

上側 $100\alpha\%$ 点計算ライブラリ

C 言語版

杉浦 洋

2018/04/05

1 基本的な使い方

プログラムはソースファイル・ライブラリとして提供される．コンパイルには c99 規格かその上位互換の C コンパイラ (GCC, Clang, Intel C++ Compiler など) が必要である．関数 `tgamma`, `lgamma`, `expm1`, `logp1` を使用するからである．

1.1 使う前のチェック

ダウンロードした zip ファイルを解凍してできたフォルダの中に SincStatistics というフォルダがある．その中に `main.c` がある．それを SincStatistics 内の全てのファイルと共にコンパイル・実行することにより，ライブラリに用意された 20 個の表配列出力関数が全て計算され，チェックされる．正常ならば次の出力が得られ，あなたの使用している C 言語処理系がこのライブラリに適合しているかを確認できる．

```
-----Check zTab-----
最大誤差 = 1.332268e-15    最大残差 = 1.110223e-16
-----Check chi2Tab-----
最大誤差 = 1.332268e-14    最大残差 = 2.220446e-16
-----Check tTab-----
最大誤差 = 4.884981e-15    最大残差 = 2.220446e-16
-----Check FTab-----
最大誤差 = 1.953993e-14    最大残差 = 6.661338e-16
-----Check aTab-----
最大誤差 = 8.881784e-15    最大残差 = 4.440892e-16
-----Check taTab-----
最大誤差 = 1.554312e-14    最大残差 = 7.771561e-16
-----Check b1Tab-----
最大誤差 = 3.552714e-15    最大残差 = 4.440892e-16
-----Check b2Tab-----
最大誤差 = 2.664535e-15    最大残差 = 2.220446e-16
-----Check tb1Tab-----
最大誤差 = 4.440892e-15    最大残差 = 2.220446e-16
-----Check tb2Tab-----
最大誤差 = 2.664535e-15    最大残差 = 4.440892e-16
-----Check d1Tab-----
最大誤差 = 1.154632e-14    最大残差 = 4.440892e-16
-----Check d2Tab-----
最大誤差 = 1.509903e-13    最大残差 = 7.771561e-16
-----Check td1Tab-----
最大誤差 = 1.465494e-14    最大残差 = 8.881784e-16
-----Check td2Tab-----
最大誤差 = 1.465494e-14    最大残差 = 5.551115e-16
-----Check d3Tab-----
最大誤差 = 1.021405e-14    最大残差 = 2.220446e-16
-----Check td3Tab-----
最大誤差 = 6.439294e-15    最大残差 = 2.220446e-16
-----Check bbarTab-----
最大誤差 = 2.815526e-13    最大残差 = 0.000000e+00
-----Check cbarTab-----
最大誤差 = 1.065814e-14    最大残差 = 0.000000e+00
```

```

-----Check bbar32Tab-----
最大誤差 = 1.350031e-13      最大残差 = 6.661338e-16
-----Check cbar32Tab-----
最大誤差 = 2.664535e-15      最大残差 = 1.110223e-16
Program ended with exit code: 0

```

この出力で最大誤差は、今計算した表と Mathematica により計算された表データとで、対応した要素を比べた差の最大値である。最大残差は計算結果の各数値を、それを規定する方程式に代入した残差の最大絶対値である。

