

**stichting
mathematisch
centrum**



AFDELING TOEGEPASTE WISKUNDE
(DEPARTMENT OF APPLIED MATHEMATICS)

TW 202/80 JUNI

N.M. TEMME

THE NUMERICAL COMPUTATION OF THE CONFLUENT
HYPERGEOMETRIC FUNCTION $U(a, b, z)$

Preprint

Kruislaan 413, 1098 SJ Amsterdam,

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

The numerical computation of the confluent hypergeometric function $U(a,b,z)$ *)

by

N.M. Temme

ABSTRACT

An algorithm is given for the computation of the confluent hypergeometric function $U(a,b,z)$. For real values of a , b and z , $z > 0$, ALGOL 60 procedures are given. The computations are based on a Miller algorithm and on asymptotic expansions.

KEY WORDS & PHRASES: *confluent hypergeometric function, Kummer function, computation of special functions, Miller algorithm, ALGOL 60 programs*

*) This report will be submitted for publication elsewhere.

1. INTRODUCTION

1.1. Definitions and relevant properties

We consider the computation of the confluent hypergeometric function

$$(1.1) \quad U(a,b,z) = \frac{1}{\Gamma(a)} \int_0^{\infty} e^{-zt} t^{a-1} (1+t)^{b-a-1} dt.$$

This representation is valid for $\operatorname{Re} a > 0$, $\operatorname{Re} z > 0$, $b \in \mathbb{C}$. For other values of a and z we define $U(a,b,z)$ by analytic continuation. In general, i.e., for general a and b values, $U(a,b,z)$ is singular at $z = 0$. It is a many-valued function with respect to z and we consider for $|\arg z| < \pi$ the principal branch, which is real (if a and b are real) for $z > 0$. With respect to a (or b) $U(a,b,z)$ is an entire function. For $a = 0, -1, -2, \dots$ it can be written in terms of Laguerre polynomials

$$(1.2) \quad U(-n, \alpha+1, z) = (-1)^n n! L_n^{(\alpha)}(z).$$

If $b-a-1 = n$ ($n = 0, 1, 2, \dots$) it also reduces to an elementary function. It easily follows from (1.1) that

$$(1.3) \quad U(a, a+n+1, z) = \sum_{k=0}^n (a)_k \binom{n}{k} z^{-a-k},$$

where $(a)_k = \Gamma(a+k)/\Gamma(a)$, $k = 0, 1, 2, \dots$.

The function $U(a,b,z)$ is a solution of Kummer's equation

$$(1.4) \quad zw'' + (b-z)w' - aw = 0.$$

A second solution is the function (also denoted by ${}_1F_1(a,b,z)$)

$$(1.5) \quad M(a,b,z) = \sum_{n=0}^{\infty} (a)_n / (b)_n z^n / n!$$

which is regular for all finite z -values, although it is undefined for some pairs (a,b) .

The functions $U(a,b,z)$ and $M(a,b,z)$ satisfy recurrence relations with

respect to a and b . With respect to a we have

$$(1.6) \quad f_{a-1} + (b-2a-z)f_a + a(1+a-b)f_{a+1} = 0$$

which is satisfied by

$$(1.7) \quad U(a,b,z) \quad \text{and} \quad M(a,b,z)/\Gamma(1+a-b).$$

With respect to b we have

$$(1.8) \quad (b-a-1)f_{b-1} + (1-b-z)f_b + zf_{b+1} = 0.$$

with solutions

$$(1.9) \quad U(a,b,z) \quad \text{and} \quad \Gamma(b-a)/\Gamma(b)M(a,b,z).$$

We use here the notation of ABRAMOWITZ & STEGUN (1964, Chapter 13). More information on Kummer's function can be found in this and in many other references, for instance in SLATER (1960).

For certain combinations of a and b the function $U(a,b,z)$ reduces to other special functions. We list here some examples (see ABRAMOWITZ & STEGUN (1964, p.510) for a more extensive table).

a	b	z	relation	function
$\nu + \frac{1}{2}$	$2\nu + 1$	$2z$	$\pi^{-\frac{1}{2}} e^z (2z)^{-\nu} K_\nu(z)$	modified Bessel
$-n$	$\alpha + 1$	z	$(-1)^n n! L_n^\alpha(z)$	Laguerre polynomial
$1 - \alpha$	$1 - \alpha$	z	$e^z \Gamma(\alpha, z)$	incomplete gamma
1	$1 + \alpha$	z	$e^z z^{-\alpha} \Gamma(\alpha, z)$	
$-\frac{1}{2}\nu$	$\frac{1}{2}$	$\frac{1}{2}z^2$	$2^{-\frac{1}{2}\nu} e^{\frac{1}{2}z^2} D_\nu(z)$	parabolic cylinder

The incomplete gamma function has as special cases the exponential integrals, the sine and cosine integrals and the error functions. $D_n(z)$ is an Hermite polynomial, $D_{-n-1}(z)$ is a repeated integral of the error function

($n = 0, 1, 2, \dots$).

1.2. Contents of the paper

The algorithms given here can be used for complex values of a, b and z . In order to formulate concrete stability conditions it is better to concentrate on real a, b and $z = x > 0$.

The algorithms are implemented in two ALGOL 60 procedures:

1. the procedure *chu* computes the values

$$(1.10) \quad u_k = (a)_k U(a+k, b, x) \quad \text{and} \quad (a)_K U'(a+K, b, x)$$

for $a \geq 0$, $k = 0, 1, \dots, K$, $b \in \mathbb{R}$, $x > 0$ and K is an integer ≥ 0 .

2. the procedure *uabx* computes the values

$$(1.11) \quad U(a, b, x) \quad \text{and} \quad U'(a, b, x)$$

for $a \in \mathbb{R}$, $b \in \mathbb{R}$, $x > 0$.

The prime denotes differentiation with respect to x . The single value $(a)_K U'(a+K, b, x)$, delivered by *chu* is used for a backward recursion process and it plays an important role when *chu* calls recursively for itself. Also *uabx* calls for *chu*, and then the derivative is important too. It is possible to obtain the derivative $U'(a+K, b, x)$ from the values u_K, u_{K-1} , for instance by using

$$xU'(a, b, x) = (a-b+x)U(a, b, x) - U(a-1, b, x).$$

For small values of x and/or large values of b this formula is not stable. Our methods in *chu* guarantee a stable computation for the derivative of $U(a+K, b, x)$.

In Section 2 we discuss recursion with respect to a . For values of x bounded away from zero we can use a Miller algorithm. For x -values close to zero, we use asymptotic expansions. In an earlier publication TEMME (1975a) we used analogous methods for the computation of (1.10) with $a = b = 1$.

In Section 3 we consider some aspects of recursion with respect to b . The ALGOL 60 procedures are described in Section 4. The procedures call for procedures published earlier in TEMME (1975b), a publication on the computation of modified Bessel functions.

