統計計算

ポリ・ガンマ関数のC言語，およびFortran 77言語による算譜

株式会社リコー　ソフトウェア事業部 石岡 恒憲

Polygamma Functions Programming in C and Fortran77

Tsunenori Ishioka

1　はじめに

ガンマ関数$\Gamma(\cdot)$，ポリ・ガンマ関数$\psi^{(k)}(\cdot)$，不完全ガンマ関数比$P(\cdot,\cdot)$などのガンマ関数群は，破壊や寿命を取扱う信頼性工学，医学，疫学などの分野では，極めて頻繁に用いられる関数である．たとえば装置寿命，材料の静強度，腐食進行の深さなどによく適合する分布として，現在ワイブル分布が最も一般的に用いられているが，2母数ワイブル分布

$$F(x) = 1 - \exp[-(x/\eta)^{\beta}] \tag{1}$$

の平均値μ，および分散σ^2は，

$$\begin{aligned} \mu &= \eta\Gamma(1+1/\beta), && (2)\\ \sigma^2 &= \eta^2[\Gamma(1+2/\beta) - \{\Gamma(1+1/\beta)\}^2] && (3) \end{aligned}$$

で与えられ，また最尤推定値を$\hat{\beta},\ \hat{\eta}$としたとき，$\sqrt{n}\hat{\beta}$，および$\sqrt{n}\hat{\eta}$の漸近分散，ならびに共分散は

$$\mathrm{V}[\sqrt{n}\hat{\beta}] \simeq \frac{\beta^2}{\psi'(1)}, \tag{4}$$

$$\mathrm{V}[\sqrt{n}\hat{\eta}] \simeq \left(\frac{\eta}{\beta}\right)^2 \frac{\psi'(1)+\psi^2(2)}{\psi'(1)}, \tag{5}$$

$$\mathrm{Cov}[\sqrt{n}\hat{\eta},\ \sqrt{n}\hat{\eta}] \simeq \frac{\psi(2)}{\psi'(1)}\eta \tag{6}$$

で与えられるように，(ポリ)ガンマ関数を含む形で示される．また偶発的な衝撃が一定回数，累積したときに始めて故障が発生する，いわゆるポアソン・ショックモデルとして広く用いられているガンマ分布

$$\begin{aligned} F(x) &= \frac{\lambda}{\Gamma(\alpha)}\int_0^x (\lambda t)^{\alpha-1} e^{-\lambda t} dt \\ &= \frac{1}{\Gamma(\alpha)}\int_0^{\lambda x} t^{\alpha-1} e^{-t} dt && (7) \end{aligned}$$

では，累積分布関数値それ自体が不完全ガンマ関数比$P(\alpha,\ \lambda x)$で示される．このようにワイブル分布，およびガンマ分布を統計的に取扱うときに，ガンマ関数群はしばしば登場する．またワイブル分

Key words: gamma function, incomplete gamma function ratio, numerical analysis, free software

布，ガンマ分布以外にも，Sibuya(1979, 1980)にあるように，ディ・ガンマ分布，トリ・ガンマ分布にも現れることを注意しておく．

さてこのうちガンマ関数$\Gamma(\cdot)$，あるいは対数ガンマ関数$\ln\Gamma(\cdot)$は，組込関数として用意されていることが多い．しかしポリ・ガンマ関数や不完全ガンマ関数比が用意されていることはきわめて稀である．特にポリ・ガンマ関数については，著名な数値計算ライブラリであるIMSLライブラリでさえ，ディ・ガンマ関数があるのみである．

もっとも不完全ガンマ関数比については，Didonato(1986)にその計算を高速におこなう算法（アルゴリズム）が紹介されている．また山内(1972)には，別の算法に基づいて作成されたFortranによる算譜がある．しかしDidonato(1986)における算法のインプリメンテーションは決して容易ではない．また山内(1972)の算譜にしても，（この算譜の利用者は決して少なくないと思われるが）任意のパラメータに対して十分な精度を保証しないことに注意すべきである．本来この算譜は$0 < a < 1$に対する数表の値を与えることを意図したものであり，確かに結果の掲載されている$a = 0.1, 0.5, 0.8$に対して十分な精度を与えている．ここで大きなaに対しては

$$P(a, x) = P(a-1, x) - \frac{1}{\Gamma(a)} x^{a-1} e^{-x} \tag{8}$$

なる漸化式を用いるべきであろうが，算譜自体はそうなっていない．このため，例えば$a = 5.0,\ x = 5.0$に対しては本来，正の値であるべきところで，負の値(-0.1190134)を出力するほど誤差が大きくなってしまう．また$0 < a < 1$であっても$x \ll a$であるならば，（山内(1972)の算譜では）被加項数が大きく，多くの計算時間を要する．

