BIT 2 (1962), 232—255

AN ARSENAL OF ALGOL PROCEDURES FOR COMPLEX ARITHMETIC*

P. WYNN

Abstract.

This paper contains a complete system of ALGOL procedures which enable arithmetic operations to be carried out upon complex numbers. Further procedures for carrying out the evaluation of certain elementary functions (e.g. ln, exp, sin, ...) of a complex variable are given. Application of these procedures is then illustrated by their use in the computation of the confluent hypergeometric function and the Weber parabolic cylinder function. Procedures relating to the application of the ε-algorithm to series of complex terms are described. Two integrated series of procedures, relating to Stieltjes type S-fractions and to corresponding continued fractions respectively, are given. Complete programmes, which illustrate the use of these procedures, may be used for the computation of the incomplete β-function, the incomplete Γ-function (of arguments of large and small modulus) and the Weber function.

Introduction.

The purposes of this note are to present a complete system of Algol [1] procedures for carrying out arithmetic operations upon complex numbers, to describe the convention which governs their use, and to give some examples of their application.

A Convention.

We stipulate that all complex numbers are to be represented by real arrays containing at least two members. There is an integer i which is declared globally throughout the block in which the complex arithmetic takes place, and all complex numbers (e.g. z, l_s) may be recognized by virtue of the fact that they contain the index i (e.g. $z[i]$, $l[s,i]$). i takes two values, zero corresponding to the real part (e.g. $\mathrm{Re}(z) \equiv z[0]$, $\mathrm{Re}(l_s) \equiv l[s,0]$) and unity corresponding to the imaginary part. The integer i may not, therefore, (except in circumstances which are difficult to envisage) be used for any other purpose.

The most general form of the input to any of these procedures (with

* Communication MR 51 of the Computation Department of the Mathematical Centre, Amsterdam.

the exception of those relating to reading) is $a \times x + b \times y + c \times z + \ldots$ where $a, b, c, \ldots$ are real numbers, and $x, y, z, \ldots$ are complex. The output of these procedures (except for those relating to the reading and printing of numbers) is, in the terminology of [1], of type **real**.

Procedures.

The system of procedures is now described.

1) An assignment procedure, corresponding to the normal *one* := *other*

procedure *eq*(*one*, *other*); **real** *one*, *other*; **for** *i* := 0,1 **do** *one* := *other*;

This procedure may, of course, be used for the addition and subtraction of complex numbers.

2) A double assignment procedure corresponding to *third* := *second* := *first*

procedure *seqeq*(*third*, *second*, *first*); **real** *third*, *second*, *first*;
for *i* := 0,1 **do** *third* := *second* := *first*;

3) Complex multiplication, corresponding to *res* := *one* × *other*

procedure *cm*(*res*, *one*, *other*); **real** *res*, *one*, *other*;
begin real *Reone*, *Imone*, *Reother*, *Imother*;
i := 0; *Reone* := *one*; *Reother* := *other*;
i := 1; *Imone* := *one*; *Imother* := *other*;
res := *Reone* × *Imother* + *Imone* × *Reother*; *i* := 0;
res := *Reone* × *Reother* − *Imone* × *Imother* **end**;

4) Complex division, corresponding to *res* := *one*/*other*

procedure *cd*(*res*, *one*, *other*); **real** *res*, *one*, *other*;
begin real *Reone*, *Imone*, *Reother*, *Imother*, *denom*;
i := 0; *Reone* := *one*; *Reother* := *other*;
i := 1; *Imone* := *one*; *Imother* := *other*;
denom := *Reother* × *Reother* + *Imother* × *Imother*;
res := (*Imone* × *Reother* − *Reone* × *Imother*)/*denom*;
i := 0; *res* := (*Reone* × *Reother* + *Imone* × *Imother*)/*denom* **end**;

5) It is necessary to ensure that numbers which occur in the arithmetic as real numbers are treated as such (i.e. with their imaginary parts put equal to zero); for this purpose we have the following

real procedure *real*(*variable*); **real** *variable*;
real := (**if** *i* = 0 **then** *variable* **else** 0.0);

6) Similarly, for pure imaginary quantities we have the

real procedure *imaginary*(*variable*); **real** *variable*;
imaginary := (**if** *i* = 0 **then** 0.0 **else** *variable*);

7) The complex conjugate of a complex quantity is taken by the

real procedure *compconj*(*it*); **real** *it*; *compconj* := (**if** *i* = 0 **then** *it* **else** −*it*);

8) The modulus of a complex number is given by the

real procedure *mod*(*it*); **real** *it*; **begin real** *Reit*, *Imit*;
i := 0; *Reit* := *it*; *i* := 1; *Imit* := *it*;
mod := sqrt(*Reit* × *Reit* + *Imit* × *Imit*) **end**;

9) The argument of a complex number z is assumed to satisfy the inequalities $-\pi < \arg(z) \leqq \pi$. It is computed by the
real procedure *arg*(*it*); **real** *it*; **begin real** *Reit*, *Imit*;
i := 0; *Reit* := *it*; *i* := 1; *Imit* := *it*;
arg := (**if** *Reit* > 0.0 **then** arctan(*Imit*/*Reit*) **else**
sign(*Imit*) × 1.57079 63267 949 − arctan(*Reit*/*Imit*)) **end**;

10) The complex number *res* is translated from the polar form $re^{i\vartheta}$ by means of the following
procedure *polar form*(*res*, *r*, *theta*); **real** *r*, *theta*; **begin real** *r*1, *theta*1;
*r*1 := *r*; *theta*1 := *theta*; *i* := 0; *res* := *r*1 × cos(*theta*1);
i := 1; *res* := *r*1 × sin(*theta*1) **end**;

11) In formulae containing complex quantities, the variable z often occurs in the combination iz. To compute this we have the
procedure *imult*(*res*, *it*); **real** *res*, *it*; **begin real** *aux*;
i := 0; *aux* := *it*; *i* := 1; *res* := *aux*; *aux* := *it*; *i* := 0; *res* := −*aux*
end;

12) Similarly z^2 is computed by means of the
procedure *compsq*(*res*, *it*); **real** *res*, *it*;
begin real *Reit*, *Imit*; *i* := 0; *Reit* := *it*; *i* := 1; *Imit* := *it*;
res := 2.0 × *Reit* × *Imit*; *i* := 0; *res* := *Reit* × *Reit* − *Imit* × *Imit* **end**;

13) Further, $1/it$ is computed by means of the
procedure *comprecip*(*res*, *it*); **real** *res*, *it*; **begin real** *Reit*, *Imit*, *denom*;
i := 0; *Reit* := *it*; *i* := 1; *Imit* := *it*; *denom* := *Reit* × *Reit* + *Imit* × *Imit*;
res := −*Imit*/*denom*; *i* := 0; *res* := *Reit*/*denom* **end**;

14) As the first of the functional procedures, corresponding to *res* := sqrt(*it*) we have the
procedure *compsqrt*(*res*, *it*); **real** *res*, *it*;
polar form(*res*, sqrt(*mod*(*it*)), 0.5 × *arg*(*it*));

15) Corresponding to *res* := ln(*it*), we have the
procedure *comp ln*(*res*, *it*); **real** *res*, *it*; **begin real** *aux*;
aux := ln(*mod*(*it*)); *i* := 0; *res* := *aux*; *aux* := *arg*(*it*); *i* := 1;
res := *aux* **end**;

16) Corresponding to *res* := exp(*it*), we have the
procedure *comp exp*(*res*, *it*); **real** *res*, *it*; **begin real** *aux*1, *aux*2;
i := 0; *aux*1 := exp(*it*); *i* := 1; *aux*2 := *it*;
res := *aux*1 × sin(*aux*2); *i* := 0; *res* := *aux*1 × cos(*aux*2) **end**;