2. RECURSION WITH RESPECT TO a

From (1.6) and (1.7) it follows that u_k of (1.10) satisfies

$$(2.1) \quad (a+k-1)u_{k-1} + (b-2a-x-2k)u_k + (a+k+1-b)u_{k+1} = 0$$

with as a second solution

$$(2.2) \quad g_k = \Gamma(a+k)/\Gamma(1+a+k-b) M(a+k, b, x).$$

From asymptotic expansions of the gamma functions and from SLATER (1960, p.80) it follows easily that for $k \rightarrow \infty$

$$(2.3) \quad u_k = 2/\Gamma(a) (k/x)^{(b-1)/2} e^{x/2} K_{b-1} [2(kx)^{1/2}] [1+O(k^{-1/2})],$$

$$g_k = \Gamma(b) (k/x)^{(b-1)/2} e^{x/2} I_{b-1} [2(kx)^{1/2}] [1+O(k^{-1/2})],$$

where $I_\nu(z)$ and $K_\nu(z)$ are modified Bessel functions. For $\text{Re } kx \rightarrow \infty$ we obtain by using well known expansions for the Bessel function

$$(2.4) \quad u_k \sim \pi^{1/2}/\Gamma(a) (k/x)^{(b-1)/2} (kx)^{-1/4} \exp[\frac{1}{2}x - 2(kx)^{1/2}],$$

$$g_k \sim \frac{1}{2}\pi^{-1/2} \Gamma(b) (k/x)^{(b-1)/2} (kx)^{-1/4} \exp[\frac{1}{2}x + 2(kx)^{1/2}].$$

So, in the terminology of GAUTSCHI (1967), u_k is a minimal solution and g_k is a dominant solution of (2.1). Hence the computation of u_{k+1} from u_k , u_{k-1} (using (2.1)) is not stable. If we want to use (2.1) for the computation of $\{u_k\}$ we have to use it backwardly, i.e. from u_k , u_{k+1} we compute u_{k-1} . Backward recursion may be unstable for small k ; see §3.2.

2.1. A Miller algorithm for $\{u_k\}$

As follows from (2.4) we can use a Miller algorithm for the computation of $\{u_k\}$. For details of such an algorithm we refer to GAUTSCHI (1967). As normalization we use

$$(2.5) \quad \sum_{k=0}^{\infty} m_k u_k = x^{-a}, \quad m_k = (-1)^k \binom{b-a-1}{k}.$$

This relation is easily verified by substituting (1.10) in (2.5) and using (1.1). For $x > 0$, $a \geq 0$, $b \in \mathbb{R}$ the u_k are non-negative; for $b \leq a+1$ all m_k are non-negative. Hence in these cases (it will appear that we can restrict b to $[0,1]$) the series in (2.5) has non-negative terms, which is important in the numerical procedure. For $a = 0$ we have $u_0 = 1$, $u_1 = u_2 = \dots = u_k = 0$. For the Miller algorithm we suppose $0 < a \leq 1$, other positive a -values need not to be considered.

This algorithm was also used in TEMME (1975b) for the computation of the Bessel function $K_\nu(x)$. For x -values satisfying $1 < x \leq 4$, CAMPBELL (1980) modified this algorithm by using the Wronskian relation for the modified Bessel functions. His faster method applies also in the present case. For $1.4 < x \leq 6.5$ we use (instead of (2.5)) the normalization

$$(2.6) \quad U(a,b,x)M'(a,b,x) - U'(a,b,x)M(a,b,x) = \frac{\Gamma(b)}{\Gamma(a)} x^{-b} e^x.$$

The M -functions are computed by using (1.5); the rate of convergence of (1.5) is the same as that of the exponential function ($a = b$).

In (2.6) U , M and M' are positive and U' is negative for $a > 0$, $b \geq 0$, $x > 0$. Therefore, (2.6) is stable for the indicated ranges of a , b and x . The Wronskian (2.6) is also used in the case $x > 6.5$ and $a = b$. In that case the M -functions are exponentials and (2.6) becomes

$$U(a,a,x) - U'(a,a,x) = x^{-a}.$$

For small a and/or b the gamma functions in (2.6) are not easy to handle in calculations. In the computer program we combine these functions with M' and U' in such a way that underflow or overflow will never occur when

a or b is small.

In the algorithm a different version of (2.1) is used. From (2.1) it is not clear whether backward recursion is stable for small values of k. By introducing

$$(2.7) \quad v_k = (a)_k U'(a+k, b, x)$$

(compare (1.10)) we obtain the first order (2x2)-system

$$(2.8) \quad \begin{cases} v_k = v_{k+1} - u_{k+1} \\ u_k = [-xv_{k+1} + (a+k+1+x-b)u_{k+1}]/(a+k) \end{cases} \quad k \geq 0.$$

If $x > 0$, $a > 0$, $b \in \mathbb{R}$, then v_k is negative. Hence, backward recursion is stable for all $k \geq 0$ if $a+1+x-b \geq 0$. As mentioned earlier, we take $a \in (0,1]$, $b \in [0,1]$ and $x > 0$. Hence (2.8) is stable for these limited ranges of a and b. Other values of b are treated in Section 3.

The starting value v in the Miller algorithm is computed as in TEMME (1975b). In fact it is the method of OLVER (1967). We take as starting values

$$\bar{u}_v = 1, \quad \bar{v}_v = -2v/[x+(x^2+4xv)^{1/2}].$$

This choice follows from approximating the differential equation for $y = U'(a,b,x)/U(a,b,x)$, viz. $x(y'+y^2) + (b-x)y - a = 0$, by putting $y' = b = 0$.

2.2. Backward recursion with computed starting values

For small values of x, the Miller algorithm is not very efficient to generate u_k of (1.10). As follows from (2.4) the dominance of g_k over u_k becomes rather weak if x is small. For small x we use computed starting values for the recursion (2.8). These values are obtained from asymptotic expansions of $U(a,b,x)$ and $U'(a,b,x)$ for large a, which are valid for small x. Such expansions are given in SLATER (1960, p.80). Similar expansions are derived in TEMME (1979), where for the case of real a, b and $x > 0$ simple bounds are given for the remainders in the expansions. The

expansions are for $N = 1, 2, 3, \dots$

$$\begin{aligned}
 U(a, b, x) &= \sum_{n=0}^{N-1} c_n(b, x) \phi_n(a, b, x) + R_N(a, b, x) \\
 U'(a, b, x) &= \sum_{n=0}^{N-1} d_n(b, x) \phi_{n-1}(a, b, x) + T_N(a, b, x) \\
 (2.9) \quad \phi_n(a, b, x) &= 2e^{x/2} / \Gamma(a) (x/a)^{(n+1-b)/2} K_{n+1-b} [2(ax)^{1/2}] \\
 c_n(b, x) &= \sum_{j=0}^n b_j^{(1)}(x) b_{n-j}^{(2)}(b), \\
 d_n(b, x) &= \sum_{j=0}^n b_j^{(1)}(x) b_{n-j}^{(3)}(b).
 \end{aligned}$$

The $b_j^{(i)}$ are defined by the generating functions (for $|\tau| < \pi$)

$$\begin{aligned}
 (2.10) \quad \exp[x\mu(\tau)] &= \sum_{j=0}^{\infty} b_j^{(1)}(x) \tau^j; \quad [\tau/(1-e^{-\tau})]^b = \sum_{j=0}^{\infty} b_j^{(2)}(b) \tau^j; \\
 b_j^{(3)}(b) &= (1-j/b) b_j^{(2)}(b); \quad \mu(\tau) = 1/\tau - 1/(e^\tau - 1) - \frac{1}{2}.
 \end{aligned}$$

The function K_ν in ϕ_n is the modified Bessel function. Coefficients $b_j^{(1)}(x)$ and $b_j^{(2)}(b)$ are for $j = 0, \dots, 8$ incorporated in the ALGOL 60 program (in fact $b_j^{(2)}(b)/b$); see arrays bx , $bb[0:8]$ in the block announced by "if $x \leq 1.4$ then").