1.2 ライブラリのインクルード

ライブラリを使った標準的なプログラムと実行結果は次の様なものである。

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "SincStatistics.h"
int main(void)
{
    printf("value = %e\n",TD1CDF(0.1,3,4));
    return 0;
};
.....
value = 2.059189e-01
Program ended with exit code: 0

```

5 行目の include 文でライブラリ関数のヘッダーファイル SincStatistics.h を組み込む。プログラムの下から 3 行目でライブラリ関数 TD1CDF を使っている。

SincStatistics.h は 9 個のヘッダーファイル Chap1.h, Chap2.h, Chap3.h, Chap4.h, Chap5.h, Chap6.h, Chap6.h, Chap7.h, Chap8.h, Chap9.h を include 文で統合したものである。また, Chap7.h は 3 つのデータファイル HayterData.dat, LeeSpurrierData.dat, WilliamsData.dat をインクルードしている。これらはコンパイル時に同じフォルダーに含まれている必要がある。

最後に, Check.h は表関数のチェックのための関数を納めたものである。それらが必要でない限り, 組み込む必要はない。

2 ライブラリの内容

『多重比較法の理論と数値計算』の第 2 章から第 5 章で解説される, 統計解析に用いられる分布の上側 $100\alpha\%$ 点を求める C 関数を作成した。

それを用いて, 閉検定手順で用いられる分布の, 上側 $100\alpha(M, \ell)\%$ 点 ($k \geq M \geq 2, 2 \leq \ell \leq M, \ell \neq M-1$) を要素とする T1 型 2 次元配列 (表 1), 上側 $100\alpha\%$ 点 ($1 \leq \ell \leq k-1$) を要素とする T2 型 1 次元配列 (表 2), そして上側 $100\alpha\%$ 点 ($2 \leq \ell \leq k$) を要素とする T3 型 1 次元配列 (表 3) を出力する C 関数を作成した。ただし, $\alpha(M, \ell) \equiv 1 - (1 - \alpha)^{(\ell/M)}$ である。また, 出力された配列を定められた表形式で標準出力 (通常はコンピュータ・ディスプレイ) に出力する簡単な C 関数を作成した。

計算に階層確率 (Level Probability) を用いる分布関数がある．ライブラリには階層確率を計算する C 関数が附属している．

表 1 T1 型 2 次元配列 : { 上側 $100\alpha(M, \ell)\%$ 点 } ($k = 8$ のとき)

$M \setminus \ell$	2	3	4	5	6	7	8
8	5.144	5.781	6.046	6.172	6.233	◇	6.256
7	4.902	5.519	5.770	5.887	◇	5.956	
6	4.626	5.219	5.455	◇	5.604		
5	4.303	4.868	◇	5.177			
4	3.914	◇	4.637				
3	◇	3.905					
2	2.756						

表 2 T2 型 1 次元配列 : { 上側 $100\alpha\%$ 点 } ($k = 8$ のとき)

$k \setminus \ell$	7	6	5	4	3	2	1
8	6.256	6.241	6.233	6.172	6.046	5.781	5.144

表 3 T3 型 1 次元配列 : { 上側 $100\alpha\%$ 点 } ($k = 8$ のとき)

$k \setminus \ell$	8	7	6	5	4	3	2
8	6.256	6.241	6.233	6.172	6.046	5.781	5.144

2.1 上側 $100\alpha\%$ 点を求める C 関数

以下の表 4 に上側 $100\alpha\%$ 点を求める C 関数を示す．

表 4 上側 100 α % 点を計算する C 関数

ページ	上側 100 α % 点	C 関数
33	$ta(k, m; \alpha)$	<code>double taFun(int k,int m,double al)</code>
34	$a(k; \alpha)$	<code>double aFun(int k,double al)</code>
78	$tb_1(k, \mathbf{n}; \alpha)$	<code>double tb1Fun(int k,int *ns,double al)</code>
78	$tb_2(k, \mathbf{n}; \alpha)$	<code>double tb2Fun(int k,int *ns,double al)</code>
81	$b_1(k, \boldsymbol{\lambda}; \alpha)$	<code>double b1Fun(int k,double *lms,double al)</code>
81	$b_2(k, \boldsymbol{\lambda}; \alpha)$	<code>double b2Fun(int k,double *lms,double al)</code>
126	$\bar{b}^2(k, \boldsymbol{\lambda}, m; \alpha)$	<code>double bbar2Fun(int k,double *lms,int m,double al)</code>
129	$\bar{c}^2(k, \boldsymbol{\lambda}; \alpha)$	<code>double cbar2Fun(int k,double *lms,double al)</code>
133	$td_1(k, m; \alpha)$	<code>double td1Fun(int k,int m,double al)</code>
135	$d_1(k; \alpha)$	<code>double d1Fun(int k,double al)</code>
156	$td_2^*(k, m; \alpha)$	<code>double td2Fun(int k,int m,double al)</code>
158	$d_2^*(k; \alpha)$	<code>double d2Fun(int k,double al)</code>

この表で「ページ」は、『多重比較法の理論と数値計算』において、その関数が定義されているページを示す。「上側 100 α % 点」の母数 \mathbf{n} , $\boldsymbol{\lambda}$ は

$$\begin{aligned} \mathbf{n} &\equiv (n_1, n_2, \dots, n_k), \\ \boldsymbol{\lambda} &\equiv (\lambda_1, \lambda_2, \dots, \lambda_k) \equiv (n_1/n, n_2/n, \dots, n_k/n), \quad n \equiv \sum_{i=1}^k n_i \end{aligned} \quad (1)$$

である。

C 関数の入力引数と母数の対応は

$$\begin{aligned} \text{int } k &= k, \quad \text{int } l = \ell, \quad \text{int } m = m, \quad \text{int } *ns = \mathbf{n} \\ \text{double } al &= \alpha \quad \text{double } *lms = \boldsymbol{\lambda} \end{aligned} \quad (2)$$

である。 m は通常 $m = n - k$ として用いられるが、ここでは独立した引数である。

C 関数の出力は倍精度の上側 100 α % 点である。例えば、母数 $k = 5$, $m = 100$ に対する、表 4 先頭のテューキー・クレーマーの方法の上側 1% 点 $t = ta(5, 100; 0.01)$ は