17) We now introduce procedures for the computation of trigonometric functions. We could of course use a formula such as

$$\sin(z) = (\exp(iz) - \exp(-iz))/2i$$

in conjunction with the preceding procedure. However this would result in a loss of relative accuracy in the neighbourhood of the origin. To avoid this we use a formula such as

$$\sin(x+iy) = \sin(x)\cosh(y)+i\cos(x)\sinh(y)$$

and use a special Chebyshev expansion for evaluating the hyperbolic functions in which relative accuracy near the origin is preserved. The procedure for evaluating the hyperbolic functions is:

```
procedure hyp(sinh, cosh, y); value y; real sinh, cosh, y;
begin real y1; y1 := exp(y); cosh := 0.5 × (y1 + 1.0/y1);
if abs(y) ≧ 1.0 then sinh := 0.5 × (y1 − 1.0/y1) else
begin integer r; real br, brplus1, brplus2; array CWC[0:5];
CWC[0] := 1.13031 82079 8497; CWC[1] := 4.43368 49848 66₁₀−2;
CWC[2] := 5.42926 31191₁₀−4; CWC[3] := 3.19843 646₁₀−6;
CWC[4] := 1.10367 7₁₀−8; CWC[5] := 2.498₁₀−11;
brplus1 := brplus2 := 0.0; y1 := 2.0 × (2.0 × y × y − 1.0);
for r := 5 step −1 until 0 do begin br := y1 × brplus1 − brplus2 + CWC[r];
if r ≠ 0 then begin brplus2 := brplus1; brplus1 := br end end;
sinh := y × (br − brplus1) end end;
```

The Chebyshev coefficients have been extracted from [27]. The coefficients are the odd-order coefficients in the Chebyshev polynomial expansion for $\exp(x)$. Thus if this procedure is to be written at the machine code level and a Chebyshev polynomial expansion for $\exp(x)$ is already present the coefficients in the latter may be made use of.

Thus corresponding to $res := \sin(it)$, we have the

```
procedure compsin(res, it); real res, it;
begin real Reit, Imit, sinhImit, coshImit;
i := 0; Reit := it; i := 1; Imit := it; hyp(sinhImit, coshImit, Imit);
res := cos(Reit) × sinhImit; i := 0; res := sin(Reit) × coshImit end;
```

18) Corresponding to $res := \cos(it)$, we have the

```
procedure compcos(res, it); real res, it;
begin real Reit, Imit, sinhImit, coshImit;
i := 0; Reit := it; i := 1; Imit := it; hyp(sinhImit, coshImit, Imit);
res := −sin(Reit) × sinhImit; i := 0; res := cos(Reit) × coshImit end;
```

19) Corresponding to $res := \tan(it)$ (not given in [1]), we have the

```
procedure comptan(res, it); real res, it;
begin real Reit, Imit, sinhImit, coshImit, sinReit, cosReit;
array aux1, aux2[0:1]; i := 0; Reit := it; i := 1; Imit := it;
hyp(sinhImit, coshImit, Imit);
sinReit := sin(Reit); cosReit := cos(Reit);
aux1[1] := cosReit × sinhImit; aux2[1] := −sinReit × sinhImit;
```

*aux*1[0] := *sin Reit* × *cosh Imit*; *aux*2[0] := *cos Reit* × *cosh Imit*;
cd(*res*, *aux*1[*i*], *aux*2[*i*]) **end**;

Clearly this could be written a little more concisely with the aid of the previous two procedures, but the result would be somewhat repetitive and wasteful with regard to time.

20) The inverse function arcsine, corresponding to *res* := arcsin(*it*), (also not given in [1]) may be computed by means of the

procedure *comp arcsin*(*res*, *it*); **real** *res*, *it*; **begin real** *x*, *y*, *d*1, *d*2, *d*3, *d*4;
i := 0; *x* := *it*; *i* := 1; *y* := − *it*;
*d*3 := *x* × *x*; *d*4 := *y* × *y*; *d*1 := *d*3 + *d*4; *d*2 := *d*3 − *d*4;
*d*3 := *d*1 × *d*1 − 2.0 × *d*2; *d*4 := sqrt(*d*3 + 1.0);
res := sign(*y*) × 0.5 × ln(*d*1 + *d*4 + sqrt(*d*3 + *d*1 × (*d*1 + 2.0 × *d*4))); *i* := 0;
res := arctan(*x* × sqrt(2.0/(1.0 − *d*2 + *d*4))) **end**;

21) Corresponding to *res* := arccos(*it*) (also not given in [1]), we have the

procedure *comp arccos*(*res*, *it*); **real** *res*, *it*; **begin array** *aux*[0:1];
comp arcsin(*aux*[*i*], *it*); eq(*res*, *real*(1.57079 63267 949) − *aux*[*i*]) **end**;

22) Corresponding to *res* := arctan(*it*) we have the

procedure *comp arctan*(*res*, *it*); **real** *res*, *it*; **begin array** *aux*1, *aux*2[0:1];
eq(*aux*1[*i*], *it*); *compsq*(*aux*2[*i*], *aux*1[*i*]);
compsqrt(*aux*2[*i*], *real*(1.0) + *aux*2[*i*]);
cd(*aux*2[*i*], *aux*1[*i*], *aux*2[*i*]); *comp arcsin*(*res*, *aux*2[*i*]) **end**;

23) Corresponding to *res* := *one* ↑ *other*, we have the

procedure *onehochother*(*res*, *one*, *other*); **real** *res*, *one*, *other*;
begin array *aux*1[0:1]; *comp ln*(*aux*1[*i*], *one*); *cm*(*aux* 1[*i*], *other*, *aux*1[*i*]);
comp exp(*res*, *aux*1[*i*]) **end**;

A number of procedures relating to the input and output of complex numbers are now given. No provisions are made in [1] for dealing with the input and output of numbers; nevertheless the following procedures may be of use with many installations.

24)

procedure *compread*(*it*); **real** *it*; **for** *i* := 0, 1 **do** *it* := *read*;

25)

procedure *readpolarform*(*it*); **real** *it*; **begin real** *r*, *theta*;
r := *read*; *theta* := *read*; *i* := 0; *it* := *r* × cos(*theta*);
i := 1; *it* := *r* × sin(*theta*) **end**;

26)

procedure *compprint*(*it*); **real** *it*; **for** *i* := 0, 1 **do** *print*(*it*);

27)

procedure *druck*(*it*); **real** *it*; **begin** *compprint*(*it*); *print*(*mod*(*it*)) **end**;

28)

procedure *printpolarform*(*it*); **real** *it*;

begin *print*(*mod*(*it*)); *print*(*arg*(*it*)) **end**;
29) To print the vector of complex numbers it_j $(j=h,h+1,\ldots,k)$ in rows of *col* numbers, the real parts being immediately above the imaginary parts, we have the*
procedure *printcompvect*(*it*,*j*,*h*,*k*,*col*); **value** *h*, *k*, *col*;
integer *j*, *h*, *k*, *col*; **real** *it*; **begin integer** *janfang*;
for *janfang* := *h* **step** *col* **until** *k* **do for** *i* := 0,1 **do begin** *NLCR*;
for *j* := *janfang* **step** 1 **until** *janfang*+*col*−1 **do if** *j* ≦ *k* **then** *print*(*it*)
end end;
30) Finally the following general purpose
boolean procedure *even*(*integer*); **integer** *integer*;
even := (*integer* = (*integer* ÷ 2) × 2);
is given.

Examples.

Some examples of the application of the preceding procedures will now be given. It must be emphasized however that these examples are to be regarded as no more than a preliminary essay in the computation of functions of a complex variable; they certainly do not represent an assemblage of foolproof procedures.

A further matter deserving special comment is that the definition of the argument of a complex variable has been made quite arbitrarily. Consequences of the definition chosen are that the procedure *compsqrt* (*res*, *it*) maps the whole of the *it*-plane onto the right hand half of the *res*-plane, the procedure *compln*(*res*, *it*) maps the *it*-plane onto the strip $-\pi < \mathrm{Im}(res) \leqq \pi$, and so on. If the reader wishes to adopt another convention, he must first modify the procedure *arg*(*it*).

Despite the foregoing warnings, if the given procedures and the examples which are to be given are used with discretion, they may be made to yield useful results.