The remainders R_N and T_N are bounded by simple expressions. Let

$$(2.11) \quad K(d, b, x) = |\sin d|^{-b} \exp[\frac{1}{2}x(1/d+1/|\sin d|)]$$

then, as proved in TEMME (1979), for $a > 0$, $x > 0$, $b \geq 0$

$$\begin{aligned}
 (2.12) \quad |R_N(a, b, x)| &< d^{b-N} K(d, b, x) \phi_N(a, b, x) \\
 |T_N(a, b, x)| &< d^{b-N+1} K(d, b+1, x) \phi_N(a, b+1, x),
 \end{aligned}$$

where d is arbitrary in $[3\pi/2, 2\pi)$.

We use the above expansions for the computation of u_k and v_k' for a large value of k (see (1.10) and (2.7)). We fix N ; in the computer program we use $N = 9$. Furthermore we take $0 \leq b \leq 1$. We have to choose $k \geq K$ so large that, given $a \in (0,1]$, $b \in [0,1]$, $x > 0$, $\varepsilon > 0$, $d \in [3\pi/2, 2\pi)$,

$$(2.13) \quad (a)_k |R_N(a+k, b, x)| < \varepsilon u_k \quad \text{and} \quad (a)_k |T_N(a+k, b, x)| < \varepsilon |v_k'|.$$

The minimal value of k satisfying both inequalities of (2.13) is computed by using inequalities for $\phi_j(a, b, x)$ as given in TEMME (1979). Also the value of d is computed according to a device given there.

When u_k and v_k are computed we use (2.8) for backward recursion.

2.3. Negative a-values

The integral (1.1) defines $U(a, b, z)$ for $\text{Re } a > 0$. However, as follows from recursion with respect to a , see (1.6), $U(a, b, x)$ is an entire function of a and recursion can be used for the analytic continuation of (1.1) to $\text{Re } a \leq 0$. First values for the recursion can be obtained from the algorithms of Section 2. We give an ALGOL 60 program *uabx* which computes $U(a, b, x)$ for $a \in \mathbb{R}$, $b \in \mathbb{R}$, $x > 0$, and the x -derivative. Computation of (1.10) for $a < 0$ is not attractive due to possible singularities (for negative integer values of a) of the factor $\Gamma(a+k)/\Gamma(a)$. Special values for $a = -n$, $n = 0, 1, 2, \dots$, follow from (1.2).

For $a \rightarrow -\infty$, b bounded, $x > 0$ we have

$$U(a, b, x) = \Gamma(\frac{1}{2}b - a + \frac{1}{4}) \pi^{-\frac{1}{2}} e^{\frac{1}{2}bx} x^{\frac{1}{4} - \frac{1}{2}b} \cos(\chi + a\pi) [1 + O(-a)^{-\frac{1}{2}}]$$

$$M(a, b, x) = \Gamma(b) \pi^{-\frac{1}{2}} e^{\frac{1}{2}bx} (-ax)^{\frac{1}{4} - \frac{1}{2}b} \cos(\chi) [1 + O(-a)^{-\frac{1}{2}}]$$

with $\chi = (2bx - 4abx)^{\frac{1}{2}} - \frac{1}{2}b\pi + \frac{1}{4}\pi$. It follows that $U(a, b, x)$ and $M(a, b, x)/\Gamma(1+a-b)$, the two solutions of (1.6), are not dominant with respect to each other. Thence recursion is possible for both solutions in negative a -direction.

We use (2.8) in the form

$$\begin{cases} U'(a,b,x) = U'(a+1,b,x) - U(a+1,b,x) \\ U(a,b,x) = -U'(a+1,b,x) + (a+x+1-b)U(a+1,b,x). \end{cases}$$

It is not possible to give full-proof conditions for the stability as was done for $a > 0$. In fact $U(a,b,x)$ and $U(a+1,b,x)$ may have different signs when $a < 0$. The same remark applies for the derivatives.

When $a < 0$ and $a = b$ we have the incomplete gamma case, viz.

$$U(-a,-a,x) = e^x \Gamma(1+a,x).$$

In that case we can claim stability since we can recur according to the relation

$$e^x \Gamma(a+1,x) = a e^x \Gamma(a,x) + x^a,$$

which is stable for increasing (positive) a . In terms of the U-functions the recursion is

$$U(-a,-a,x) = aU(1-a,1-a,x) + xU(1-a,2-a,x),$$

in which the last term is an elementary function (see (1.3)).

3. RECURSION WITH RESPECT TO b

For convergence aspects of the algorithms of the previous section we restricted b to the interval $[0,1]$. Here we are concerned with the remaining b -values and we will start with $b < 0$. Throughout this section we suppose that $a > 0$ and $x > 0$.

3.1. Negative b -values

The crucial relation is the reflexion formula

$$(3.1) \quad U(a,b,x) = x^{1-b} U(1+a-b,2-b,x).$$

If b is negative the b -place in the U -function at the right is positive (in fact the reflexion occurs at $b = 1$, for convenience we use it at $b = 0$). In order to compute

$$(3.2) \quad \begin{aligned} u_k &= (a)_k U(a+k, b, x), & k = 0, \dots, K, \\ v_K &= (a)_K U'(a+K, b, x) \end{aligned}$$

for $b < 0$ we first compute

$$(3.3) \quad \begin{aligned} \bar{u}_k &= (c)_k U(c+k, 1-b, x), & k = 0, \dots, K, & \quad c = 1+a-b \\ w &= (c)_K U'(c+K, 1-b, x). \end{aligned}$$

Using (3.1) and $U'(a, b, x) = -aU(a+1, b+1, x)$, we write this as

$$(3.4) \quad \begin{aligned} \bar{u}_k &= (c)_k x^b U(a+k+1, b+1, x) = -x^b \frac{(c)_k}{(a)_{k+1}} v_k \\ w &= -(c)_{K+1} U(c+K+1, 2-b, x) = -x^{b-1} \frac{(c)_{K+1}}{(a)_{K+1}} u_{K+1} \end{aligned}$$

where $v_k = u'_k$ (see (2.7)).