しかしながら，Press(1988)には，パラメータの値によってそれぞれ計算の早いLengendreの公式による第1種の不完全ガンマ関数と，Lengendreの連分数展開による第2種の不完全ガンマ関数の2つを使い分け，平均としての計算時間が少なくなるような不完全ガンマ関数比のC言語による算譜が掲載されている．著者自身，コーディング上での不満がないわけではないが，概ね妥当な計算時間で，倍精度を保証することを確認している．そこで本稿では，不完全ガンマ関数比$P(\cdot, \cdot)$のプログラミングには特に言及しないこととする．

一方，ポリ・ガンマ関数$\psi^{(k)}(\cdot)$については，Bowman & Shenton (1988)に詳しい研究があり，Appendix C VIにはその算譜が掲載されている．またAmos (1983)にも，ほぼ同様なアルゴリズムが紹介され，その算譜はACM Algorithms Distribution Serviceより入手することができる．計算時間についていうならば，Amos (1983)の算譜では，オーバー/アンダーフローを起こさない範囲，ならびに精度を保証できる範囲をパラメータの値に応じて計算するため，そこに多くの計算時間を要する．Bowman & Shenton (1988)の算譜では，定義域を$0.01 \leq x \leq 4000.0$に制限し，値の範囲に関する計算を省いているものの，高価(expensive)なベキ乗計算を多用しており，Amos (1983)の算譜以上に計算時間を要する．

そこで本稿では，絶対精度で小数点以下15桁ないし16桁を保証し，ポリ・ガンマ関数値が0に近づく大きなxに対しては，相対精度で倍精度程度の精度をもち，しかも任意のパラメータに対し，計算時間が大きくならないようなポリ・ガンマ関数$\psi^{(k)}(\cdot)$の算譜を示すこととする．シミュレーションなど多大な計算時間を必要とする応用に有効であると思われる．プログラムははじめにC言語にて作成したが，利用者の便宜を考えてFortran77にも移植しておく．

2 ポリ・ガンマ関数

2.1 漸近展開

ガンマ関数は

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt \tag{9}$$

により定義され，その基本性質は

$$\Gamma(x+1) = x\Gamma(x) \tag{10}$$

にある（森口他(1960)）．ガンマ関数の対数微分

$$\psi(x) = \frac{d}{dx}\ln\Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)},\ \text{ディ・ガンマ関数} \tag{11}$$

およびその逐次導関数

$$\psi'(x) = \frac{d}{dx}\psi(x),\ \text{トリ・ガンマ関数} \tag{12}$$

$$\psi''(x) = \frac{d}{dx}\psi'(x),\ \text{テトラ・ガンマ関数} \tag{13}$$

$$\cdots,\ \psi^{(k)}(x),\ \cdots$$

は，総称してポリ・ガンマ関数と呼ばれる．

対数ガンマ関数の漸近展開は，Bernoulli 数 B_{2n}を用いて

$$\begin{aligned}\ln\Gamma(x) \quad &\simeq \quad \left(x - \frac{1}{2}\right)\ln x - x + \frac{1}{2}\ln(2\pi) \\ &\quad + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)x^{2n-1}}\end{aligned} \tag{14}$$

であるから，ポリ・ガンマ関数の漸近展開は

$$\psi^{(k)}(x) \simeq \begin{cases} \ln x - \dfrac{1}{2x} - \displaystyle\sum_{n=1}^{\infty} B_{2n}\frac{(2n-1)!}{(2n)!x^{2n}} & k = 0 \\ (-1)^{k-1}\left[\dfrac{(k-1)!}{x^k} + \dfrac{k!}{2x^{k+1}} + \displaystyle\sum_{n=1}^{\infty} B_{2n}\frac{(2n+k-1)!}{(2n)!x^{2n+k}}\right] & k = 1, 2, \cdots \end{cases} \tag{15}$$

となる．

ここで，(15)式の級数部分を n 項まで加えるとしたとき，与えられた微分の次数 kに対してどの程度の大きさの $x(>1)$を適用したときに，我々が目標とする 15–16 桁の有効桁数が得られるかを求めてみる．(15)式の級数部分を便宜上，$a_1, \cdots, a_n$とおくと，この級数は交差級数で，かつ被加項の絶対値が漸次減少するから，その級数和は第 1 項の大きさ $|a_1|$ として見積もることができる．したがって有効桁数で 15 桁を保証するには，

$$|a_1| \geq 10^{15}|a_n| \tag{16}$$

を満たす最小のxを求めればよい．これをxについて解くと，

$$x \geq \left\{ 10^{15} \frac{2B_{2n}(2n+k-1)!}{B_2(2n)!(k+1)!} \right\}^{1/(2n-2)} \tag{17}$$

となりこの下限が求める値である．ここで例えば$n = 10$とおけば，$k = 0, 1, 2, 3$に対しては$x \geq 9.38, 10.66, 11.88, 13.06$となる．実用上，$k > 3$を用いることは稀であると思われるので，この場合には漸近展開を適用すべきxの下限s（プログラムでは変数`slv`）を(17)式により計算し，$k = 0, 1, 2, 3$に対しては$s = 13.06$で固定とする．これにより全体としての計算速度を減少させることができると考えられる．