The Confluent Hypergeometric Function.

The following procedure computes (to a given relative accuracy in the modulus) the numerical sum of the confluent hypergeometric series

$$ {}_1F_1(a;c;z) = 1+\frac{a}{c}\frac{z}{1!}+\frac{a(a+1)}{c(c+1)}\frac{z^2}{2!}+\ldots : $$

procedure *oneFone*(*result*, *a*, *c*, *argument*, *relacc*);
value *a*, *c*, *relacc*; **real** *result*, *a*, *c*, *argument*, *relacc*;
begin integer *s*; **array** *term*, *sum*, *relerror*, *z*[0:1];

* Here and subsequently NLCR = New Line Carriage Return.

```
eq(z[i], argument); seqeq(term[i], sum[i], real(1.0)); s := 1;
TERM: cm(term[i], term[i], z[i] × (a+s−1)/((c+s−1) × s));
eq(sum[i], sum[i]+term[i]); s := s+1; cd(relerror[i], term[i], sum[i]);
if mod(relerror[i]) > relacc then go to TERM;
eq(result, sum[i]) end;
```

If the reader has punched the above procedures he may care to test some of them with the following complete programme:

```
begin integer i; array check[0:1];
comment This comment must be replaced by the procedures eq, seqeq,
cm, cd, real, imaginary, compprint, oneFone and mod;
oneFone(check[i], 1.0, 1.0, imaginary (1.57079 63267 949), 1.0₁₀−6);
compprint(check[i]) end
```

If the reader is still inclined to use the given procedures without due forethought, then let him construct a programme which contains the following segment:

```
oneFone(check[i], 1.0, 1.0, real(−50.0), 1.0₁₀−10);
druck(check[i]);
oneFone(check[i], 1.0, 1.0, real(50.0), 1.0₁₀−10);
comprecip(check[i], check[i]); druck(check[i]);
```

A more useful application of the procedure *oneFone* is provided by the computation of the Weber parabolic cylinder function

$$S_1(a;z) = 2^{-a/2-1/4}\pi^{1/2}e^{-z^2/4}\left\{\frac{{}_1F_1\left(\frac{a}{2}+\frac{1}{4};\frac{1}{2};\frac{1}{2}z^2\right)}{\Gamma\left(\frac{a}{2}+\frac{3}{4}\right)} - 2^{1/2}z\,\frac{{}_1F_1\left(\frac{a}{2}+\frac{3}{4};\frac{3}{2};\frac{1}{2}z^2\right)}{\Gamma\left(\frac{a}{2}+\frac{1}{4}\right)}\right\}$$

For the evaluation of the Γ-function of real argument (to a relative accuracy of $1.2_{10}-12$) we give the following procedure based upon data given by C.-E. Fröberg [2]:

```
real procedure realgammafunction(argument); value argument;
                                                    real argument;
begin if argument > 2.0 then
realgammafunction :=
(argument − 1.0) × realgammafunction (argument − 1.0)
else if argument < 1.0 then
realgammafunction := realgammafunction(argument + 1.0)/argument else
begin integer r, s; real x1; array CEF[0:1, 0:5]; x1 := argument − 1.0;
```

```
CEF[0,0] := 0.99999 99999 9888; CEF[0,1] := 0.28789 44308 9491;
CEF[0,2] := 0.16274 01560 5949; CEF[0,3] := 1.42275 09863 39_10-2;
CEF[0,4] := 6.72642 58470 1_10-3; CEF[0,5] := -6.20265 03473_10-4;
CEF[1,0] := 1.0; CEF[1,1] := 0.86511 00954 8768;
CEF[1,2] := -0.32696 07263 5275; CEF[1,3] := -0.12266 28393 2781;
CEF[1,4] := 6.26494 50210 1_10-2; CEF[1,5] := -7.16772 23899 3_10-3;
for r := 0,1 do for s := 4 step -1 until 0 do
CEF[r,5] := CEF[r,5] × x1 + CEF[r,s];
realgammafunction := CEF[0,5]/CEF[1,5] end end;
```

The complete programme for evaluating the Weber parabolic cylinder function by means of an ascending power series is then:

```
begin integer i; real a, eps, rg1, rg2, ratio, eps1, eps2;
array z, zsquared, aux1, aux2[0:1];
comment This comment must be replaced by the procedures eq, seqeq, cm,
cd, real, mod, compsq, compexp, compread, compprint, druck, oneFone and
realgammafunction;
a := read; compread(z[i]); eps := read;
NLCR; print(a); compprint(z[i]); print(eps);
compsq(zsquared[i], z[i]); rg1 := realgammafunction(0.75+a/2);
rg2 := 0.70710 67811 8655 × realgammafunction(0.25+a/2);
ratio := abs(rg1 × mod(z[i])/rg2);
if ratio < 1.0 then begin eps1 := eps; eps2 := eps/ratio end else
begin eps1 := eps × ratio; eps2 := eps end;
oneFone(aux1[i], 0.25+a/2, 0.5, zsquared[i]/2, eps1);
oneFone(aux2[i], 0.75+a/2, 1.5, zsquared[i]/2, eps2);
cm(aux2[i], z[i], aux2[i]); eq(aux1[i]/rg1-aux2[i]/rg2);
compexp(aux2[i], -zsquared[i]/4.0);
cm(aux1[i],
1.77245 38509 05516 × (2.0 ↑ (-0.25-a/2.0)) × aux2[i], aux1[i]);
druck(aux1[i]) end
```

It is assumed for simplicity that in the addition of the two constituent terms in braces, no loss of accuracy due to cancellation takes place. There is a preliminary skirmish to determine the relative accuracies to which both series must be evaluated, subsequent to which the confluent hypergeometric functions are evaluated and the function $S_1(a; z)$ computed. In a more extensive programme, the extent of the loss of figures, both in the evaluation of the separate sums and in their combination, would be estimated a posteriori and the relative accuracy requirements correspondingly increased.

The ε-algorithm [3].

The ε-algorithm is a computational device for accelerating the convergence of a slowly convergent series. The relationships

$$\varepsilon_{s+1}^{(m)} = \varepsilon_{s-1}^{(m+1)} + (\varepsilon_s^{(m+1)} - \varepsilon_s^{(m)})^{-1}$$

are applied to the initial values

$$\varepsilon_0^{(m)} = \sum_{s=0}^{m-1} u_s, \quad \varepsilon_1^{(m)} = u_m^{-1} \qquad (m = 0, 1, \ldots).$$

In certain cases the even order sequences $\varepsilon_{2s}^{(m)}$ $(s = 0, 1, \ldots)$ converge to the formal sum $\sum_{s=0}^{\infty} u_s$ far more rapidly than do the partial sums of the latter. The quantities $\varepsilon_s^{(m)}$ may be placed in a triangular array in which the superscript m denotes a forward diagonal and the suffix s a column.

The ε-algorithm is a lozenge algorithm. The constituents of the ε-algorithm relationship occur in the following lozenge

$$\begin{array}{ccc} & \varepsilon_s^{(m)} & \\ \varepsilon_{s-1}^{(m+1)} & & \varepsilon_{s+1}^{(m)} \\ & \varepsilon_s^{(m+1)} & \end{array}$$

For the evaluation of the ε-array use may therefore be made of a one-dimensional array l_s of complex numbers. At any given moment this array contains the following quantities: $\varepsilon_0^{(m)}(\equiv l_0)$, $\varepsilon_1^{(m-1)}(\equiv l_1)$, $\varepsilon_2^{(m-2)}(\equiv l_2), \ldots, \varepsilon_m^{(0)}(\equiv l_m)$. We arrive with a new term of the original series and replace in succession this sequence by $\varepsilon_0^{(m+1)}, \varepsilon_1^{(m)}, \varepsilon_2^{(m-1)}, \ldots, \varepsilon_m^{(1)}$, and add $\varepsilon_{m+1}^{(0)}$. This proces requires three auxiliary complex number storage locations (*aux*0, *aux*1, *aux*2).