For computing the values in (3.3) we need an algorithm for computing $\{u_k\}$ for $b > 1$. Details on this point will be discussed in the next subsection.

The values in (3.4) are used to compute the requested u_k . First u_K is computed by using

$$(3.5) \quad aU(a, b, x) = a(1+a-b)U(a+1, b, x) - xU'(a, b, x)$$

and the remaining u_k ($k = K-1, \dots, 0$) follow from the second of (2.8). This recursion is stable since v_k and b are negative. Also (3.5) is stable.

REMARK. Instead of (3.3) we might have computed

$$\bar{u}_k = (c)_k U(c+k, 2-b, x), \quad k = 0, \dots, K, \quad c = 1+a-b,$$

$$w = (c)_K U'(c+K, 2-b, x) = - (c)_{K+1} x^{b-2} U(a+K, b-1, x).$$

Then

$$\bar{u}_k = \frac{(c)_k}{(a)_k} x^{b-1} u_k;$$

hence u_k follows rather straightforward from the computed \bar{u}_k . However, we need also v_k . This value can be computed from the u_k , u_{k-1} , or w , u_k by using one of the recursions

$$xU'(a, b, x) = (a-b+x)U(a, b, x) - U(a-1, b, x),$$

$$xU'(a, b, x) = (1-b)U(a, b, x) - (1+a-b)U(a, b, -1, x).$$

For small values of x and/or large values of $-b$ they are not stable. In the approach described earlier all computations are stable.

3.2. The case $b > 1$

Recursion with respect to b can be done by using

$$(3.6) \quad (b-a-1)U(a, b-1, x) + (1-b-x)U(a, b, x) + xU(a, b+1, x) = 0,$$

of which a second solution is $\Gamma(b-a)/\Gamma(b)M(a, b, x)$. From the series (1.5) it follows that $M(a, b, x) = 1 + O(b^{-1})$ for $b \rightarrow \infty$, a and x bounded. From (1.1) it follows that for $b-a-1 \geq 0$ (by using $(1+t)^{b-a-1} \geq 1$)

$$(3.7) \quad U(a, b, x) > x^{1-b} \Gamma(b-1)/\Gamma(a).$$

Hence for the recursion (3.6) the function $U(a, b, x)$ is dominant with respect to the second solution. It follows that recursion in the forward b -direction is stable.

More insight is gained when we use the derivative as in (2.8). Let us write

$$(3.8) \quad \begin{aligned} f_k &= U(a, b+k, x) \\ g_k &= U'(a, b+k, x). \end{aligned}$$

Then the recursion in the b-direction is given by

$$(3.9) \quad \begin{cases} f_{k+1} = f_k - g_k \\ xg_{k+1} = (k+b)g_k - af_k. \end{cases}$$

Since $f_K > 0$, $g_k < 0$ ($k \geq 0$), (3.9) is a recursion without subtractions (formula (3.6) lacks this property), and hence it is stable.

Suppose we want to compute (1.10) with $b > 1$. Then we define $b_1 = b - [b]$ and we compute the values

$$({}_a)_K U(a+K, b_1, x), \quad ({}_a)_K U'(a+K, b_1, x)$$

by using the algorithms of Section 2. Then we use (3.9) for obtaining u_k and v_k and then (2.8) can be used for the remaining u_k ($k = K-1, \dots, 0$).

As mentioned earlier, (2.8) is stable if $a+k+1+x-b$ is not negative. Large b values may violate this condition and in fact the second of (2.8) is not stable for large b -values. To show this we need the asymptotic relation (compare (3.7))

$$(3.10) \quad U(a, b, x) \sim x^{1-b} \Gamma(b-1) / \Gamma(a),$$

which is valid for $x \rightarrow 0$ ($b > 1$) or $b \rightarrow \infty$ (x fixed). Inserting this in the second of (2.8) (with modifications for v_{k+1} and u_{k+1}) we infer that indeed large values will cancel each other in order to obtain a smaller value, especially when k is small. Repeated application of (2.8) is allowed as long as k is so large that $a+k+1+x-b$ is not negative. If a, b and x (and K) are such that for some k this quantity becomes negative, instabilities may arise.

It is not easy to formulate an "if and only if" condition for the stability. The subtraction $p-q$ of two positive numbers ($q < p$) is harmless if, say, $q < \frac{1}{2}p$. The recursion (2.8) can be done by checking this criterion. However, in the computer program it is convenient to have a priori information on safe k -values. We use simply the above condition: a k -value is safe if $a+k+1+x-b$ is non-negative. Call m the largest value of k that makes $a+k+1+x-b$ negative. Then

$$(3.11) \quad \{u_{m+1}, u_{m+2}, \dots, u_K\}$$

are computed via (2.8) and for

$$(3.12) \quad \{u_0, \dots, u_m\}$$

we need a different approach.

The value m is defined by

$$(3.13) \quad m = [b-x-1-a];$$

if $b-x-1-a = [b-x-1-a]$ then $m := b-x-2-a$. It may happen that $m < 0$. In that case all u_k are obtained by (2.8). If $m \geq K$ none of the u_k are obtained by (2.8). In other words, the sets in (3.11) or (3.12) may be empty.

To obtain the values in (3.12) we may suppose that the values

$$(3.14) \quad (a)_j U(a+j, b_1, x), \quad j = 0, \dots, m, \quad b_1 = b - [b],$$

are available, together with their derivatives. We can use (3.9) for recursion up to b with

$$(3.15) \quad f_k = (a)_j U(a+j, b_1+k, x), \quad g_k = (a)_j U'(a+j, b_1+k, x).$$

All these recursions are stable. However, it may be rather expensive, since every element in (3.14) is recurred to the b -level. For an alternative method, which may be much more efficient, we proceed as follows. We start with the element with $j = 0$ in (3.14) and we recur up to $b-m$. That is, we compute (3.15) with $j = 0$ for $k = 0, 1, \dots, [b]-m$, giving

$$f_{[b]-m} = U(a, b-m, x), \quad g_{[b]-m} = U'(a, b-m, x).$$

Next we compute

$$(3.16) \quad F_j = (a)_j U(a+j, b-m+j, x), \quad G_j = (a)_j U'(a+j, b-m+j, x)$$