さて$x < s$に対しては，$m = [s-x]$として，すなわち$s-x$を越えない最大整数をmとし，$y = x+m+1$に対して上記の漸近展開を適用し，ポリ・ガンマ関数の基本性質

$$\psi^{(k)}(x+1) = \psi^{(k)}(x) + (-1)^k k!/x^{k+1} \tag{18}$$

を逐次繰り返すことによって，所望の結果を得る．

ここで(18)式の計算における桁落ちについて考察してみる．$k = 2, 3, \ldots$に対しては，例えば森口他(1960)にある級数展開

$$\psi^{(k)}(x) = \sum_{j=0}^{\infty} \frac{(-1)^{k+1}k!}{(x+j)^{k+1}} \tag{19}$$

により$(-1)^{k+1}\psi^{(k)}(x) > 0$である．$k = 1$の場合も$\ln\Gamma(x)$が凸であるから，$\psi'(x) > 0$である．したがって$k \geq 1$のとき

$$\psi^{(k)}(x) = \psi^{(k)}(y) - \frac{(-1)^k}{k!} \sum_{j=0}^{m} \frac{1}{(x+j)^{k+1}}, \ y = x+m+1 \tag{20}$$

による計算で実質的な減算はなく，修正項で絶対値が最大になるのは$j = 0$の場合であるから，xが小さいとき$1/x^{k+1}$を$1/(y-m-1+j)^{k+1}$で計算してはならない．この点に注意して$j = m$から0まで加えていけば，Amos(1983)の方法によらずに精度を保つことができる．(Amos(1983)はこのような場合に，ポリ・ガンマ関数を指数近似してこの危険を回避している．)

さて$k = 0$の場合は，$x = 1.5$の近くに零点があり，上の計算法を用いると実質的な減算が行われる．したがってここでは絶対精度で14桁しか保証できない．ちなみに，$\ln\Gamma(1) = \ln\Gamma(2) = 0$の近傍においても，(15)式をそのまま用いるならば，絶対精度で14桁しか保証できないことになる．

なおエラーは以下の通りに処理する．

- (15)式の級数部分の計算では，大きなxに対してアンダーフローを起こす危険がある．この場合，ANSI/IEEE Std 754–1985の規約にしたがった処理系では，IEEE NaNを特別な場合の値として返す．VAX D Formatでは，0.0を返す．
- $x < 0$が入力されたときは（ポリ・ガンマ関数の定義域には負の値を含むにもかかわらず）IEEE NaN (ANSI/IEEE Std 754-1985)，もしくは0.0 (VAX D Format)を返す．これは入力された任意の負の値に対して精度を保証することができないためである．この対処はAmos(1983)やBowman & Shenton(1988)においても同様である．$k < 0$が入力されたときも同様に処理する．

- x が 0 （または負の整数）に一致するときは$\pm\infty$ になるので，始めから別扱いにする．このような場合，ANSI/IEEE Std 754–1985 の規約にしたがった処理系では，IEEE ∞ を特別な場合の値として返す．VAX D Format では，実数の最大値($\approx 1.70 \times 10^{38}$)を返す．

以上，述べたような算法にしたがって得たポリ・ガンマ関数$\psi^{(k)}(x) = \texttt{polygamma(k, x)}$の $k = 0, 1, 2$ のときの出力結果(C, IEEE Double)を表 1 に示す．それぞれ順にディ・ガンマ，トリ・ガンマ，テトラ・ガンマの各関数値である．参考のために数式処理システム Maple V (Char1992 など)で計算した値を併記する．Maple V では，ポリ・ガンマ関数がシステム関数として用意されており，数値計算は任意の桁の整数計算と任意精度の浮動小数点演算で行うことができる．したがって丸め誤差の影響を全く受けないで計算を行うことが可能である．要求精度は Digits で指定する．表 1 より本稿の算譜による計算値は，絶対精度で 15 桁程度まで正しく計算されていることが確認できる．零点の近くである $\psi(1.5) = \texttt{polygamma(0, 1.5)} = 2 - \ln 4 - \gamma$においては有効桁数（相対精度）は 14 桁である．