The following procedure will print out the moduli of the numbers in the even order columns of the ε-array (in vertical strips of *col* columns, to be glued together subsequently). As well as the non-local integer i, the procedure uses a non-local (two member) array *term*, and a non-local integer m indicating the suffix of the term.

```
procedure display complex epsilon algorithm(mmax, compute term, col);
value mmax, col; integer mmax, col; procedure compute term;
begin integer s, sanfang, twommax;
array l[0:mmax+1,0:1], display[1:((mmax+1)×(mmax+5))÷4],
                                    aux0, aux1, aux2[0:1];
eq(l[0,i],0.0); twommax := 2×mmax; for m := 0 step 1 until mmax do
begin compute term; eq(aux1[i], term[i]+l[0,i]);
for s := 0 step 1 until m do
begin compprecip(aux0[i], (if s = 0 then term[i] else aux[i]-l[s,i]));
```

```
if s ≠ 0 then
begin eq(aux0[i], aux0[i] + l[s − 1, i]); eq(l[s − 1, i], aux2[i]) end;
if even(s) then
display[(s × (twommax − s + 2)) ÷ 4 + m + 1] := mod(aux1[i]);
eq(aux2[i], aux1[i]); eq(aux1[i], aux0[i]) end;
eq(l[m, i], aux2[i]); eq(l[m + 1, i], aux1[i]);
if ¬even(m) then
display[((m + 1) × (twommax − m + 5)) ÷ 4] := mod(aux1[i]) end;
for sanfang := 0 step 2 × col until mmax + 1 do begin NLCR;
for m := 1 step 1 until mmax + 1 − sanfang ÷ 2 do begin NLCR;
for s := sanfang step 2 until sanfang + 2 × (col − 1) do
begin if s ÷ 2 ≦ m ∧ m ≦ mmax + 1 − (s ÷ 2)
then print(display[(s × (twommax + 4 − s)) ÷ 4 + m]) end end end end;
```

The power of the ε-algorithm may be illustrated by applying it to the series

$$\ln(2) = \sum_{m=0}^{\infty} (-1)^m/(m+1) .$$

The terms in this series may be computed by means of the following

```
procedure ln2; eq(term[i], real((if even(m) then 1.0 else −1.0)/(m + 1)));
```

The application of the ε-algorithm to the above series may be displayed by the following programme:

```
begin integer i, m; array term[0:1];
comment This comment must be replaced by the procedures eq, real,
mod, comprecip, even, display complex epsilon algorithm and ln2;
display complex epsilon algorithm(5, ln2, 6) end
```

It produces the following array:

1.0	0.66667		
0.5	0.7	0.69231	
0.83333	0.69048	0.69333	0.69312
0.58333	0.69444	0.69309	
0.78333	0.69242		
0.61667			

(The quantity $|\varepsilon_0^{(0)}|$ is missing from this array, but since this is always zero, no information has been lost in this manner.)

The above procedure should only be used in a provisional inquiry as to whether or not application of the ε-algorithm is likely to be successful. In the case of the example considered, this seems likely.

The following procedure computes (to a prescribed relative accuracy in the modulus) the transformed formal sum of the series whose successive terms are computed by means of the procedure *compute term*.

It has as input an integer (M say) which indicates that there is storage space available for $M+1$ complex numbers.

Included in the output is a boolean variable which has the value **true** if this storage space is not exceeded.

```
procedure complex epsilon algorithm(result, compute term, relative accuracy,
available storage, storage not exceeded);
value relative accuracy, available storage;
integer available storage; real result, relative accuracy;
boolean storage not exceeded; procedure compute term;
begin integer s; array aux0, aux1, aux2[0:1], l[0:available storage, 0:1];
eq(l[0,i], 0.0); m := 0;
EPSALG: compute term; eq(aux1[i], term[i]+l[0,i]); s := 0;
EPSLOOP: comprecip(aux0[i], (if s = 0 then term[i] else aux1[i]-l[s,i]));
if s ≠ 0 then
begin eq(aux0[i], aux0[i]+1[s-1,i]); eq(l[s-1,i], aux2[i]) end;
eq(aux2[i], aux1[i]); eq(aux1[i], aux0[i]); s := s+1;
if m ≧ s then go to EPSLOOP;
eq(l[m,i], aux2[i]); eq(l[m+1, i], aux1[i]);
if m > 0 then begin if ¬even(m) then eq(aux2[i], l[m-1,i]) else
begin eq(aux1[i], aux2[i]); eq(aux2[i], l[m-2,i]) end;
cd(aux2[i], aux2[i], aux1[i]); if abs(1-mod(aux2[i])) < relative accuracy
                                                              then
begin storage not exceeded := true; eq(result, aux1[i]); go to END end end;
m := m+1; if m < available storage then go to EPSALG
else storage not exceeded := false;
END: end;
```

Its use may be illustrated by the following complete programme:

```
begin integer i, m; boolean successful; array term, check[0:1];
comment This comment must be replaced by the procedures eq, cd, real,
mod, comprecip, compprint, even, ln2 and complex epsilon algorithm;
complex epsilon algorithm(check[i], ln2, +1.0₁₀-9, +2000, successful);
if successful then compprint(check[i]) end
```

A more useful example of the application of the ε-algorithm is provided by the problem of mapping the interior of the ellipse $x=a\cos(t)$, $y=b\sin(t)$ in the $z-(\equiv x+iy)$ plane by means of the transformation $w=f(z)$ onto the interior of the circle $|w|=1$ in the w-plane, such that

the origin is preserved and the point $z=a+i0$ becomes $w=1+i0$. The following series solution of this problem has been given by Szegö [4]

$$\ln(f(z)) = \ln\left(\frac{2z}{R}\right) + \sum_{m=1}^{\infty} \frac{(-1)^m}{m} \frac{2}{R^{4m}+1} T_{2m}(z)$$

where

$$2z = u+u^{-1},\ 2T_m(z) = u^m+u^{-m}.$$

Here $R=a+b$, and it is assumed that $a-b=1$ (this is not an essential restriction and corresponds to a real multiplicative factor of $f(z)$).

The terms in this series may be computed by means of the

```
procedure Gabor Szegö(argument, a, b); value a, b; real argument, a, b;
begin own real R4, R4m, twominus1m;
own real array fourzsquaredminus2, T2mminus4, T2mminus2, T2m[0:1];
if m = 0 then begin real R, R2; array z, zsquared[0:1];
R := a+b; R2 := R×R; R4 := R2×R2; R4m := 1.0;
twominus1m := 2.0; eq(z[i], argument);
compsq(zsquared[i], z[i]); eq(T2mminus4[i], 2×zsquared[i]-real(1.0));
eq(T2mminus2[i], real(1.0));
eq(fourzsquaredminus2[i], 4.0×zsquared[i]-real(2.0));
compln(term[i], 2.0×z[i]/R) end else
begin twominus1m := -twominus1m; R4m := R4×R4m;
cm(T2m[i], fourzsquaredminus2[i], T2mminus2[i]);
eq(T2m[i], T2mminus4[i]);
eq(T2mminus4[i], T2mminus2[i]); eq(T2mminus2[i], T2m[i]);
eq(term[i], twominus1m×T2m[i]/((R4m+1.0)×m)) end end;
```

This procedure may be tested by means of the following programme:

```
begin integer i, m, j, N, col, very end; real k, t, a, b, eps;
array z, check, term[0:1]; boolean successful;
comment This comment must be replaced by the procedures eq, cm, cd,
real, mod, compsq, comprecip, compprint, even, complex epsilon algorithm,
compln, arg and Gabor Szegö;
k := read; N := read; eps := read; very end := read;
NLCR; print(k); print(eps); b := 1.0/sqrt(k×k-1.0); a := k×b;
for j := 1 step 1 until N do
begin t := j×3.14159 26535 898/N; z[0] := a×cos(t); z[1] := b×sin(t);
complex epsilon algorithm(check[i], Gabor Szegö(z[i],a,b), eps, very end,
                                                          successful);
NLCR; if successful then compprint(check[i]) end end
```

Since, for the test values chosen, z is always on the ellipse $x=a\cos(t)$, $y=b\sin(t)$, the real part of the result (i.e. $\ln(|w|)$ on the unit circle) should be zero. The imaginary part of the result, which satisfies a certain Lichtenstein–Gershgorin equation, is tabulated for $k=1.2$ and 2.0 by Todd and Warschawski [5].