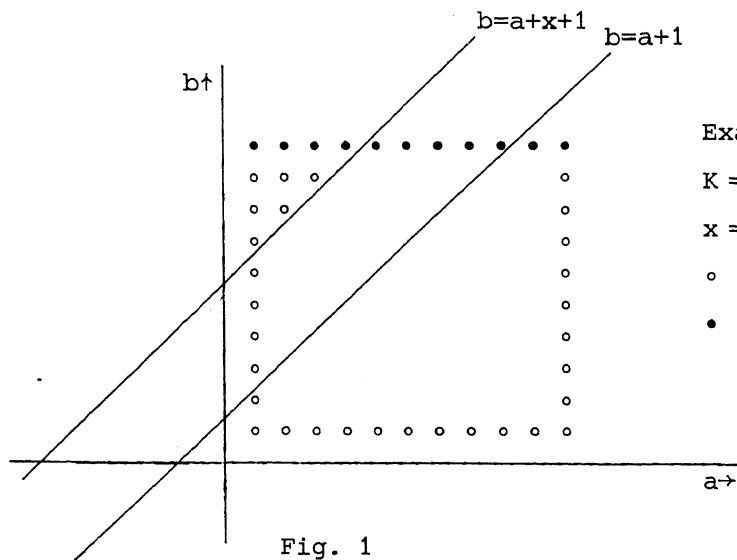
for $j = 1, 2, \dots, m$. (Consider the (a, b) -plane. We compute U and U' along a diagonal in the (a, b) -plane.) These diagonal elements F_j, G_j are obtained by using

$$(3.17) \quad \begin{cases} aU(a+1, b+1, x) = -U'(a, b, x) \\ axU'(a+1, b+1, x) = -aU(a, b, x) + (b-x)U'(a, b, x), \end{cases}$$

which in fact is Kummer's equation (1.4). In terms of F_j, G_j it reads as

$$(3.18) \quad \begin{cases} F_{j+1} = -G_j \\ xG_{j+1} = -(a+j)F_j + (b+j-m-x)G_j \end{cases}$$

for $j = 0, 1, \dots, m-1$. This recursion is stable: G_j is negative and $(b+j-m-x)$ is positive for $j = 0, \dots, m-1$; (cf. (3.13)). From the diagonal a final recursion in the b -direction is performed by using (3.9) with starting values F_j, G_j in order to obtain (3.12). The procedure is illustrated in Fig. 1.



Example:

$$K = 10, \quad a = 0.5, \quad b = 9.75$$

$$x = 4, \quad m = 3, \quad b_1 = 0.75$$

○ auxiliary values

• requested values

The values • at the right of the line $b = a + x + 1$ are computed via (2.8).

In the procedure *chu* this algorithm is controlled by the Boolean variables *r* and *s*; *r* = true means that (3.11) is empty, all requested values follow from diagonal elements; *s* = false means that (3.12) is empty, all requested values follow from backward *a*-recursion. In ALGOL 60 notation : $r := K \leq m$, $s := m \geq 0$ (in *chu* *K* is replaced by *kmax*).

3.3. The polynomial case

The actual algorithm for the case $b > 1$ is more intricate than described above. The point is that we take advantage of the possibility that some or all of the wanted values $\{u_k\}$ are elementary functions. See (1.3). In that case the time-wasting algorithms of Section 2 may be circumvented.

Let us introduce

$$(3.19) \quad c = b - a - 1$$

and we suppose here that *c* is a non-negative integer. Then the values

$$(3.20) \quad \{u_0, u_1, \dots, u_c\}$$

are polynomials, and the values

$$(3.21) \quad \{u_{c+1}, \dots, u_K\}$$

are higher transcendentals. The set in (3.21) may be empty. If it is not, its elements are computed as in the previous subsection. Since *m* (see 3.13) is not larger than *c*, the set (3.21) is a subset of (3.11). Hence, the higher transcendentals (3.21) can be computed by backward *a*-recursion, whereas some of the set (3.20) (i.e., u_0, \dots, u_m) are to be computed from diagonal elements. These diagonal elements are elementary functions as well. If (3.21) is empty than all $\{u_k\}$ are polynomials. Then the first $\{u_0, \dots, u_m\}$ are computed from elementary diagonal elements, the remaining with backward *a*-recursion with elementary starting values.

The polynomial case is recognized in *chu* by the Boolean variables *p* and *q*: *p* = true iff *c* is a non-negative integer, *p* = true and *q* = true iff

all $\{u_k\}$ are polynomials. That is, if $K \leq c$. In ALGOL 60 notation:
 $p := c = \text{entier}(c) \wedge c \geq 0, q := K \leq c$.

If we combine the possible cases of §3.2 with those of the present subsection we obtain 8 different situations A,B,...,H. An illustration by means of the (a,b)-plane is again very instructive. In Fig. 2 the positions of the (a,b) parameters of the wanted elements $\{u_k\}$ are depicted with respect to the lines $b = a+1, b = a+x+1$.

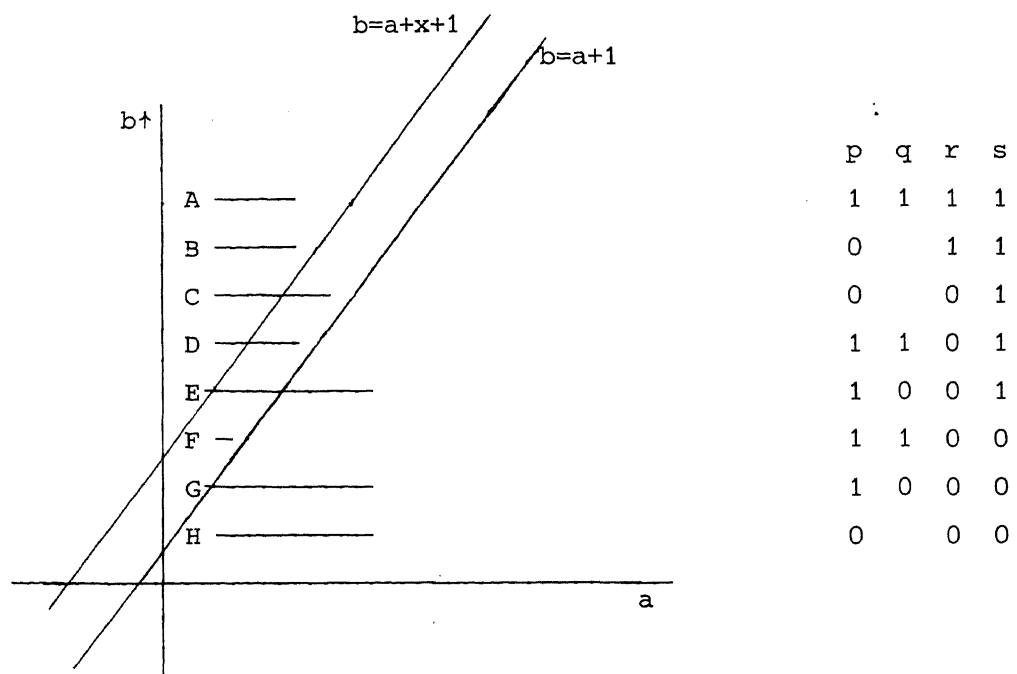


Fig. 2

For convenience, $p = \text{true}$, $p = \text{false}$, etc. are replaced by $p = 1$, $p = 0$, respectively. In the cases A,B,C,D,E diagonal elements must be used ($s = 1$). In A,B all elements follow from diagonal elements ($r = 1$). In C,D,E,F,G,H ($r = 0$) backward a-recursion is used for some (in F,G,H for all) elements. Values of q if $p = 0$ are not significant; r and s are not independent: $s = 0$ implies $r = 0$. G and H are treated as being the same: in G higher transcendental functions are used in the backward a-recursion and the process is not terminated at the moment that elementary functions turn up. If $K = 0$ some cases are equivalent. Then the location with respect to the line $b = a+x+1$ is not relevant; the only question is whether it is a polynomial case or not. For

convenience we put $r = s = 1$ (case A or B) if $K = 0$.