```
int k = 5, m = 100; double al = 0.01;
double t;
t = taFun(k,m,al);
```

で計算できる。

C 関数の入力引数 k, m, al には以下の制限を設ける。 k, l の制限は計算速度向上のためである。また、 al の制限は精度保証のためである。この制限内で、C 関数の出力は小数点以下 6 桁まで正しいことが保証される。

$$\begin{aligned} 2 &\leq k \leq 20, \quad 2 \leq l \leq 20, \\ m &\geq 1 \text{ または } m = -1, \\ 10^{-4} &\leq al < \frac{1}{2} \end{aligned} \quad (3)$$

入力引数 $m = -1$ は $m = \infty$ と解釈され、対応する漸近分布に関する計算値が出力される。すなわち、 $m = -1$

のとき

$$\begin{aligned}
\text{taFun}(\mathbf{k}, \mathbf{m}, \mathbf{al}) &= ta(k, \infty; \alpha) &= a(t, k), \\
\text{tb1Fun}(\mathbf{k}, *ns, \mathbf{m}, \mathbf{al}) &= tb_1(k, \mathbf{n}, \infty; \alpha) = b_1(k, \boldsymbol{\lambda}; \alpha), \\
\text{tb2Fun}(\mathbf{k}, *ns, \mathbf{m}, \mathbf{al}) &= tb_2(k, \mathbf{n}, \infty; \alpha) = b_2(k, \boldsymbol{\lambda}; \alpha), \\
\text{bbar2Fun}(\mathbf{ik}, *lms, \mathbf{m}, \mathbf{al}) &= \bar{b}^2(k, \boldsymbol{\lambda}, \infty; \alpha) = \bar{c}^2(k, \boldsymbol{\lambda}; \alpha), \\
\text{td1}(\mathbf{k}, \mathbf{m}, \mathbf{al}) &= td_1(k, \infty; \alpha) &= d_1(k; \alpha), \\
\text{td2}(\mathbf{k}, \mathbf{m}, \mathbf{al}) &= td_2^*(k, \infty; \alpha) &= d_2^*(k; \alpha)
\end{aligned} \tag{4}$$

である．最左辺の C 関数と最右辺に対応する C 関数とは，出力が一致する．

表 4 の 7, 8 番目の C 関数 `bbar2Fun`, `cbar2Fun` は内部で階層確率を計算するため， 2^{k+1} 個の倍精度実数を格納するためのワーキングメモリ約 16×2^k バイトを消費する．それに見合うメモリ容量を確保する必要がある．また，階層確率の計算には時間が掛かる．関数 `bbar2Fun`, `cbar2Fun` を繰り返し計算する際には，一度計算した階層確率を使える限り使い回して，再計算を抑止する必要がある．そのアルゴリズムについては，4.2 節をご覧ください．

2.2 上側 $100\alpha(M, \ell)\%$ 点の 2 次元配列を出力する C 関数

以下の表 5 に，閉検定手順で用いられる分布の上側 $100\alpha(M, \ell)\%$ 点 ($k \geq M \geq 2$, $2 \leq \ell \leq M, \ell \neq M-1$) を要素とする T1 型 2 次元配列 (表 1) を作る C 関数を示す．ただし， $\alpha(M, \ell) \equiv 1 - (1 - \alpha)^{(\ell/M)}$ である．

表 5 上側 $100\alpha(M, \ell)\%$ 点の T1 型 2 次元配列を出力する C 関数

節番号	配列要素 $[M][\ell]$	C 関数
2.1.3	$ta(\ell, m; \alpha(M, \ell))$	<code>double** taTab(int k, int m, double al)</code>
2.1.5	$a(\ell; \alpha(M, \ell))$	<code>double** aTab(int k, double al)</code>
2.1.3	$F_m^{\ell-1}(\alpha(M, \ell))$	<code>double** FTab(int k, int m, double al)</code>
2.3.3	$\chi_{\ell-1}^2(\alpha(M, \ell))$	<code>double** chi2Tab(int k, double al)</code>
5.3.3	$td_1(\ell, m; \alpha(M, \ell))$	<code>double** td1Tab(int k, int m, double al)</code>
5.3.3	$d_1(\ell; \alpha(M, \ell))$	<code>double** d1Tab(int k, double al)</code>
5.3.4	$td_2^*(\ell, m; \alpha(M, \ell))$	<code>double** td2Tab(int k, int m, double al)</code>
5.4.3	$d_2^*(\ell; \alpha(M, \ell))$	<code>double** d2Tab(int k, double al)</code>
5.6.1	$\bar{b}_1^2(\ell, m; \alpha(M, \ell))$	<code>double** bbarTab(int k, int m, double al)</code>
5.6.1	$\bar{c}_1^2(\ell; \alpha(M, \ell))$	<code>double** cbarTab(int k, double al)</code>
5.6.1	$t(m; \alpha(M, \ell))$	<code>double** tTab(int k, int m, double al)</code>
5.6.1	$z(\alpha(M, \ell))$	<code>double** zTab(int k, double al)</code>

この表で「節番号」は、『多重比較法の理論と数値計算』において，その統計量を用いる閉検定手順を解説している節の番号である．C 関数の入力引数と母数の対応は (2) に従う．出力は T1 型 2 次元配列である．

たとえば，表 5 の $ta(\ell, m; \alpha(M, \ell))$ について，母数 $k = 5$, $m = 100$ に対する水準 1% の閉検定手順に用いられる T1 型 2 次元配列 `tab` は，