S-fractions.

The ε-algorithm has close connections [6] with the theory of certain types of continued fractions. The coefficients $q_r^{(m)}$, $e_r^{(m)}$ in the continued fraction

$$\sum_{s=0}^{m-1} c_s z^{-s-1} + z^{-m} \left\{ \frac{c_m}{z-} \frac{q_1^{(m)}}{1-} \frac{e_1^{(m)}}{z-} \cdots \frac{q_r^{(m)}}{1-} \frac{e_r^{(m)}}{z-} \cdots \right\}$$

may (except in certain singular cases) be determined by imposing the condition that the series expansion of its r^{th} $(r=0,1,\ldots)$ convergent should agree with the power series $\sum_{s=0}^{\infty} c_s z^{-s-1}$ as far as the term $c_{m+r-1}z^{-m-r}$. If the ε-algorithm is applied to the initial conditions $\varepsilon_0^{(m)}=\sum_{s=0}^{m-1} c_s z^{-s-1}$, $\varepsilon_1^{(m)}=c_m{}^{-1}z^{m+1}$ then the successive convergents of the above continued fraction lies on the staircase $\varepsilon_0^{(m)}$, $\varepsilon_0^{(m+1)}$, $\varepsilon_2^{(m)}$, $\varepsilon_2^{(m+1)}$, $\varepsilon_4^{(m)}$, $\varepsilon_4^{(m+1)}$, ... of the even column ε-array.

Continued fractions of the above form in which the $q_r^{(m)}$, $e_r^{(m)}$ are real and negative are known as real S-fractions. Such continued fractions have been extensively studied by Chebyshev [7], Stieltjes [8], Markoff [9] [10] [11] [12] [13] [14] [15] and others. They converge [16] for values of z not on the negative real axis if the series $\sum_{s=0}^{\infty} l_s$ diverges, where

$$l_0 = 1,\ l_1 = \frac{1}{q_1^{(m)}},\ l_{2s} = \frac{q_s^{(m)}}{e_s^{(m)}} l_{2s-2},\ l_{2s+1} = \frac{e_s^{(m)}}{q_{s+1}^{(m)}} l_{2s-1}\ (s = 1,2,\ldots).$$

These conditions relating to the coefficients of the continued fraction are not always immediately available. The first may be replaced by the following [17]: There exists a function $F_m(z)$ which has the asymptotic expansion $\sum_{s=0}^{\infty} c_{m+s} z^{-s-1}$ in the domain $\varepsilon \leqq \arg(z) \leqq \pi-\varepsilon$, where $0<\varepsilon<\pi/2$, $F_m(z)$ is analytic and $\operatorname{Im}\{F_m(z)\}<0$ for $I(z)>0$, and furthermore $F_m(z)$ has the asymptotic representation

$$F_m(z) = \frac{c_m}{z} + \frac{O(1)}{z \operatorname{Im}(z)} \qquad (c_m > 0);$$

the second by [18]

$$\liminf_{s\to\infty} \sqrt[s]{\frac{c_{m+s}}{(2s)!}} \qquad < \infty .$$

The successive convergents $A_{2,s}$ $(s=0,1,\ldots)$ of the continued fraction

$$b_0+\frac{a_1}{b_1+}\,\frac{a_2}{b_2+}\cdots\frac{a_s}{b_s+}\cdots$$

may be determined by evaluating the double sequence $A_{j,s}$ $(j=0,1;$ $s=0,1,\ldots)$ by means of the recursion

$$A_{j,s} = b_sA_{j,s-1}+a_sA_{j,s-2}$$

from the initial conditions

$$A_{0,-1} = 1,\ A_{0,0} = b_0,\ A_{1,-1} = 0,\ A_{1,0} = 1\ ,$$

when

$$A_{2,s} = A_{0,s}/A_{1,s} \qquad (s = 0,1,\ldots)\ .$$

It may occur that the quantities $A_{j,s}$ $(j=0,1)$ grow to exceed the capacity of the floating point arithmetic being used. This may be obviated by use of a suitable equivalence transformation. However, it has so far been the author's experience that if the quantities $A_{j,s}$ go out of range before the continued fraction has converged numerically, then there are better ways of evaluating the function concerned.

Firstly, let us assume that the coefficients of a convergent real S-fraction are available in the form of **real procedures** *c*0, *qr*, *er* or as real expressions involving the non-local integer *r*. The following procedure computes the value of this continued fraction to a given relative accuracy in the modulus of the result (assuming that the relative difference between one convergent and the next is greater than the relative difference between one convergent and the value of the continued fraction):

```
procedure qerealSfraction(result, c0, qr, er, argument, relerror);
value relerror; real result, c0, qr, er, argument, relerror;
begin integer j, S, Sdash; real partnum; boolean q;
array z, aux[0:1], A[0:2, 0:1, 0:1];
eq(z[i], argument); eq(A[0,0,i], real(c0)); eq(A[1,0,i],z[i]);
eq(A[1,1,i], real(1.0)); seqeq(A[0,1,i], A[2,1,i], 0.0);
r := Sdash := 1; S := 0; q := false;
go to CONVERGENT;
RECURSION: Sdash := S; S := 1-Sdash; q := ¬q;
partnum := -(if q then qr else er);
for j := 0,1 do if q then eq(A[j,S,i], A[j,Sdash,i]+partnum×A[j,S,i])
else
```

```
begin cm(aux[i],z[i],A[j,Sdash,i]);
eq(A[j,S,i], aux[i]+partnum×A[j,S,i]) end;
if ¬q then r := r+1;
CONVERGENT: cd(A[2,S,i], A[0,S,i], A[1,S,i]);
cd(aux[i], A[2,Sdash,i], A[2,S,i]);
if abs(1.0-mod(aux[i])) > relerror then go to RECURSION
else eq(result,A[2,S,i]) end;
```

It uses a non-local integer r indicating the suffices of the partial numerators.

This procedure may be used to evaluate the incomplete gamma function of large (non-negative) argument

$$\Gamma(\alpha,z) = \int_z^\infty t^{\alpha-1}e^{-t}dt \sim e^{-z}z^\alpha \left\{ \frac{1}{z+} \frac{1-\alpha}{1+} \frac{1}{z+} \cdots \frac{r-\alpha}{1+} \frac{r}{z+} \cdots \right\}.$$

Firstly the continued fraction in braces is evaluated, and the result is then multiplied by the factor $e^{-z}z^\alpha$ outside the braces.

```
begin integer i, r; real alpha, eps; array z, aux1, aux2[0:1];
comment This comment must be replaced by the procedures eq, seqeq,
cm, cd, real, mod, compexp, onehochother, compln, arg, compread, compprint,
druck and qerealSfraction;
alpha := read; eps := read; compread(z[i]);
NLCR; print(alpha); print(eps);
compprint(z[i]); compexp(aux1[i], -z[i]);
onehochother(aux2[i], z[i], real(alpha));
cm(aux1[i],aux1[i],aux2[i]);
qerealSfraction(aux2[i], 1.0, -r+alpha, -r, z[i], eps);
cm(aux1[i], aux2[i], aux1[i]); druck(aux1[i]) end
```

In general, of course, the coefficients $q_r^{(m)}$, $e_r^{(m)}$ relating to the sequence of coefficients c_s $(s=0,1,\ldots)$ are not available in closed form. They may be evaluated by applying the $(q-d)$ algorithm relationships of H. Rutishauser [19]

$$q_r^{(m)}+e_r^{(m)} = q_r^{(m+1)}+e_{r-1}^{(m+1)} \quad q_{r+1}^{(m)}e_r^{(m)} = q_r^{(m+1)}e_r^{(m+1)}$$

to the initial values

$$e_0^{(m)} = 0, \; q_1^{(m)} = c_{m+1}/c_m \qquad (m = 0,1,\ldots).$$

The $q-d$ algorithm is also a lozenge algorithm. Use may thus be made of a one-dimensional real array l_s in the evaluation of the sequence $q_r^{(0)}$, $e_r^{(0)}$ $(r=0,1,\ldots)$.