Examples with numerical values for the different eight cases are given in Table 1.

	a	b	x	K	c	m
A	2.5	8.5	1.4	2	5	3
B	2.0	8.5	1.4	2	5.5	4
C	2.0	8.5	1.4	5	5.5	4
D	2.5	8.5	1.4	4	5	3
E	2.5	8.5	1.4	6	5	3
F	2.5	8.5	6.5	2	5	-2
G	2.5	8.5	6.5	6	5	-2
H	2.0	8.5	6.5	2	5.5	-2

Table 1

With the shown values of a, b, x, K all cases A through H are covered.

4. ALGOL 60 PROCEDURES

The procedures given here make use of external ALGOL 60 procedures for the computation of the gamma function for positive argument and of the Bessel function $K_\nu(x)$, for $x > 0$ and $\nu \in [0, 2)$. For the Bessel function we call the procedure given in TEMME (1975b).

4.1. The procedure *chu*

The heading of the procedure reads as follow:

```
procedure chu(a, b, x, kmax, eps, u, uprime); value a, b, x, kmax, eps;  

            real a, b, x, eps, uprime; integer kmax; array u;
```

The meaning of the formal parameters is:

a, b, x : <arithmetic expressions>;
 the parameters of the confluent hypergeometric function $U(a, b, x)$;
 $a \geq 0, b \in \mathbb{R}, x > 0$.

$kmax$: <arithmetic expression>;
 the upper bound of the array u , $kmax \geq 0$.

eps : <arithmetic expression>;
 the desired relative accuracy; $eps > 0$.

u : <array identifier>
array $u[0: kmax]$;
 exit: the values of $\Gamma(a+k)/\Gamma(a)U(a+k, b, x)$, $0 \leq k \leq kmax$, are
 assigned to $u[k]$;

$uprime$: <identifier>;
 exit: the value of $\Gamma(a+kmax)/\Gamma(a)U'(a+kmax, b, x)$ is assigned to
 $uprime$.

The procedure *chu* calls for the nonlocal procedures *besska* and *gamma*; the first is published in TEMME (1975b) (*besska* also calls for other procedures). The procedure is not protected against underflow or overflow and does not give a message when the parameters are out of range. This last aspect is easily incorporated by the user, the first aspect strongly depends on the computing machinery. In general the functions $U(a+k, b, x)$ are singular at $x = 0$:

if $b < 1$, the functions are bounded at $x = 0$;
 if $b = 1$, $U(a, b, x) = O(\ln x)$ as $x \rightarrow 0$;
 if $b > 1$, $U(a, b, x) = O(x^{1-b})$ as $x \rightarrow 0$.

Hence for large b -values this singularity may cause overflow. For negative b -values the reflexion formula (3.1) is used. As a consequence, meanwhile an overflow situation may occur. For x -values bounded away from zero together with large b -values see (3.3), which shows that large function values arise in that case too.

If $a = 0$ we have simply $u[0] = 1$, $u[k] = 0$, $k \geq 1$. It follows that values of $U(k, b, x)$ cannot be obtained from

$$U(a+k, b, x) = \frac{\Gamma(a)}{\Gamma(a+k)} u[k]$$

taking $a = 0$ (except for $k = 0$). However, by calling the procedure with $a = 1$, thus calling $chu(1, b, x, kmax, eps, u, uprime)$, we obtain

$$u[k] = \frac{\Gamma(1+k)}{\Gamma(1)} U(1+k, b, x), \quad k = 0, 1, \dots, kmax,$$

from which the values of $U(a, b, x)$ with positive integer values of a easily follow. Remark that a is not restricted to $[0, 1]$.

If $kmax$ or a are very large, underflow will occur in $u[k]$ as can be seen from the first of (2.4).

The (relative) accuracy can be controlled by eps . It is used to control the truncation errors in the approximation processes. Rounding errors are not considered. They may become important, although the recursions in chu are all strictly stable (no significant subtractions). Successive smaller choices of eps may yield worse results, especially if eps is of the same size as the machine accuracy. The relative accuracy in the external procedures $recip\ gamma$ and $sinh$ (called by $bess\ ka$) is about 10^{-14} .

The user may avoid a call of the Bessel function procedure $bess\ ka$ (with its external procedures $recip\ gamma$ and $sinh$) by skipping the part of the conditional statement announced by if $x \leq 1.4$ then. In that case the procedure will use the Miller algorithm for all $x > 0$. This will result in a less efficient algorithm with more rounding errors as x becomes smaller.

4.2. The procedure uabx

For convenience we supply a function procedure which gives directly the value of $U(a, b, x)$ for $a \in \mathbb{R}$, $b \in \mathbb{R}$, $x > 0$. This procedure calls for chu of the previous subsection. As a second value it delivers $U'(a, b, x)$.

The heading of the procedure is:

```
real procedure uabx (a, b, x, eps, uprime); value a, b, x, eps;
real a, b, x, eps, uprime;
```

The meaning of the formal parameters is:

a, b, x : <arithmetic expressions>;
the parameters of $U(a, b, x)$ and $U'(a, b, x)$;

$$a \in \mathbb{R}, b \in \mathbb{R}, x > 0.$$

eps : <arithmetic expression>;

the desired relative accuracy; *eps* > 0.

uprime: <identifier>;

exit: the value of $U'(a,b,x)$, the x -derivative of $U(a,b,x)$.

uabx : *uabx* := $U(a,b,x)$.

For underflow and overflow aspects and the role of *eps*, see the remarks in the description of *chu*.

4.3. Testing

The procedures can be compared with existing procedures for the computation of special cases of confluent hypergeometric functions. In this way we checked successfully the modified Bessel functions and the incomplete gamma functions. The procedure *chu* gives as special case the computation of the repeated integrals of the coerror function. GAUTSCHI (1977) wrote a FORTRAN program for computing

$$i^n \operatorname{erfc} x = \int_x^\infty i^{n-1} \operatorname{erfc} t \, dt, \quad n = 1, 2, \dots$$

with $i^0 \operatorname{erfc} x = \operatorname{erfc} x$, $i^{-1} \operatorname{erfc} x = (2/\sqrt{\pi})e^{-x^2}$. In terms of the U -functions we have for $x \geq 0$

$$i^n \operatorname{erfc} x = \pi^{-\frac{1}{2}} 2^{-n} e^{-x^2} U(\frac{1}{2}n + \frac{1}{2}, \frac{1}{2}, x^2),$$

with as special case the error function for $n = 0$.