表 1： x のいくつかの値に対するポリ・ガンマ関数値

```
         digamma; psi(x)          trigamma; psi'(x)        tetragamma; psi''(x)
  x            k=0                      k=1                      k=2
 0.5
       -1.963510026021423e+00    4.934802200544679e+00   -1.682879664423432e+01
       -1.963510026021423        4.934802200544679       -16.82879664423432
 1.0
       -5.772156649015324e-01    1.644934066848226e+00   -2.404113806319188e+00
       -.5772156649015329        1.644934066848226       -2.404113806319189
 1.5
        3.648997397857667e-02    9.348022005446793e-01   -8.287966442343200e-01
       .03648997397857652        .9348022005446793       -.8287966442343200
 2.0
        4.227843350984666e-01    6.449340668482264e-01   -4.041138063191886e-01
        .4227843350984671        .6449340668482264       -.4041138063191886
 3.0
        9.227843350984666e-01    3.949340668482265e-01   -1.541138063191886e-01
        .9227843350984671        .3949340668482264       -.1541138063191886
 4.0
        1.256117668431800e+00    2.838229557371154e-01   -8.003973224511450e-02
        1.256117668431800        .2838229557371153      -.08003973224511450
 5.0
        1.506117668431800e+00    2.213229557371153e-01   -4.878973224511450e-02
        1.506117668431800        .2213229557371153      -.04878973224511450
10.0
        2.251752589066720e+00    1.051663356816857e-01   -1.104983497080207e-02
        2.251752589066721        .1051663356816857      -.01104983497080207
20.0
        2.970523992242149e+00    5.127082293520312e-02   -2.628122402314655e-03
        2.970523992242149       .05127082293520312     -.002628122402314655
50.0
        3.901989673427892e+00    2.020133322669712e-02  -4.080799893375970e-04
        3.901989673427892       .02020133322669713   -.0004080799893375969
```

上から順に著者(C, IEEE Double) の算譜による計算値と数式処理システム Maple V (Digits := 16)で計算した値を示す．

さて算譜は数値計算上，以下の点に留意してある:

- 加えるべき級数の項は，その値の絶対値が小さい方から順に加えてゆく．これにより“積み残し”の危険を回避している．
- (20)式による計算法を用いることで，関数の再帰呼び出しの回数を高々1回とした．関数呼び出しの際は，呼び出し側のステータスと戻り番地がスタックに積まれることになるので，再帰呼び出しのネストが深くなると関数呼び出しだけで多大の計算時間を要することになる．ここではその危険を回避している．

なお算譜は付録に示しておく．

2.2 従来の算譜との比較

著者の算譜に基づくポリ・ガンマ関数値，ならびにその計算時間をAmos(1983), Bowman(1988)のそれと比較する．表1で与えた値に対する計算時間を表2に示す．

これより著者の算譜はAmos(1983)の算譜の2倍以上速く，Bowman(1988)の算譜の10倍程度速いことがわかる．ちなみにSun OS Version 4.1，Sun IV/370ワークステーション稼働下での計算時間である0.07ms/callという数値は，組込関数である対数ガンマ関数`lgamma()`の約2倍である．

表2: 表1で与えた値に対する計算時間の比較

算譜	言語	実数の内部表現	マシン	計算時間(ms/call)
著者(1993)	C	IEEE Double	Sun IV/370	0.07
著者(1993)	C	VAX D Format	VAX 11/780	1.94
著者(1993)	Fortran	VAX D Format	VAX 11/780	1.97
Amos(1983)	Fortran	VAX D Format	VAX 11/780	5.48
Bowman(1988)	Fortran	VAX D Format	VAX 11/780	19.57

次に（トリ・ガンマ，およびテトラ・ガンマ関数値が0に近づく）大きなxと，逆に（全てのポリ・ガンマ関数値が$\pm\infty$に近づく）小さな正数xに対して，3者の算譜に基づくポリ・ガンマ関数値を表3に示す．