Now in the case of certain series (Whittaker functions, the generalized hypergeometric series, etc.) it so occurs that the ratio $q_1^{(m)}$ is easier to express than the general term c_m. For such series we have the following

```
procedure q1realSfraction(result, c0, q1m, argument, relerror, available
storage, storage not exceeded);
value available storage; integer available storage;
real result, c0, q1m, argument, relerror; boolean storage not exceeded;
begin array l[0:available storage];
real procedure qdalgorithm(m1); value m1; integer m1;
begin integer s1; real aux0, aux1, aux2;
if m1 > available storage then
begin storage not exceeded := false; go to END end;
m := m1; aux1 := q1m; for s1 := 0 step 1 until m1-1 do begin
if even(s1) then
aux0 := aux1-l[s1]+(if s1 > 0 then l[s1-1] else 0.0) else
aux0 := aux1×l[s1-1]/l[s1]; if s1 > 0 then l[s1-1] := aux2;
aux2 := aux1; aux1 := aux0 end;
if m1 > 0 then l[m1-1] := aux2; qdalgorithm := l[m1] := aux1 end;
qerealSfraction(result, c0, qdalgorithm(2×r-2), qdalgorithm(2×r-1),
argument, relerror); storage not exceeded := true; END: end;
```

As can be seen this procedure makes use of the preceding *qerealSfraction* procedure.

This procedure may be used to evaluate the Weber parabolic cylinder function $S_1(a; z)$. This has the asymptotic expansion

$$S_1(a; z) \sim e^{-z^2/4} z^{-a+3/2} \left\{ \frac{1}{z^2} - \frac{(a+\frac{1}{2})(a+\mathrm{a})}{2z^4} + \frac{(a+\frac{1}{2})(a+\frac{3}{2})(a+\frac{5}{2})(a+\frac{7}{2})}{2.4.z^6} - \ldots \right\}.$$

Firstly we apply the $q-d$-algorithm to the series in braces in order to convert it into and evaluate it as an S-fraction, and then we multiply the result by $e^{-z^2/4} z^{-a+3/2}$ to obtain $S_1(a; z)$.

```
begin integer i, m, r, very end; real a, eps;
array z, zsquared, aux1, aux2[0:1]; boolean successful;
comment This comment must be replaced by the procedures eq, seqeq,
cm, cd, real, mod, arg, compsq, compln, compexp, onehochother, compread,
compprint, druck, even, qerealSfraction and q1realSfraction;
a := read; compread(z[i]); eps := read; very end := read;
NLCR; print(a); compprint(z[i]); print(eps); compsq(zsquared[i], z[i]);
```

```
compexp(aux1[i], -zsquared[i]/4.0);
onehochother(aux2[i], z[i], real (-a+1.5));
cm(aux2[i], aux1[i], aux2[i]);
q1realSfraction(aux1[i], 1.0, -(a+2×m+0.5)×(a+2×m+1.5)/
                              (2×(m+1)), zsquared[i],eps, very end, successful);
if successful then
begin cm(aux[i], aux1[i], aux2[i]); NLCR; druck(aux1[i]) end end
```

Finally we give a procedure which accepts the coefficients c_m themselves:

```
procedure crealSfraction(result, c, argument, relerror, available storage,
storage not exceeded);
integer available storage; real result, c, argument, relerror;
boolean storage not exceeded; begin real cm, cmplus1;
real procedure q1m; begin m := m+1;
cmplus 1 := c; q1m := cmplus1/cm; cm := cmplus1 end;
m := 0; cm := c; q1realSfraction(result, cm, q1m, argument, relerror, avail-
able storage, storage not exceeded) end;
```

This procedure in turn makes use of the preceding procedures *qerealSfraction* and *q1realSfraction*.

This procedure may be used to evaluate the integral of Goodwin and Staton [20]

$$\int_0^\infty \frac{e^{-t^2}}{z+t}\,dt \sim -\frac{1}{2}\sum_{s=0}^{\infty} \Gamma\left(\frac{s+1}{2}\right)(-z)^{-s-1}$$

for which

$$c_s = \tfrac{1}{2}\Gamma\left(\frac{s+1}{2}\right)(-1)^s \qquad (s = 0,\ 1, \ldots)\,.$$

For this we need the

```
real procedure Goodwin Staton coefficient;
begin own real cmminus2, cmminus1, cm;
if m = 0 then
Goodwin Staton coefficient := cmminus2 := 0.88622 69254 52758
else if m = 1 then
Goodwin Staton coefficient := cmminus1 := -0.5 else
begin cm := cmminus2×(m-1.0)/2.0; cmminus2 := cmminus1;
cmminus1 := Goodwin Staton coefficient := cm end end;
```

The final programme is as follows:

begin integer *i, r, m, very end*; **real** *eps*; **boolean** *successful*;
array *z, final answer*[0:1];
comment This comment must be replaced by the procedures *eq, seqeq, cm, cd, real, mod, compread, compprint, druck, even, qerealSfraction, q1realSfraction, crealSfraction* and *Goodwin Staton coefficient*;
compread(*z*[*i*]); *eps* := *read*; *very end* := *read*;
NLCR; *compprint*(*z*[*i*]); *print*(*eps*);
crealSfraction(*final answer*[*i*], *Goodwin Staton coefficient, z*[*i*], *eps, very end, successful*); **if** *successful* **then** *druck*(*final answer*[*i*]) **end**

J-fractions.

The even part

$$\frac{c_m}{z-q_1^{(m)}-}\ \frac{e_1^{(m)}q_1^{(m)}}{z-q_2^{(m)}-e_1^{(m)}-}\ \cdots\ \frac{e_r^{(m)}q_r^{(m)}}{z-q_{r+1}^{(m)}-e_r^{(m)}-}\ \cdots$$

of an S-fraction is called a J-fraction. It might be thought that the number of arithmetic operations may be reduced by using J-fraction as opposed to S-faction expansions. This is true. But for positive real z the convergents of a convergent S-fraction form an oscillating sequence, and the method of estimating the relative errors of the successive convergents used in the foregoing procedures may therefore be justified for positive real z, and numerical experiments indicate that this method of testing is vindicated for general values of z. In the case of J-fraction expansions, however, numerical experience indicates that this method of estimation sometimes produces misleading results, and moreover is misleading in an unpredictable fashion. Such continued fractions are not therefore considered in this note.

Corresponding Continued Fractions [21].

If the power series $\beta(x)\sim\sum_{s=0}^{\infty}c_sx^s$ is given, it is perhaps most efficient to write $z=x^{-1}$ and $\beta(x)=z\{\sum_{s=0}^{\infty}c_sz^{-s-1}\}$. The latter series may be transformed into an S-fraction as before and evaluated as such. Slightly more efficient would be the direct use of the relationship

$$\sum_{s=0}^{\infty}c_sz^{-s}\sim\frac{c_0}{1-}\ \frac{q_1^{(0)}}{z-}\ \frac{e_1^{(0)}}{1-}\ \cdots\ \frac{q_r^{(0)}}{z-}\ \frac{e_r^{(0)}}{1-}\ \cdots$$

If we write $x=z^{-1}$ in this relationship, we have, after an equivalence transformation, the continued fraction corresponding [21] to the power series $\sum_{s=0}^{\infty}c_sx^s$:

$$\frac{c_0}{1-}\ \frac{q_1^{(0)}x}{1-}\ \frac{e_1^{(0)}x}{1-}\ \cdots\ \frac{q_r^{(0)}x}{1-}\ \frac{e_r^{(0)}x}{1-}\ \cdots$$

Again it transpires that the corresponding continued fraction may well converge in domains of the x-plane in which the power series diverges. We quote the following theorem [22]: If the sequence $q_r^{(0)}$, $e_r^{(0)}$, $(r=0,1,\ldots)$ converges to a non-zero constant c, then the corresponding continued fraction converges for any finite value of x which does not lie on that segment of the line $\arg(x)=-\arg(c)$ which joins the point $(-\frac{1}{4}c)$ to the point at infinity and does not pass through the origin.