The procedures can also be checked against themselves. The x -interval is divided in 3 subintervals: $(0, 1.4]$, $(1.4, 6.5]$, $(6.5, \infty)$. With

$$x_1^\pm = 1.4(1 \pm \delta), \quad x_2^\pm = 6.5(1 \pm \delta)$$

we computed $U(a,b,x_1^\pm)$ and $U(a,b,x_2^\pm)$ and we compared the results with each other. We also compared $U(a,a+n+1,x)$ with $U(a(1+\delta), a+n+1,x)$ (δ somewhat larger than the machine accuracy; the computer should recognize the poly-

nomial case in $U(a, a+n+1, x)$ and the non-polynomial case in $U(a(1+\delta), a+n+1, x)$. All tests were satisfactory. They were done on the CD CYBER 73 of SARA, Amsterdam (machine accuracy 2^{-48}).

The boundary points 1.4 and 6.5 for the x -intervals were obtained by comparing computing time for the following choice of the parameters in *uabx*

$$a = .32, \quad b = .56, \quad \text{eps} = 10^{-10}.$$

In the immediate neighbourhood of $x = 1.4$ the computing time was about 0.02 s, at $x = 6.5$ it was 0.03 s. A call of *chu* with large values of *kmax* and $|b|$ will require much more computing time.

Finally we give the starting index v for the Miller algorithm for $a = .5, b = 1, \text{eps} = 10^{-12}, kmax = 0$ and for several values of x :

x	1.41	2.0	6.5	6.6	10.0	50.0	100.0
v	52	40	19	38	28	12	9

4.4. Codes of *chu* and *uabx*

```

procedure chu(a,b,x,kmax,eps,u,uprime); value a,b,x,kmax,eps;
real a,b,x,eps,uprime; integer kmax; array u;
comment computes gamma(a + k) / gamma(a) * u(a + k, b, x)
for k= 0 (1) kmax, and the x-derivative of u(a+kmax,b,x)
if a<0 or x<0 or kmax<0 or eps<0 then
begin comment here the user can incorporate an output statement
with the message "parameters out of range"
end else
if a=0 then
begin u[0]:= 1; uprime:= 0;
for kmax:= kmax step -1 until 1 do u[kmax]:= 0
end else
if b<0 then
begin real c,d,e,w; integer j; array v[0:kmax];
c:= a-b+1; d:= x+(-b); chu(c,l-b,x,kmax,eps,v,w);
for j:= 0 step 1 until kmax do
begin e:= (a+j)*d; v[j]:= -e*v[j]; d:= e/(c+j) end;
uprime:= v[kmax]; u[kmax]:= -x*(uprime+e*w)/(a+kmax);
c:= c+x; for j:= kmax-1 step -1 until 0 do
u[j]:= (-x*v[j+1]+(c+j)*u[j+1])/(a+j)

```

```

end else
if b>1 then
begin real al,b1,c,d,e,f,g,h,u3,v,w; integer i,j,k,m,n;
boolean p,q,r,s;
procedure brec(a,b,k,f,g); value a,b,k; real a,b,f,g;
integer k;
begin k:= k-1; for i:= 0 step 1 until k do
begin h:= f-g; g:= ((i+b)×g-axf)/x; f:= h end
end brec;
n:= entier(b); b1:= b-n; al:= a+kmax; c:= b-a-1;
e:= c-x; m:= entier(e); if m=e then m:= m-1;
p:= c=entier(c) and c>0; q:= kmax<c; r:= kmax<m; s:= m>0;
if kmax=0 then r:= s:= true ;
if r then m:= kmax; k:= (if p then c else n)-m;
if ¬r then
begin if p and q then
begin g:= 1; i:= kmax-1; for j:= 0 step 1 until i do
g:= g×(j+a); f:= g×x+(-al); g:= -al×f/x;
brec(al,al+1,c-kmax,f,g)
end else
begin chu(a,b1,x,kmax,eps,u,u3); f:= u[kmax]; g:= u3;
brec(al,b1,n,f,g); if ¬p and s then
begin for j:= kmax step -1 until 1 do
u3:= u3-u[j]; v:= u[0]; w:= u3; d:= b1
end
end; n:= m+1; u[kmax]:= f; uprime:= g; if ¬s then n:= 0;
for j:= kmax-1 step -1 until n do
begin h:=(-x×g+(j-e)×f)/(a+j); g:= g-f; f:= u[j]:= h end
end; if s then
begin if p then
begin v:= x+(-a); w:= -a×v/x; d:= a+1 end else
if r then
begin chu(a,b1,x,0,eps,u,w); v:= u[0]; d:= b1 end;
brec(a,d,k,v,w); e:= b-n-x;
for j:= 0 step 1 until m do
begin if j=0 then
begin f:= u[0]:= v; g:= w end else
begin h:= -w; g:= w:= -((a+j-1)×v-(j+e)×w)/x; v:= u[j]:= h
end; brec(a+j,b+j-m,m-j,u[j],g)
end; if m=kmax then uprime:= g
end
end
end else
if x < 1.4 then
begin real d,delta,e,f,p,q,r,s,t,t0,t1,u0,u1,u2,u3,v,w,x2,y,z;
integer n,nu,k0,k1,i,j; array bb,bb1,bx[0:8],fi[-1:8];
n:= 9;

```

```

v:= 12.56637; r:= (x-v*(b+1))/2; d:=(v*n+r-sqrt(r*r+4*x*n*x))/(2*n);
if d<4.7124 then
begin d:= 4.7124; v:= w:= 1 end else
begin v:= abs(sin(d)); w:= v*(-1-b) end;
w:= w*exp(.5*x*(1/v+1/d));
delta:= eps*exp(-.5*x+(n-1-b)*ln(d))/w;
z:= .5/delta; v:= .5-b; i:= n-1;
for j:= 1 step 1 until i do z:= z*(j+v); i:= 0;
t:= sqrt(x)*z*(1/(2*n)); e:= ln(delta)+n-n*ln(x);
lab: r:= n+b+t; s:= 1+b+t; p:= ln(r); q:= ln(s);
f:= (ln(t+.5)-2*n*ln(t)+(r-.5)*p-(s-.5)*q-e)/
(1/(t+.5)-2*n/t+.5*(n-1)/(r*s)+p-q);
if f<0 then
begin t:= t-f; t:= sqrt(2*x+t*t); i:= i+1;
if i<10 then goto lab
end else k0:= 1+entier(t*t/x-a);
nu:= if kmax>k0 then 1+kmax else k0;
r:= a+nu; w:= sqrt(x/r); v:= 2*r*w;
bess ka(-b,v,t0,t1);
v:= w*(-b); bb[0]:= bbl[0]:= bx[0]:= 1;
ul:= fi[-1]:= v*t0; u0:= fi[0]:= v*w*t1; x2:= x*x;

bx[1]:= -x/12;
bx[2]:= x2/288;
bx[3]:= -x*(5*x2-72)/51840;
bx[4]:= x2*(5*x2-288)/2488320;
bx[5]:= -x*(x2*(7*x2-1008)+6912)/209018880;
bx[6]:= x2*(x2*(35*x2-10080)+279936)/75246796800;
bx[7]:= -x*(x2*(x2*(x2*5-2520)+176256)-746496)/902961561600;
bx[8]:= x2*(x2*(x2*(x2*5-4032)+566784)-9953280)/86684309913600;

bb[1]:= .5;
bb[2]:= (3*b-1)/24;
bb[3]:= b*(b-1)/48;
bb[4]:= (b*(b*(b*15-30)+5)+2)/5760;
bb[5]:= b*(b*(b*(b*3-10)+5)+2)/11520;
bb[6]:= (b*(b*(b*(b*(b*63-315)+315)+91)-42)-16)/2903040;
bb[7]:= b*(b*(b*(b*(b*(b*9-63)+105)+7)-42)-16)/5806080;
bb[8]:= (b*(b*(b*(b*(b*(b*(b*135-1260)+3150)-840)-2345)-
540)+404)+144)/1393459200;

for i:= 1 step 1 until n-1 do
begin
t0:= bb[i]; t1:= bbl[i]:= (b-i)*t0;
for j:= 1 step 1 until i-1 do
begin