表 3： x が正で小さな値，あるいは大きな値に対するポリ・ガンマ関数値の比較

```
x         digamma                 trigamma                 tetragamma
1.0e-10
        -1.0000000000577215e+10  1.0000000000000000e+20  -1.9999999999999995e+30
        -1.0000000000577216e+10  1.0000000000000000e+20  -2.0000000000000000e+30
        -0.1000000000057722e+11  0.1000000000000000e+21  -0.2000000000000000e+31
        -0.1000000000057722e+11  0.9999999999999999e+20  -0.2000000000000000e+31
         ----------------------  ----------------------   ----------------------
         -.1000000000057722e+11   .1000000000000000e+21   -.2000000000000000e+31
1.0e-5
        -1.0000057719921567e+05  1.0000000001644909e+10  -2.0000000000000020e+15
        -1.0000057719921568e+05  1.0000000001644910e+10  -2.0000000000000025e+15
        -0.1000005771992157e+06  0.1000000000164491e+11  -0.2000000000000002e+16
        -0.1000005771992157e+06  0.1000000000164491e+11  -0.2000000000000002e+16
         ----------------------  ----------------------   ----------------------
         -100000.5771992157       .1000000000164491e+11   -.2000000000000002e+16
1.0e-2
        -1.0056088545787081e+02  1.0001621213528739e+04  -2.0000023403988050e+06
        -1.0056088545786859e+02  1.0001621213528296e+04  -2.0000023403986717e+06
        -0.1005608854578686e+03  0.1000162121352830e+05  -0.2000002340398672e+07
        -0.1005608854578687e+03  0.1000162121352831e+05  -0.2000002340398677e+07
        -0.1005608854578687e+03  0.1000162121352831e+05  -0.2000002340398677e+07
         -100.5608854578687       10001.62121352831       -.2000002340398677e+7
1.0e2
         4.6001618527380881e+00  1.0050166663333571e-02  -1.0100499983335001e-04
         4.6001618527380874e+00  1.0050166663333571e-02  -1.0100499983335000e-04
         0.4600161852738087e+01  0.1005016666333357e-01  -0.1010049998333500e-03
         0.4600161852738087e+01  0.1005016666333357e-01  -0.1010049998333500e-03
         0.4600161852738087e+01  0.1005016666333357e-01  -0.1010049998333500e-03
          4.600161852738087      .01005016666333357    -.0001010049998333500
4.0e3
         8.2939246348936937e+00  2.5003125260416665e-04  -6.2515626953124957e-08
         8.2939246348936944e+00  2.5003125260416663e-04  -6.2515626953124959e-08
         0.8293924634893694e+01  0.2500312526041666e-03  -0.6251562695312496e-07
         0.8293924634893694e+01  0.2500312526041667e-03  -0.6251562695312497e-07
         0.8293924634893694e+01  0.2500312526041666e-03  -0.6251562695312496e-07
          8.293924634893694    .0002500312526041666       -.6251562695312496e-7
1.0e5
         1.1512920464961896e+01  1.0000050000166668e-05  -1.0000100000500001e-10
         1.1512920464961895e+01  1.0000050000166667e-05  -1.0000100000500000e-10
         0.1151292046496190e+02  0.1000005000016667e-04  -0.1000010000050000e-09
         0.1151292046496190e+02  0.1000005000016667e-04  -0.1000010000050000e-09
         ----------------------  ----------------------   ----------------------
          11.51292046496190   .00001000005000016667       -.1000010000050000e-9
1.0e10
         2.3025850929890456e+01  1.0000000000500000e-10  -1.0000000000999999e-20
         2.3025850929890457e+01  1.0000000000500000e-10  -1.0000000001000000e-20
         0.2302585092989046e+02  0.1000000000050000e-09  -0.1000000000100000e-19
         0.2302585092989046e+02  0.1000000000050000e-09  -0.1000000000100000e-19
         ----------------------  ----------------------   ----------------------
          23.02585092989046       .1000000000050000e-9    -.1000000000100000e-19
```

上から順に著者(C, IEEE Double)，著者(C, VAX D Format)，著者(Fortran, VAX D Format)，Amos(Fortran, VAX D Format)， Bowman(Fortran, VAX D Format)の算譜による計算値，数式処理システム Maple V (Digits := 16)で計算した値を示す．表中，横線は計算不能（エラーコードを返すこと）を示す．また $x = 4.0 \times 10^3$ を含むのは，この値が Bowman の算譜で計算できる上限であることによる．

この表より，著者の算譜は$x = 1.0 \times 10^{10}$のようなきわめて大きな値に対しても，また$x = 1.0 \times 10^{-10}$のようなきわめて小さい正数に対しても倍精度程度の精度をもつことがわかる．これより実用に際しては著者の算譜で十分であることが見てとれる．特にxがきわめて小さい正数に対して，Amos(1983)の算譜と著者の算譜とでは，その対処が違うにもかかわらず，同じ数値が得られていることは興味深い．

さて“絶対精度の保証でどこまで計算できるか” の一つの実験として，著者の算譜(C, IEEE Double)を用いて$\psi(x) = 0$の区間[1.0, 1.5]における解を2分法(Bisection Method)により小数点以下16桁まで求めてみた．その結果，1.46163 21499 68363 4を得た．またNewton法では初期値を1.5にして計算した結果，1.46163 21499 68362 0を得た．数式処理システムMaple V (Digits := 20)で計算した値は1.46163 21499 68362 3413であるから，いずれの方法でも有効桁数で15–16桁まで正しいことが確認される．

3　おわりに

本稿で示した算譜は恐らく最も基本的で素直な算法に基づいている．したがって数値解析のほんの初歩の知識と，C言語もしくはFortran77言語の理解力があれば，誰でもそのソースコードを読み下すことが可能であろう．手持ちに適当な算譜がなかったり，算譜を作るのが面倒な方にお使いいただければ，著者自身の喜びとするところである．算譜の入手を希望される方は，著者tunenori@src.ricoh.co.jpまでE-mailにて連絡されたい．なおACM Algorithms Distribution Serviceに注文すれば，本稿でも言及したAmos(1983)のポリ・ガンマ関数(Algorithm 610)，および不完全ガンマ関数(Algorithm 654)など多くの算譜を入手することができる．注文用紙は*ACM Trans. on Mathematical Software*の各巻最終号(No. 4)巻末にある．