```

int k = 5, m = 100; double al = 0.01;
double **tab;
tab = taTab(k, m, al);

```

で作られる．配列 `tab` の要素は

$$\text{tab}[M][\ell] = ta(\ell, m; \alpha(M, \ell)) \quad (2 \leq M \leq k, 2 \leq \ell \leq M, \ell \neq M - 1)$$

となる．

C 関数の入力引数 `m = -1` は母数 $m = \infty$ と解釈され，対応する漸近分布に関する配列が作られる．すなわち，`m = -1` のとき

$$\begin{aligned} \text{taTab}(k, m, \text{al}) &= \text{aTab}(k, \text{al}) \\ \text{td1Tab}(k, m, \text{al}) &= \text{d1Tab}(k, \text{al}), \\ \text{td2Tab}(k, m, \text{al}) &= \text{d2Tab}(k, \text{al}) \text{bbar2Tab}(k, m, \text{al}) = \text{cbar2Tab}(k, m, \text{al}), \end{aligned} \tag{5}$$

である．

C 関数の入力引数 `k, m, al` は (3) で制限される．この制限内で C 関数の出力は小数点以下 6 桁まで正しいことが保証される．

2.3 上側 $100\alpha\%$ 点の 1 次元配列を出力する C 関数

以下の表 6 に，閉検定手順で用いられる分布の，上側 $100\alpha\%$ 点 ($1 \leq \ell \leq k - 1$) を要素とする T2 型 1 次元配列 (表 2)，及び上側 $100\alpha\%$ 点 ($2 \leq \ell \leq k$) を要素とする T3 型 1 次元配列 (表 3) を出力する C 関数を示す．

表 6 上側 $100\alpha\%$ 点の T2 型，T3 型の 1 次元配列を作る C 関数

節番号	配列要素 $[\ell]$	T2 型 C 関数
3.3.1	$tb_1^*(\ell, m, n_2/n_1; \alpha)$	<code>double* tb1Tab(int k, int n1, int n2, double al)</code>
3.3.1	$tb_2^*(\ell, m, n_2/n_1; \alpha)$	<code>double* tb2Tab(int k, int n1, int n2, double al)</code>
3.3.2	$b_1^*(\ell, m, n_2/n_1; \alpha)$	<code>double* b1Tab(int k, int n1, int n2, double al)</code>
3.3.2	$b_2^*(\ell, m, n_2/n_1; \alpha)$	<code>double* b2Tab(int k, int n1, int n2, double al)</code>
節番号	配列要素 $[\ell]$	T3 型 C 関数
5.5.1	$td_3(\ell, m, 1; \alpha)$	<code>double* td3Tab(int k, int m, double al)</code>
5.5.1	$d_3(\ell, 1; \alpha)$	<code>double* d3Tab(int k, double al)</code>
5.6.2	$\bar{b}_3^2(\ell, \lambda, m; \alpha)$	<code>double* bbar32Tab(int k, double *lms, int m, double al)</code>
5.6.2	$\bar{c}_3^2(\ell, \lambda; \alpha)$	<code>double* cbar32Tab(int k, double *lms, double al)</code>

この表で「節番号」は、『多重比較法の理論と数値計算』において，その統計量を用いる閉検定手順を解説している節の番号である．C 関数の入力引数と母数の対応は (2) に従う．出力は T2 型，あるいは T3 型の 1 次元配列である．

たとえば，表 6 の $tb_1^*(\ell, m, n_2/n_1; \alpha)$ について，母数 $k = 10$ ， $n_1 = 23$ ， $n_2 = 45$ に対する水準 1% の逐次棄却型検定に用いられる T2 型 1 次元配列 `tab` は，