The reason for preferring the continued fraction expansions in z to those in x is that in the evaluation of the twin recursions for $A_{j,s}$ $(j=0,1)$ the S-faction involves complex multiplication only in the evaluation of $A_{j,2s+1}$ $(s=0,1,\ldots)$ whereas the corresponding expansion involves complex multiplication in the evaluation of each $A_{j,s}(s=0,1,\ldots)$.

However, in certain cases, if we write the corresponding expansion in the form

$$\frac{c_0}{b_1-}\ \frac{q_1^{(0)\prime}x}{b_2-}\ \frac{e_1^{(0)\prime}x}{b_3-}\ \cdots\ \frac{q_r^{(0)\prime}x}{b_{2r}-}\ \frac{e_r^{(0)\prime}x}{b_{2r+1}-}\ \cdots$$

it transpires that the coefficients c_0, b_s $(s=1,2,\ldots)$, $q_r^{(0)\prime}$, $e_r^{(0)\prime}$ $(r=1,2,\ldots)$ may be given exactly, and this is a favorable point when considering the propagation of error in the evaluation of the successive convergents. For the evaluation of such continued fractions, we give the following

```
procedure corresponding continued fraction(result, c0, qr, er, bs, argument,
                                                              relerror);
value relerror; real result, c0, qr, er, bs, argument, relerror;
begin integer j, S, Sdash; real coefft, partdenom; boolean q;
array x, aux[0:1], A[0:2, 0:1, 0:1];
eq(x[i], argument); eq(A[0, 0, i], real(c0));
s := r := Sdash := 1; eq(A[1, 0, i], real(bs));
eq(A[1, 1, i], real(1.0)); seqeq(A[0, 1, i], A[2, 1, i], 0.0);
S := 0; q := false;
go to CONVERGENT;
RECURSION: Sdash := S; S := 1-Sdash; q := ¬q; partdenom := bs;
coefft := -(if q then qr else er); for j := 0, 1 do begin
cm(aux[i], x[i], coefft×A[j, S, i]);
eq(A[j, S, i], aux[i]+partdenom×A[j, Sdash, i] end;
if ¬q then r := r+1;
```

CONVERGENT: *cd*(*A*[2, *S*, *i*], *A*[0, *S*, *i*], *A*[1, *S*, *i*]);
cd(*aux*[*i*], *A*[2, *Sdash*, *i*], *A*[2, *S*, *i*]);
s := *s*+1; **if** abs(1.0−*mod*(*aux*[*i*])) > *relerror* **then go to** *RECURSION* **else** *eq*(*result*, *A*[2, *S*, *i*]) **end**;

It uses nonlocal integers r and s indicating the suffices of the partial numerators and denominators respectively.

This procedure may be used to evaluate the incomplete beta function $B_x(p,q)$ given by

$$B_x(p,q) = \int_0^x t^{p-1}(1-t)^{q-1}dt = p^{-1}x^p(1-x)^q {}_2F_1(1,p+q;\,p+1;\,x)$$

$$= x^p(1-x)^q \left\{ \frac{1}{p-} \frac{p(p+q)x}{p+1-} \frac{1(1-q)x}{p+2-} \frac{(p+1)(p+q+1)x}{p+3-} \right.$$

$$\left. \cdots \frac{(p+r-1)(p+q+r-1)x}{p+2r-1-} \frac{r(r-q)x}{p+2r-} \cdots \right\}$$

A complete programme for doing this is as follows:

begin integer *i*, *r*, *s*; **real** *eps*, *p*, *q*; **array** *x*, *aux*1, *aux*2[0:1];
comment This comment must be replaced by the procedure *eq*, *seqeq*, *cm*, *cd*, *real*, *mod*, *arg*, *compln*, *compexp*, *onehochother*, *compread*, *compprint*, *druck* and *corresponding continued fraction*;
p := *read*; *q* := *read*; *compread*(*x*[*i*]); *eps* := *read*;
NLCR; *print*(*p*); *print*(*q*); *compprint*(*x*[*i*]); *print*(*eps*);
onehochother(*aux*1[*i*], *x*[*i*], *real*(*p*));
onehochother(*aux*2[*i*], *real*(1.0)−*x*[*i*], *real*(*q*));
cm(*aux*2[*i*], *aux*1[*i*], *aux*2[*i*]); *corresponding continued fraction*
(*aux*1[*i*], 1.0, (*p*+*r*−1)×(*p*+*q*+*r*−1), *r*×(*r*−*q*), *p*+*s*−1, *x*[*i*], *eps*);
cm(*aux*1[*i*], *aux*1[*i*], *aux*2[*i*]); *NLCR*; *druck*(*aux*1[*i*]) **end**

This programme may be checked by use of the procedures 15), 20), and 22), and use of the relationships

$$\ln(1+x) = -B_{-x}(1,0)\,,$$

$$\arctan(x) = -\frac{i}{2}B_{-x^2}(\tfrac{1}{2},0)\,,$$

$$\arcsin(x) = \tfrac{1}{2}B_{x^2}(\tfrac{1}{2},\tfrac{1}{2})\,.$$

The same procedure may be used to evaluate the incomplete γ-function of small argument

$$\gamma(c;x) = \int_0^x e^{-t}t^{c-1}dt$$

$$= x^c e^{-x}\left\{\frac{1}{c-}\,\frac{cx}{c+1+}\,\frac{1x}{c+2-}\,\frac{(c+1)x}{c+3+}\cdots\frac{(c+r-1)x}{c+2r-1+}\,\frac{rx}{c+2r-}\cdots\right\}$$

when $c \neq 0, -1, -2, \ldots$ A programme which does this is as follows:

comment This comment must be replaced by the procedures *eq*, *seqeq*, *cm*, *cd*, *real*, *mod*, *arg*, *compln*, *compexp*, *onehochother*, *compread*, *compprint*, *druck* and *corresponding continued fraction*;
c := *read*; *compread*(*x*[*i*]); *eps* := *read*;
NLCR; *print*(*c*); *compprint*(*x*[*i*]); *print*(*eps*);
corresponding continued fraction(*aux*1[*i*], 1.0, *c*+*r*−1, −*r*, *c*+*s*−1, *x*[*i*], *eps*);
onehochother(*aux*2[*i*], *x*[*i*], *real*(*c*)); *cm*(*aux*1[*i*], *aux*2[*i*], *aux*1[*i*]);
compexp(*aux*2[*i*], −*x*[*i*]); *cm*(*aux*1[*i*], *aux*1[*i*], *aux*2[*i*]);
NLCR; *druck*(*aux*1[*i*]) **end**

Finally, for completeness, we remark that the continued fraction

$$\frac{J_{m+1}(z)}{J_m(z)} = \frac{z/2}{m+1-}\,\frac{(z/2)^2}{m+2-}\,\frac{(z/2)^3}{m+3-}\cdots$$

is also subsumed within those of this section.

A General Continued Fraction Procedure.