さて，本稿で紹介した算譜は，いわゆるフリー・ソフトウェアであって，誰でも無料で利用できる．ただしパブリック・ドメイン・ソフトウェアではなく，著作権の主張を放棄していないことに注意されたい．再販（算譜の営利販売）にあたっては書面をもって著者の許可を必要とする．算法自体については，著作権保護の対象ではない．またこの算譜は，（フリー・ソフトウェアの慣例に従い）無保証とする．

謝 辞

本論文を投稿するにあたり，鎌倉稔成先生（中央大学・理工学部）より有益なご意見とご指示をいただきました．また，査読者と編集理事である吉村功先生（東京理科大学・工学部）には，論文の誤りをご指摘いただき，また加筆すべき事項について多くのご教示をいただきました．さらに学会事務局からはAmos(1983)の算譜（マイクロフィッシュ）を貸与いただきました．ここに記して厚くお礼申しあげます．

付録 ポリ ·ガンマ関数の算譜

(a) C 版

```
/*
 * polygamma --- Return the polygamma function {\psi}^k(x)
 *      If k=0,1,2,..., then returns digamma, trigamma,
 *      tetragamma,... function values respectively.
 *      Double precision (VAX D FORMAT 56 bits or IEEE DOUBLE 53 bits)
 *              $Author: tunenori $
 *              $Revision: 1.5 $
 *              $Date: 1993/08/09 11:21:19 $
 *
 * Special cases:
 *      polygamma(k, x) is NaN with signal if k < 0 or x < 0;
 *      polygamma(k, x) is INF with signal if x = 0;
 *      polygamma(k, +-Inf) is NaN with signal;
 *      polygamma(k, NaN) is that NaN with no signal.
 *
 *      Copyright (c) 1993 by RICOH Co., Ltd.
 *
 *      NO WARRANTIES
 *
 *      RICOH DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 *      INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS,
 *      IN NO EVENT SHALL RIOCH BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 *      CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
 *      LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
 *      NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
 *      CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */
#include <math.h>
#ifdef  vax     /* VAX D format */
#include <errno.h>
        extern  int     errno;
        double  infnan(iarg)
                int     iarg ;
        {
                switch(iarg) {
                case      ERANGE: errno = ERANGE; return(HUGE);
                case     -ERANGE: errno = EDOM;   return(-HUGE);
                default:          errno = EDOM;   return(0);
                }
        }
#define SQRHUGE (1.3043817436596712000e+19)
#endif

static  double  brn[] = {1.6666666666666666e-01, 3.3333333333333333e-02,
 2.3809523809523809e-02, 3.3333333333333333e-02, 7.5757575757575757e-02,
 2.5311355311355311e-01, 1.1666666666666667e+00, 7.0921568627450980e+00,
 5.4971177944862155e+01, 5.2912424242424242e+02};
#define BRNTRM  9       /* the elements number of ‘brn[]‘ */

double
polygamma(k, x)
        int     k;      /* the derivative order number */
        double  x;      /* variable */
{
        static  double  zero = 0.0, one = 1.0;
        double  s;      /* return value */
        double  y;      /* minimum value more than ‘slv’, adding ‘x’ to
```

```
                                                                integers */
        double  x2;     /* x * x */
        double  *bp;    /* the pointer to `brn[]` */
        double  pk;     /* k! */
        double  pxk;    /* pxk = pow(x, k+1)    */
        double  slv;    /* sufficient large value applied for asymptotic
                                                               expansion */
        double  f;
        double  polygamma();
        double  log(), pow(), fabs();
        int     n;      /* [slv -x] */
        int     i, j;
        int     i2, isgn;
        int     finite();

#ifndef vax
        if (x != x)     /* x is NaN */
                return (x);
#endif
        if (k < 0 || x < 0 || !finite (x)){     /* k < 0 or x < 0 or x is
                                                                   +-INF */
                /*
                 * DOMAIN error: polygamma(x) return NaN
                 */
#ifdef  vax
                return (infnan(EDOM)); /* NaN */
#else   /* IEEE double */
                return (zero / zero); /* return NaN with signal */
#endif
        }else if (k > 3){
                /*
                 * calculation of `slv'
                 */
                f = 1.0;
                for (i = k + 19; i > 20; i--)
                        f *= (double) i;
                for (i = k + 1; i > 2; i--)
                        f /= (double) i;
                f *= (174611.0 / 55.0); /* B_{20} / B_{2} */
                slv = 6.812921 * pow(f, 1.0 / 18.0);
                if (slv < 13.06)
                        slv = 13.06;
        }else{  /* 1 <= k <= 3 */
                slv = 13.06;
        }

        pk = 1.0;
        for (i = 0; i < k; i++)
                pk *= (double) (i + 1); /* pk = k! */

        if (x == zero){
                /*
                 * SING error: polygamma(x) return infinity
                 */
#ifdef  vax
                return (infnan(ERANGE));        /* INF */
#else   /* IEEE double */
                return (one / zero); /* return INF with signal */
#endif
        }else if (x >= slv){
                /* Adopted `x' to the asymptotic expansion. */
```

```
                s = 0.0;
#ifdef  vax
                if (x > SQRHUGE)         /* x > sqrt(HUGE) */
                        return (infnan(EDOM));  /* NaN */
#endif
                x2 = x * x;
                bp = &brn[BRNTRM];
                isgn = k % 2 ? -1 : 1;
                if (k == 0){
                        /* digamma function */
                        for (i = BRNTRM; i >= 0; i--){
                                i2 = 2 * (i + 1);
                                s += *bp-- / (double) i2 * isgn;
                                s /= x2;
                                isgn *= -1;
                        }
                        s += log(x) - 0.5 / x;
                }else{
                        /* k >= 1; trigamm, tetragamma, ... */
                        for (i = BRNTRM; i >= 0; i--){
                                f = 1.0;
                                i2 = 2 * (i + 1);
                                j = i2 + k - 1;
                                while (j > i2)
                                        f *= (double) j--;
                                s += *bp-- * f * isgn;
                                s /= x2;
                                isgn *= -1;
                        }
                        for (i = 0; i < k; i++)
                                s /= x;
                        pxk = 1.0;
                        for (i = 0; i < k; i++)
                                pxk *= x;       /* pxk = pow(x, k) */

                        s -= pk * 0.5 / pxk / x * isgn;
                        f = pk / (double) k;
                        s -= f / pxk * isgn;
                }
        }else{
                /*
                 * x < slv;
                 * Adopted ‘y’ instead of ‘x’ to the asymptotic expansion,
                 * we calculation the value.
                 */
                n = (int) (slv - x);
                y = (double) n + x + 1.0;
                s = polygamma(k, y);
                isgn = k % 2 ? 1 : -1;
                for (i = 0; i <= n; i++){
                        y -= 1.0;
                        if (fabs(y) < 1.e-3){
                                if (x > 0)
                                        y = x - (double)((int) (x + 0.5));
                                else
                                        y = x - (double)((int) (x - 0.5));
                        }
                        pxk = 1.0;
                        for (j = 0; j < k; j++)
                                pxk *= y;       /* pxk = pow(y, k) */
```

```
#ifdef  vax
                        if (pxk * y == zero)
                                return (infnan(isgn * ERANGE)); /* INF */
#endif
                        s += isgn *  pk / pxk / y;
                }
        }
        return(s);
}
```

