 4　シミュレーションによる信頼限界の精度，$n = 50,\ \alpha = 0.001$

exact	approx	using t distribution
[–0.48996, 0.32791]	[–0.50266, 0.35172]	[–0.56761, 0.36903]
[–0.67661, 0.20633]	[–0.68588, 0.21878]	[–0.68179, 0.20731]
[–0.40927, 0.59146]	[–0.41891, 0.60550]	[–0.40038, 0.58375]
[–0.45749, 0.57474]	[–0.48349, 0.58987]	[–0.44534, 0.54031]
[–0.52707, 0.52908]	[–0.55161, 0.54715]	[–0.54506, 0.52538]
[–0.46164, 0.41617]	[–0.47592, 0.43997]	[–0.45257, 0.37200]
[–0.48386, 0.65494]	[–0.49765, 0.66849]	[–0.50840, 0.62068]
[–0.49631, 0.57840]	[–0.50900, 0.59988]	[–0.45112, 0.57000]
[–0.33871, 0.70901]	[–0.35532, 0.72476]	[–0.36110, 0.74033]
[–0.50742, 0.47635]	[–0.52402, 0.48843]	[–0.48645, 0.45978]

表 5　対応のないデータの中央値の差の信頼区間の近似効率

n_1	n_2	α	K_{exact}	exact level	K_{approx}	approx level
20	20	0.100	139	0.9035004	140	0.8978391
		0.050	128	0.9509097	128	0.9509097
		0.010	106	0.9905164	105	0.9912881
		0.001	82	0.9990666	79	0.9993324
25	25	0.100	228	0.9006068	228	0.9006068
		0.050	212	0.9505749	212	0.9505749
		0.010	181	0.9904314	180	0.9909868
		0.001	147	0.9990426	143	0.9992978
30	30	0.100	339	0.9004952	339	0.9004952
		0.050	318	0.9503800	318	0.9503800
		0.010	277	0.9903963	276	0.9908194
		0.001	232	0.9990252	228	0.9992261
40	40	0.100	629	0.9009302	630	0.8989247
		0.050	597	0.9501613	597	0.9501613
		0.010	534	0.9901924	533	0.9904716
		0.001	463	0.9990249	459	0.9991576
50	50	0.100	1011	0.9011339	1012	0.8997055
		0.050	966	0.9504523	966	0.9504523
		0.010	878	0.9901883	877	0.9903877
		0.001	778	0.9990231	773	0.9991416

表 6　シミュレーションによる信頼限界の精度，$n_1 = n_2 = 50$, $\alpha = 0.001$

exact	approx	using t distribution
[0.15161, 1.8093]	[0.14087, 1.8181]	[0.21284, 1.7358]
[0.31909, 1.6589]	[0.31274, 1.6619]	[0.29815, 1.6242]
[0.20679, 1.7454]	[0.19995, 1.7566]	[0.25797, 1.7181]
[0.05884, 1.3606]	[0.05005, 1.3650]	[0.02290, 1.2870]
[–0.01148, 1.4656]	[–0.02515, 1.4788]	[–0.01622, 1.4198]
[0.44409, 1.7561]	[0.44019, 1.7634]	[0.46292, 1.7544]
[–0.01294, 1.5554]	[–0.01831, 1.5681]	[0.01438, 1.4967]
[0.25610, 1.5383]	[0.24878, 1.5437]	[0.26666, 1.5382]
[0.48413, 1.8142]	[0.46704, 1.8201]	[0.46493, 1.8337]
[0.41626, 1.8367]	[0.40845, 1.8469]	[0.51567, 1.8367]

表 7　Kendall の検定統計量の有意確率を求めるための近似多項式の係数 ($11 \leq n \leq 30$)