```

```

        t0:= t0 + bb[i-j]×bx[j];
        t1:= t1 + bbl[i-j]×bx[j]
    end;
    t0:= bx[i]+bx×t0; t1:= t1+bx[i];
    fi[i]:= (x×fi[i-2]+(i-b)×fi[i-1])/r;
    u0:= u0 + t0×fi[i]; u1:= u1 + t1×fi[i-1]
end;
w:= 2×exp(x/2)/gamma(1+a);
u2:= wxu0; u3:= -wxu1; v:= a+1-b+x; kl:= nu-1;
for j:= kl step -1 until 1 do
begin u1:= (-x×u3+(v+j)×u2)/(a+j);
      u3:= u3-u2; u2:= u1; if j<kmax then u[j]:= axu2;
      if j=kmax then uprime:= axu3
end; u[0]:= -x×u3+v×u2;
if kmax=0 then uprime:= ax(u3-u2)
end else
begin real ar,br,cr,c,er,m0,m1,mr,p0,p1,p2,q,u1,u2,u3,v,w;
      integer k,n,r; boolean large x;
      procedure recursion;
      begin p2:= (br×p1-ar×p0)/cr; er:= er×ar/cr;
            r:= r+1; if large x then mr:= mr×(1+c/r); v:= er/p2;
            br:= br+2; cr:= cr+1; p0:= p1; p1:= p2
      end recursion;
      n:= entier(a); if a=n then n:= n-1; a:= a-n; kmax:= kmax+n;
      large x:= x>6.5 and a≠b;
      if large x then mr:= 1 else
      begin mr:= 0; if a=b then
      begin m0:= a; m1:= 1 end else
      begin m0:= 0; m1:= v:= 1;
            for r:= 1,r+1 while v>m1×eps do
            begin v:= v×x/r; m0:= m0+v; v:= v×(a+r)/(b+r); m1:= m1+v
            end;
            v:= exp(-x)×gamma(a+1)/gamma(b+1);
            m0:= v×(b+a×m0); m1:= v×m1
      end
      end;
      c:= a-b; cr:= 2+c; br:= x+a+cr; p0:= 0;
      v:= p1:= er:= 1; r:= 0;
      for ar:= a+r while r< kmax do recursion; w:= p0×p1/er;
      for ar:= a+r,ar+1 while v×(w/p0+mr×(2+a/r))>eps do recursion;
      c:= 1+c; v:= x+c; u2:= 1; w:= 0;
      u3:= -2×r/(x+sqrt(x×(x+4×r)));
      for r:= r-1,r-1 while r > 0 do
      begin if large x then
      begin w:= w+mr×u2; mr:= mr×(r+1)/(c+r) end;
            u1:= (-x×u3+(v+r)×u2)/(a+r); u3:= u3-u2; u2:= u1;

```

```

    if r>n and r<kmax then u[r-n]:= u2;
    if r=kmax then uprime:= u3
end;
u1:= -x*u3+v*u2; u3:= u3-u2; v:= a; k:= n-1;
if kmax=0 then uprime:= u3; kmax:= kmax-n;
w:= if large x then x+(-a)/(a*(w+c*u2)+u1) else
    x+(-b)/(u1*m1-u3*m0);
for r:= 0 step 1 until k do v:= v/(a+r);
if n=0 then begin k:= 1; u[0]:= w*u1 end else k:= 0;
w:= v*w; uprime:= w*uprime;
for r:= k step 1 until kmax do u[r]:= w*u[r]
end chu;

real procedure uabx(a, b, x, eps, uprime); value a, b, x, eps;
real a, b, x, eps, uprime;
begin real al, c, p, q, r; integer j, n; array u[0:0];
    n:= if a < 0 then entier(a) else 0; q:= al:= a-n; u[0]:= 1;
    if n<0 and a=b then
        begin if al>0 then chu(al,al,x,0,eps,u,q);
            p:= u[0]; r:= p-q;
            for j:= 1 step 1 until -n do
                begin r:= x*r; q:= (al-j)*p; p:= r-q end
            end else
                begin if al > 0 then chu(al,b,x,0,eps,u,q);
                    c:= 1+al-b+x; al:= al-1; p:= u[0];
                    for j:= 1 step 1 until -n do
                        begin r:= (c-j)*p-x*q; q:= (al-j)*(q-p); p:= r end;
                    end; uabx:= p; uprime:= q
                end
            end
end uabx;

```

ACKNOWLEDGEMENT

The author is pleased to acknowledge the help of R. Montijn, who tested and programmed various versions of the procedures, and of J.P. Hollenberg who computed the arrays *bx*, *bb* of the ALGOL 60 program.

REFERENCES

ABRAMOWITZ, M.A. & I.A. STEGUN (1964), *Handbook of mathematical functions*, Nat. Bur. Standards Appl. Math. Ser. 55, Washington D.C..

- CAMPBELL, J.C. (1980), *On Temme's algorithm for the modified Bessel function of the third kind*, (Submitted for publication).
- GAUTSCHI, W. (1967), *Computational aspects of three-term recurrence relations*, SIAM Rev. 9, 24-82.
- GAUTSCHI, W. (1977), *Evaluation of the repeated integrals of the coerror function*, ACM Trans. Math. Software 3, 240-252.
- OLVER, F.W.J. (1967), *Numerical solution of second-order linear difference equations*, J. Res. NBS 71B, Nos. 2 and 3, 111-129.
- SLATER, L.J. (1960), *Confluent hypergeometric functions*, Cambridge Univ. Press.
- TEMME, N.M. (1975a), *Numerical evaluation of functions arising from transformations of formal series*, J. Math. Anal. Appl. 51, 678-694.
- TEMME, N.M. (1975b), *On the numerical evaluation of the modified Bessel function of the third kind*, J. Comput. Physics 19, 324-337.
- TEMME, N.M. (1979), *On the expansion of confluent hypergeometric functions in terms of Bessel functions*, Report TW 192/79, Mathematical Centre, Amsterdam. To appear in J. Comput. Appl. Math.