For the evaluation of continued fractions having the more general form

$$C = b_0 + \frac{a_1}{b_1+}\,\frac{a_2}{b_2+}\cdots\frac{a_s}{b_s+}\cdots$$

we give the following

procedure *continued fraction*(*result*, *b*0, *as*, *bs*, *relerror*); **value** *relerror*;
real *result*, *relerror*; **procedure** *b*0, *as*, *bs*;
begin integer *S*, *Sdash*; **array** *aux*1, *aux*2[0:1], *A*[0:2, 0:1, 0:1];
*b*0; *seqeq*(*A*[0, 1, *i*], *A*[2, 1, *i*], *Bs*[*i*]); *seqeq*(*A*[0, 0, *i*], *A*[1, 1, *i*], *real*(1.0));
eq(*A*[1, 0, *i*], 0.0); *s* := *S* := 1;
RECURSION as; *bs*; *Sdash* := *S*; *S* := 1−*Sdash*;
for *j* := 0, 1 **do begin** *cm*(*aux*1[*i*], *Bs*[*i*], *A*[*j*, *Sdash*, *i*]);
cm(*aux*2[*i*], *As*[*i*], *A*[*j*, *S*, *i*]); *eq*(*A*[*j*, *S*, *i*], *aux*1[*i*]+*aux*2[*i*]) **end**;
cd(*A*[2, *S*, *i*], *A*[0, *S*, *i*], *A*[1, *S*, *i*]); *cd*(*aux*1[*i*], *A*[2, *Sdash*, *i*], *A*[2, *S*, *i*]);
s := *s*+1; **if** abs(1.0−*mod*(*aux*1[*i*])) > *relerror* **then go to** *RECURSION*
else *eq*(*result*, *A*[2, *S*, *i*]) **end**;

It uses a non-local integer s indicating the suffix of the successive coefficients and two non-local arrays (As, $Bs[0:1]$) for the coefficients.

By means of this procedure we may evaluate the continued fraction

$$\cfrac{1}{1+}\ \cfrac{x}{1+}\ \cfrac{x^2}{1+}\ \cfrac{x^3}{1+}\ \cdots$$

due to Ramanujan [23]. This is not perhaps the most frequently encountered function of Mathematical Physics, but it will serve as an example.

First we have the three coefficient procedures:

```
procedure b0Ramanujan; eq(Bs[i], 0.0);
procedure asRamanujan(argument); real argument;
begin own real array x[0:1];
if s = 1 then begin eq(x[i], argument); eq(As[i], real(1.0)) end else
cm(As[i], x[i], As[i]) end;
procedure bsRamanujan; eq(Bs[i], real(1.0));
```

Finally, the complete programme:

```
begin integer i, s; real eps; array x, result, As, Bs[0:1];
comment This comment must be replaced by the procedures eq, seqeq,
cm, cd, real, mod, compread, compprint, druck, b0Ramanujan, asRamanu-
jan, bsRamanujan and continued fraction;
compread(x[i]); eps := read; NLCR; compprint(x[i]); print(eps);
continued fraction(result[i], b0Ramanujan, asRamanujan(x[i]),
                   bsRamanujan, eps); NLCR; druck(result[i]) end
```

Conclusion.

Most of the procedures described in this note relate to continued fractions. Numerical data relating to the convergence benaviour of real continued fractions may be found in [24].

A great deal of the time spent in running the given programmes is devoted to estimating the relative error of successive convergents. In the final version of a programme to compute a given function of z to a prescribed relative accuracy ε in the modulus, a simple numerical efficiency function $n(\mathrm{Re}(z), \mathrm{Im}(z), \varepsilon)$ which gives the order of the convergent which provides the required degree of approximation to the given function, should be constructed. An optimal solution to this problem is described in [25].

The computation of functions of a complex variable is a somewhat circumstantial matter, requiring the deployment of a number of special

methods. The above ALGOL programmes may not therefore be regarded as anything more than a first step in the construction of a library of ALGOL procedures for the evaluation of functions in the complex plane.

A number of special functions have been selected to illustrate the use of the procedures given; many more may be taken from [26].

Acknowledgement.

All the programmes of this note have been tested (and the numerical results produced thereby compared where possible with published tables) on an ALGOL compiler constructed for the $X1$ computer by E. W. Dijkstra and J. R. Zonneveld.

REFERENCES

1. Backus, J. W. et al., *Report on the Algorithmic Language* ALGOL 60, Num. Math., vol. 2, 1960, p. 106.
2. Fröberg, C. E., *Rational Chebyshev Approximation of Elementary Functions*, BIT, vol. 1, 1961, p. 256.
3. Wynn, P., *On a Device for Computing the $e_m(S_n)$ Transformation*, M.T.A.C., vo. 52, 1956, p. 663.
4. Szegö, G., *Über orthogonale Polynome, die zu einer gegebener Kurve der komplexen Ebene gehören*, Math. Z., vol. 9, 1921, 0. 218.
5. Todd, J. and Warschawski, S. E., *On the Solution of the Lichtenstein–Gershgorin Integral Equation in Conformal Mapping: II Computational Experiments*, N.B.S., Appl. Math. Ser. 42, p. 31.
6. Wynn, P., *The Rational Approximation of Functions which are Formally Defined by a Power Series Expansion*, Maths. of Comp., vo. 14, 1960, p. 147.
7. Chebyshev, P., *Sur les Fractions Continues*, Journ. de Math., vol. 8, 1858, p. 289.
8. Stieltjes, T. J., *Recherches sur les Fractions Continues*, Annales de la Faculté des Sciences de Toulose, (1), 8, 1894, T1–122, (1), 9, 1895, A5–57.
9. Markoff, A., *On Certain Applications of Algebraic Continued Fractions*, Thesis, St. Petersburg, 1884.
10. Markoff, A., *Proof of the Convergence of Many Continued Fractions*, Trans. Roy. Acad. Sci., St. Petersburg, 1893 (supp.).
11. Markoff, A., *On Functions Generated by Developing Power Series in Continued Fractions*, Trans. Roy. Acad. Sci., St. Petersburg, 1894 (supp.).
12. Markoff, A., *Note sur les Fractions Continues*, Bulletin de la Classe Physico-Mathématique de l'Académie Impériale des Sciences de Saint-Petersburg, vol. 5, 1895, p. 9.
13. Markoff, A., *Deux Demonstrations de la Convergence de Certaines Fractions Continues*, Acta Mathematica, vol. 19, 1895, p. 93.
14. Markoff, A., *Nouvelles Applications des Fractions Continues*, Memoires de l'Académie des Sciences de St. Petersburg, Classe Physico-Mathématique, vol. 3, 1896.
15. Markoff, A., *Nouvelles Applications des Fractions Continues*, Math. Annalen, vol. 47, 1896, p. 579.
16. Shohat, J. A. and Tamarkin, J. D., *The Problem of Moments*, Mathematical Surveys 1, Amer. Math. Soc. 1943.

17. Nevanlinna, R., *Asymptotische Entwicklungen beschränkter Funktionen und das Stieltjes Momenten Problem*, Annales Academiae Fenniae, (A), vol. 18, 1922.
18. Carleman, T., *Les Fractions Quasi-analytiques*, Gauthier–Villars, Paris 1926.
19. Rutishauser, H., *Der Quotienten-Differenzen Algorithmus*, Birkhauser Verlag, Basel, 1957.
20. Goodwin, E. T. and Staton, J., *Table of* $\int_0^\infty e^{-u^2}du/(u+x)$, Quart. Journ. Mech. and Appl. Math., vol. 1, 1948, p. 319.
21. Perron, O., *Die Lehre von den Kettenbrüchen, vol. II*, Teubner, Stuttgart, 1957.
22. van Vleck, E. V., *On the Convergence of Algebraic Continued Fractions whose Coefficients have Limiting Values*, Trans. Amer. Math. Soc., vol. 5, 1904, p. 253.
23. Ramanujan, S., *Collected Papers*, Cambridge 1927.
24. Wynn, P., *The numerical Efficiency of Certain Continued Fraction Expansions*, Proc. Kon. Ned. Akad. Wetensch. Amsterdam, vol. 65, ser. A, 1962, p. 127.
25. Wynn, P., *Numerical Efficiency Profile Functions*, Proc. Kon. Ned. Akad. Wetensch. Amsterdam, vol. 65, ser. A, 1962, p. 118.
26. Erdélyi, A. et al., *Higher Transcendental Functions*, McGraw–Hill.
27. Clenshaw, C. W., *Chebyshev Series for Mathematical Functions, Mathematical Tables, vol. 5*, H.M.S.O., London 1962.

MATHEMATISCH CENTRUM
AMSTERDAM