	$n = 11$†	$n = 12$	$n = 13$	$n = 14$	$n = 15$
S'‡	23, 45	26, 52	28, 58	33, 65	35, 73
b_0*	1.503940	1.515754	1.460504	1.177191	1.167207
b_1	–0.1359426	–0.1231230	–0.1041777	–0.05629778	–0.05069670
b_2	0.004674278	0.003864800	0.002841422	0.0002463877	0.0002150366
b_3	–7.301155e–05	–5.706168e–05	–3.558404e–05	3.378311e–05	2.435424e–05
b_4	4.540205e–07	3.729703e–07	1.836161e–07	–8.467880e–07	–5.557776e–07
b_5	–3.272471e–10	–7.031678e–10	–1.700326e–10	8.163661e–09	4.854975e–09
b_6				–2.907612e–11	–1.564736e–11
	$n = 16$	$n = 17$	$n = 18$	$n = 19$	$n = 20$
S'	38, 80	42, 88	45, 97	49, 105	52, 114
b_0	1.159704	1.183732	1.192764	1.202140	1.199940
b_1	–0.04593190	–0.04485498	–0.04252632	–0.04035180	–0.03756195
b_2	0.0001796429	0.0002717131	0.0002857333	0.0002839737	0.0002536144
b_3	1.828052e–05	1.140808e–05	7.704481e–06	5.347263e–06	4.071266e–06
b_4	–3.811112e–07	–2.370694e–07	–1.565822e–07	–1.070478e–07	–7.726024e–08
b_5	3.040044e–09	1.768366e–09	1.088545e–09	6.977657e–10	4.691526e–10
b_6	–8.947997e–12	–4.802063e–12	–2.729196e–12	–1.626573e–12	–1.015822e–12
	$n = 21$	$n = 22$	$n = 23$	$n = 24$	$n = 25$
S'	56, 122	61, 131	65, 141	68, 150	72, 160
b_0	1.200172	1.222640	1.234052	1.218155	1.222876
b_1	–0.03516880	–0.03449131	–0.03318803	–0.03039397	–0.02900124
b_2	0.0002277630	0.0002452178	0.0002399292	0.0001948879	0.0001828382
b_3	3.159718e–06	1.902975e–06	1.216150e–06	1.208301e–06	8.817594e–07
b_4	–5.708088e–08	–3.814658e–08	–2.672402e–08	–2.217493e–08	–1.657643e–08
b_5	3.246272e–10	2.089505e–10	1.402260e–10	1.075696e–10	7.664610e–11
b_6	–6.567383e–13	–3.991868e–13	–2.527028e–13	–1.812424e–13	–1.220896e–13
	$n = 26$	$n = 27$	$n = 28$	$n = 29$	$n = 30$
S'	77, 169	81, 179	86, 190	90, 120	95, 209
b_0	1.229405	1.228482	1.243629	1.238030	1.235213
b_1	–0.02778113	–0.02632078	–0.02572177	–0.02424292	–0.02297650
b_2	0.0001720210	0.0001556983	0.0001553559	0.0001374103	0.0001230170
b_3	6.408912e–07	5.189508e–07	2.832627e–07	2.653426e–07	2.416536e–07
b_4	–1.256498e–08	–9.911652e–09	–7.109538e–09	–5.896788e–09	–4.908616e–09
b_5	5.562990e–11	4.166855e–11	2.926383e–11	2.292019e–11	1.811121e–11
b_6	–8.413811e–14	–5.969927e–14	–4.014045e–14	–2.981414e–14	–2.242771e–14

†: ケース数。‡: 近似多項式が有効である S' の範囲。*: b_i は 近似多項式の i 次の係数。
表中の “e–xx” は，10^{-xx} を表す。

付録 A　Wilcoxon の符号順位検定における統計量の分布を求めるプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <math.h>

void get_p(int pair)
{
  int i, n, sum, limit;
  long double *table, demom, total = 0;

  limit = (pair+1)*pair/2+1;
  table = farcalloc(limit, sizeof(long double));
  if (table == NULL) {
    exit(1);
  }
  table[0] = 1; /* n = 0 の時の状態セット */
  for (n = 1, sum = 0; n <= pair; sum += n++) {
    for (i = sum; i >= 0; i--) {
      table[i+n] += table[i];
    }
  }
  demom = pow(2.0, -pair);
  for (i = 0; i < limit; i++) {
    total += (double)table[i]*demom;
    printf("%4d: %20.16Lg %#20.16Lg\n", i, table[i], total);
  }
  farfree(table);
}

int main(void)
{
  int i;

  printf("pair = ");
  scanf("%i", &i);
  if (i <= 68) {
    printf("\n\nn = %i\n\n", i);
    get_p(i);
```

```
    return 0;
  }
  else {
    return 1;
  }
}
```

付録 B　Mann–Whitney の U 検定における統計量の分布を求めるプログラム

B.1　再帰版

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double *table;

void increment(double *q, int n)
{
  while(n-- >= 0)
    (*q++)++;
}

void u(int n, int m, int begin, int end)
{
  if (n == 1 || m == 1) {
    increment(table+begin, end-begin);
  }
  else {
    u(n-1, m, begin, begin+(n-1)*m);
    u(n, m-1, end-n*(m-1), end);
  }
}

double combination(int n, int i)
{
  int j;
  double retv;
  if (i < 0 || i > n || n < 0) {
    exit(1);
  }
  retv = 1.0;
  i = min(i, n-i);
  for (j = 0; j < i; j++)
    retv *= (double)(n-j)/(double)(j+1);
  return retv;
}
```

```
int main(void)
{
  int n1, n2, i;
  double total = 0.0, denom;
  printf("n1,n2 = ");
  scanf("%i,%i", &n1, &n2);
  if (n1 > 12 || n2 > 12) {
    exit(1);
  }
  table = calloc(n1*n2+1, sizeof(double));
  if (table == NULL) {
    exit(1);
  }
  u(n1, n2, 0, n1*n2);
  denom = 1.0/combination(n1+n2, n2);
  for (i = 0; i <= n1*n2; i++) {
    total += table[i];
    printf("%3i: %6g %#20.16f\n", i, table[i], total*denom);
  }
  return 0;
}
```