```
int k = 10, n1 = 23, n2 = 45; double al = 0.01;
double *tab;
tab = tb1Tab(k, n1, n2, al);
```

で作られる。配列 `tab` の要素は

$$\text{tab}[\ell] = tb_1^*(\ell, m, n_2/n_1; \alpha) \quad (1 \leq \ell \leq k-1)$$

である。 $tb_1^*(\ell, m, n_2/n_1; \alpha)$, $tb_2^*(\ell, m, n_2/n_1; \alpha)$ は $m = n_1 + (k-1)n_2 - k$ として計算される。したがって、対応する C 関数 `tb1Tab`, `tb2Tab` は入力引数 `m` を持たない。

また、 $td_3(\ell, m, 1; \alpha)$ について、母数 $k = 10$, $m = 230$ に対する水準 5% のウイリアムス法に用いられる T3 型 1 次元配列 `tab` は、

```
int k = 10, m = 230; double al = 0.05;
double *tab;
tab = td3Tab(k,m,al);
```

で作られる。配列 `tab` の要素は

$$\text{tab}[\ell] = td_3(\ell, m, 1; \alpha) \quad (2 \leq \ell \leq k)$$

である。この関数は現在、 $n_2/n_1 = 1$ にしか対応出来ない。

C 関数の入力引数 `m = -1` は母数 $m = \infty$ と解釈され、対応する漸近分布に関する配列が出力される。すなわち、 `m = -1` のとき

$$\begin{aligned} \text{td3Fun}(k, m, al) &= \text{d3Fun}(k, al), \\ \text{bbar32Tab}(k, *lms, m, al) &= \text{cbar32Tab}(k, *lms, al) \end{aligned} \quad (6)$$

である。

C 関数の入力引数 `k, m, al` は (3) で制限される。この制限内で C 関数の出力は小数点以下 4 桁まで正しいことが保証される。

表 6 の 7, 8 番目の C 関数 `bbar32Tab`, `cbar32Tab` は内部で階層確率を計算するため、 2^{k+1} 個の倍精度実数を格納するためのワーキングメモリー約 16×2^k バイトを消費する。それに見合うメモリ容量を確保する必要がある。また、階層確率の計算には時間が掛かる。関数 `bbar32Tab`, `cbar32Tab` を計算する際には、一度計算した階層確率を使える限り使い回して、再計算を抑止する必要がある。そのアルゴリズムについては、4.2 節をご覧ください。

2.4 表出力プログラム

表配列出力関数が出力した配列のサイズ `k` とポインタ `tab` を入力として、指定された形式の表関数を標準出力 (通常はコンピュータ・ディスプレイ) に出力する。表の型、T1, T2, T3 に合わせて、次の 3 種類から選択する。