(b) Fortran77 版

```
C
C  polygamma --- Returns polygamma function {\psi}^k(x)
C       if k=0, 1, 2,..., then returns digamma, trigamma,
C       tetragamma, ... function values with double presision.
C               $Author: tunenori $
C               $Revision: 1.5 $
C               $Date: 1993/08/09 11:34:19 $
C       Copyright (c) 1993 by RICOH Co., Ltd.
C
C       NO WARRANTIES
C
C       RICOH DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
C       INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS,
C       IN NO EVENT SHALL RICOH BE LIABLE FOR ANY SPECIAL, INDIRECT OR
C       CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
C       LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
C       NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
C       CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
C
        double  precision function      polygamma(k, x)
        implicit real*8 (a-h, o-z)
        double  precision NaN, INF, SQRINF
        parameter       (IBRNTRM = 10)
        dimension       brn(IBRNTRM)
        data    brn /1.6666666666666666d-1, 3.3333333333333333d-2,
     $          2.3809523809523809d-2, 3.3333333333333333d-2,
     $          7.5757575757575757d-2, 2.5311355311355311d-1,
     $          1.1666666666666667d0, 7.0921568627450980d0,
     $          5.4971177944862155d1, 5.2912424242424242d2/
        data    NaN /0.0d0/
        data    INF /1.70141173319264430000d+38/
        data    SQRINF /1.30438174365967120000d+19/
C
C       int     k;      the derivative order number
C       double  x;      variable
C
C       double  s;      return value
C       double  y;      minimum value more than 10, adding ‘x’ to integers
C       double  x2;     x * x
C       double  c;      ‘brn’ * (2 * i + k -1)! / (2 * i)!
C       double  pk;     k!
C       double  pxk;    pxk = pow(x, k+1)
C       double  slv;    sufficient large value applied for asymptotic
                                                                  expansion
C
C       double  polygamma();

        if ((k .lt. 0) .or. (x .lt. 0.0)) then
C               /* DOMAIN error */
C               /* return NaN */
                polygamma = NaN
                return
        else if (k .gt. 3) then
C               /* caluculate of ‘slv’ */
                f = 1.0
                do 5 i = 21, (k + 19)
5                       f = f * dble (i)
                do 7 i = 3, (k + 1)
7                       f = f / dble (i)
                f = f * 174611.0 / 55.0
```

```
                slv = 6.812921 * f ** (1.0 / 18.0)
                if (slv .lt. 13.06) then
                        slv = 13.06
                end if
        else
C               /* 1 <= k <= 3 */
                slv = 13.06
        end if

        pk = 1.0
        do 10 i = 1, k
10              pk = pk * dble (i)
C       /* pk = k! */

        if (x .eq. 0.0) then
C               /* SING error */
C               /* return INF */
                polygamma = INF
                return
        else if (x .ge. slv) then
                s = 0.0
                if (x .gt. SQRINF) then
                        polygamma = NaN
                        return
                end if
                x2 = x * x
                if (mod (k, 2) .eq. 0) then
                        isgn = 1
                else
                        isgn = -1
                end if

                if (k .eq. 0) then
C                       /* digamma */
                        do 20 i = 1, IBRNTRM
                                ii = IBRNTRM - i + 1
                                i2 = 2 * ii
                                s = s + brn(ii) / dble (i2 * isgn)
                                s = s / x2
                                isgn = -isgn
20                      continue
                        s =  s + dlog(x) - 1.0 / 2.0 / x
                else
C                       /* k >= 1; trigamma, tetragamma,... */
                        do 30 i = 1, IBRNTRM
                                ii = IBRNTRM - i + 1
                                i2 = 2 * ii
                                c = 1.0
                                do 35 j = i2 + 1, i2 + k - 1
35                                      c = c * dble (j)
                                s = s + brn(ii) * c * dble (isgn)
                                s = s / x2
                                isgn = -isgn
30                      continue
                        do 40 i = 1, k
40                              s = s / x

                        if (x .gt. (INF ** (1.0 / dble(k)))) then
C                               /* SING error */
C                               /* ‘pxk’ is INF */
                                polygamma = NaN
```

```
                                return
                        end if
                        pxk = 1.0
                        do 50 i = 1, k
50                              pxk = pxk * x
C                       /* pxk = pow(x, k) */

                        s = s - pk / 2.0 / pxk / x * dble (isgn)
                        c = pk / dble (k)
                        s = s - c / pxk * dble (isgn)
                end if
        else
C               /*
C                * x < slv;
C                * Adopted ‘y’ instead of ‘x’ to the asymptotic exspansion,
C                * we caluculate the value.
C                */
                n = int (slv -x)
                y = dble (n) + x + 1.0
                if ((x .lt. 0.0) .and. (y .eq. dble(int (y)))) then
C                       /* x < 0.0 && ’y’ is integer */
C                       /* SING error */
C                       /* return INF */
                        polygamma = INF
                        return
                end if
                s = polygamma(k, y)
C
                if (mod (k, 2) .eq. 0) then
                        isgn = -1
                else
                        isgn = 1
                end if
                do 70 i = 1, (n + 1)
                        y = y - 1.0
                        if (abs(y) .lt. 1.0e-3) then
                                if (x .gt. 0.0) then
                                        y = x - dble(int (x + 0.5))
                                else
                                        y = x - dble(int (x - 0.5))
                                end if
                        end if
                        pxk = 1.0
                        do 65 j = 1, k
65                              pxk = pxk * y
C                       /* pxk = pow(y, k) */
                        if ((pxk * y) .eq. 0.0) then
C                               /* SING error */
C                               /* return INF */
                                polygamma = dble (isgn) * INF
                                return
                        end if
                        s = s + dble (isgn) *  pk / pxk / y
70              continue
        end if
        polygamma = s
        return
        end
```