B.2 非再帰版

```
/* ラージモデルでコンパイル */
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <math.h>

#define LIMIT 27

double combination(int n, int i)
{
  int j;
  double retv;
  if (i < 0 || i > n || n < 0) {
    exit(1);
  }
  retv = 1.0;
  i = min(i, n-i);
  for (j = 0; j < i; j++) {
    retv *= (double)(n-j)/(double)(j+1);
  }
  return retv;
}

int main(void)
{
  double huge * huge * huge *u;
  int n1, n2, i;

  u = farcalloc(LIMIT+1, sizeof(double huge * huge *));
  if (u == NULL) {
    exit(1);
  }
  for (i = 0; i <= LIMIT; i++) {
    u[i] = farcalloc(i+1, sizeof(double huge *));
    if (u[i] == NULL) {
      exit(1);
    }
  }

  for (n1 = 1; n1 <= LIMIT; n1++) {
```

```
    for (n2 = 0; n2 <= n1; n2++) {
      u[n1][n2] = farcalloc(n1*n2+1, sizeof(double));
      if (u[n1][n2] == NULL) {
        exit(1);
      }
    }
    u[n1][0][0] = 1;
    for (n2 = 1; n2 <= n1; n2++) {
      double denom = 0.0, total = 0.0;
      int i, m1, m2, num, d_start;
      printf("\n\n n1 = %i   n2 = %i\n\n", n1, n2);
      m1 = max(n1, n2-1);
      m2 = min(n1, n2-1);
      num = m1*m2+1;
      for (i = 0; i < num; i++) {
        u[n1][n2][i] = u[m1][m2][i];
      }
      m1 = max(n1-1, n2);
      m2 = min(n1-1, n2);
      num = m1*m2+1;
      d_start = n1*n2-num+1;
      for (i = 0; i < num; i++) {
        u[n1][n2][d_start+i] += u[m1][m2][i];
      }
      denom = 1.0/combination(n1+n2, n2);
      num = n1*n2+1;
      for (i = 0; i < num; i++) {
        double x;
        total += x = u[n1][n2][i];
        printf("%3i: %14g %20.16f\n", i, x, total*denom);
      }
    }
  }
  return 0;
}
```

付録 C　Kendall の 順位相関係数の分布を求めるプログラム

```
/* ラージモデルでコンパイル */
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <math.h>

#define N 20

long double permutation(int n)
{
  int i;
  long double retv;
  retv = 1.0;
  for (i = 2; i <= n; i++) {
    retv *= (long double)i;
  }
  return retv;
}

int main(void)
{
  long double *t, sum, cum;
  double kendall;
  int n, n_prev, max_prev, max_now, i, j, denom;
  if ((t = farcalloc(N*(N-1)/2+1, sizeof(long double))) == NULL) exit(1);
  t[0] = t[1] = 1;
  for (n = 3; n <= N; n++) {
    n_prev = n-1;
    max_prev = n_prev*(n_prev-1)/2;
    for (i = max_prev; i >= 0; i--) {
      for (j = n-1; j > 0; j--)
        t[i+j] += t[i];
    }
    denom = max_now = n*(n-1)/2;
    sum = 0.0;
    for (i = 0; i <= max_now; i++) {
      sum += t[i];
    }
    if (sum != permutation(n)) {
      printf("sum = %20.20Lg   expected = %20.20Lg\n", sum, permutation(n));
```

```
    exit(1);
   }
   cum = 0.0;
   printf("\n\n Kendall Rank Correlation Coefficient  N = %i\n\n"
        "%5s%11s%16s%35s\n",
        n, "S", "τ", "Cum. p", "Freq.");
   for (i = 0; i <= max_now; i++) {
    kendall = 1.0−2.0∗i/denom;
    cum += t[i];
    if (kendall >= 0.0) {
     printf("%5i: %#12.7f  %#14.12Lf %30.30Lg\n",
          denom−i∗2, kendall, cum/sum, t[i]);
    }
    else {
     break;
    }
   }
  }
  return 0;
}
```