```
int PrintTabT1(int k, double **tab),
int PrintTabT2(int k, double *tab),
int PrintTabT3(int k, double *tab).
```

例えば、サイズ $k = 10$ の T1 型の表なら、 `PrintTabT1(10, tab)` で次の表示が得られる。

M\l	2	3	4	5	6	7	8	9	10
10	2.319	2.545	2.668	2.751	2.814	2.864	2.905	nan	2.969
9	2.279	2.507	2.631	2.715	2.778	2.828	nan	2.904	

8	2.234	2.464	2.589	2.674	2.737	nan	2.828
7	2.182	2.415	2.541	2.626	nan	2.740	
6	2.121	2.357	2.484	nan	2.634		
5	2.047	2.287	nan	2.502			
4	1.955	nan	2.329				
3	nan	2.081					
2	1.645						

3 階層確率の計算

階層確率の定義については、『多重比較法の理論と数値計算』の 5.2 節をご覧ください。また、アルゴリズムについては、7.4 節に詳しい解説がある。

3.1 階層確立計算関数

与えられた整数 $k \geq 1$ と重みベクトル $\mathbf{w} = (w_1, w_2, \dots, w_k) \in \mathbb{R}_+^k$ に対し、3 次元配列上に階層確率の表

$$P[L, m, s] = P(L, m, (w_s, w_{s+1}, \dots, w_{s+m-1})), \\ 1 \leq L \leq m \leq k, 1 \leq s \leq k - m + 1$$

を計算する C 関数

```
double*** MakeTableP(int k, double *ws)
```

を作成した。整数 $k = k \geq 1$ と double 型の配列 $\mathbf{ws}[k] = \{w_1, w_2, \dots, w_k\}$ を入力し、

```
P = MakeTableP(k, *ws);
```

により、トリプルポインタ P の先に 3 次元配列が出力される、その要素は、

$$P[L][m][s] = P(L, m, (w_s, w_{s+1}, \dots, w_{s+m-1})), \\ 1 \leq L \leq m \leq k, 1 \leq s \leq k - m + 1$$

である。制約 $1 \leq L \leq m \leq k, 1 \leq s \leq k - m + 1$ を満たさないインデックス L, m, s で配列 P をアクセスすると範囲外アクセスになる。

計算のために要素数 2^{k+1} の double 型作業配列 Q が生成されるので、それに見合うメモリ容量 (16×2^k バイト) が必要である。また、計算時間も 2^{k+1} に比例する。 $k = 10$ の実行時間 s_{10} 秒を測定しておけば、任意の k における実行時間 $\cong 2^{k-10} s_{10}$ 秒が予想できる。筆者のパソコンでは (CPU 3.2GHz) では $s_{10} = 0.02$ 程度であった。

3.2 表関数 P の再利用

サイズ k の大きい表配列 P を作るためには、多大のメモリと計算時間を消費する。一度作った表配列 P は可能な限り再利用すべきである。

そのために、P を使用する関数 `bbar2Fun`, `cbar2Fun`, `bbar32Tab`, `cbar32Tab` などでは、次の様な制御

を行う．グローバル変数 `int LvPrSinck`, `double ***LevPrSincP`, `double *LevPrSinclms` を定義し，最初にこれらの関数が呼ばれるとき，`MakeTableP(k,lms)` を

```
LevPrSincP = MakeTableP(k,lms);  
LevPrSinck = k;  
LevPrSinclms = lms;
```

のように使う．以下，簡単のため 3 式の左辺の変数を `P0`, `k0`, `lms0` と略記する．

直接表配列 `P` を作らず，等価な配列を `P0` として作る．また作ったときのパラメタを `k0`, `lms0` に記録する．`P` は常に `P0` をコピーして使う．

上記の関数 `bbar2Fun`, `cbar2Fun`, `bbar32Tab`, `cbar32Tab` などが呼ばれるのが 2 度目以降のとき， $k \leq k0$ かつ `lms = Take(lms0,k)` なら `P0` を再計算しない．そうでないときは，`P0` を新しいパラメタで再計算し，`k0`, `lms0` を更新する．ここで，`Take(lms0,k)` はリスト `lms0` の最初の `k` 個の要素からなる部分配列である．

`SincStatistics.nb` の関数セルを実行すると，`k0 = 0` にリセットされる．これにより，初回の `bbar2Fun`, `cbar2Fun`, `bbar32Tab`, `cbar32Tab` などの実行では，必ず表関数 `P0` が作られる．また，二度目以降に呼ばれたとき，パラメタ `k`, `lms` が再利用可能な `P` を生成すると判断されたときは，その再計算が抑止される．