参 考 文 献

[] Amos, D. E.(1983): “Algorithm 610, A Portable FORTRAN Subroutine for Derivatives of the Psi Functions,” *ACM Trans. on Mathematical Software*, **9**, [4], 494-502.

[] Bowman, K.O. & Shenton, L.R. (1988) : *Properties of Estimators for the Gamma Distribution,* Marcel Dekker, New York.

[] Char, B. W. *et al.* (1992): *First Leaves: A Tutorial Introduction to Maple V*, Springer-Verlag, New York.

[] Didonato, A. R. and Morris, Jr., A. H. (1986): “Computation of the Incomplete Gamma Function Ratios and their Inverse,” *ACM Trans. on Mathematical Software*, **12**, [4], 377-393.

[] 森口 繁一 , 他 (1960): 「数学公式 III (全 3 冊)」, 岩波書店 .

[] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T.(1988): *Numerical Recipes in C*, Cambridge University Press.

[] Sibuya, M.(1979): “Generalized hypergeometric digamma and trigamma distributions,” *Ann, Inst. Stat. Math.*, **31**[3]A, 373-390.

[] Sibuya, M.(1980): “Multivariate digamma and distributions,” *Ann, Inst. Stat. Math.*, **32**[1]A, 25-36.

[] 山内 二郎 (編) (1972): 「統計数値表」, 日本規格協会 .

著者連絡先 :〒113 東京都文京区本郷 1-33-13 日本生命春日町ビル
株式会社リコー ソフトウェア事業部 ソフトウェア研究所
TEL 03-5689-6421 E-mail tunenori@src.ricoh.co.jp