Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, 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), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

MATHEMATICAL CENTRE TRACTS 18

R.P. VAN DE RIET

FORMULA MANIPULATION IN ALGOL 60

PART 2

MATHEMATISCH CENTRUM AMSTERDAM 1975

AMS (MOS) subject classification scheme (1970): 68A15

ACM -Computer Review- category: 5.7

ISBN 90 6196 038 X

1st printing 1968
2nd printing 1975

TABLE OF CONTENTS OF PART 2

Chapter 4

COMPLEX ARITHMETIC IN ALGOL 60

This chapter contains a set of ALGOL 60 procedures by means of which ALGOL 60 programs may be written for computations with complex numbers. Besides procedures for the elementary operations, procedures are given for the elementary functions: exp, sin, cos, tan, ln, sqrt, arctan, arcsin and arccos.
The result of the last five multivalued functions is not necessarily the ordinary principal value as the user may specify his own principal value.

4.1. Introduction

In designing a system of ALGOL 60 procedures for formula manipulation as described in chapter 3, it turned out to be desirable to have a set of procedures for complex arithmetic in ALGOL 60 since formula manipulation is used and will be used in the future for generating programs which perform ordinary as well as complex arithmetic. This is one reason for including this chapter.
It will turn out that the method of constructing complex-arithmetic ALGOL 60 programs is very similar to the method of constructing formula-manipulation ALGOL 60 programs. In fact, complex arithmetic may be considered as a special form of formula manipulation; this is the other reason for including this chapter.

A similar but less elegant method has been described in [19].

4.2.1. The method

ALGOL 60 does not admit expressions of complex arithmetic type. In order to overcome this difficulty one may split the expressions into real and imaginary parts, which leads to efficient use of computer time (but not of programmer's time), or the expressions may be written in polish prefix form which will be done in this chapter.
In the latter case the symbols +, ×, ... are replaced by the operators *S*, *P*, ...
Thus:

$$a + b + c \text{ is written as } S(a, S(b, c))$$

$$\text{and } a \times b + c \text{ is written as } S(P(a, b), c).$$

The form of the ALGOL 60 procedures *S* and *P* will now be investigated. Schoenfeld [23] proposes procedures of real type. The real part of the calculated sum or product is assigned to the procedure identifier *S* or *P*, respectively; whereas the imaginary part is assigned to a global variable *imp*.
The procedure *S* has the form:

```
real procedure S(a,b); real a, b;
begin real ra, rb, ia, ib;
      ra:= a; ia:= imp;
      rb:= b; ib:= imp;
      S:= ra + rb; imp:= ia + ib
end S;
```

Schoenfeld introduces, moreover, the procedure *T* (of *TAKE*):

```
real procedure T(a); array a;
begin T:= a[0]; imp:= a[1] end T;
```

and by means of these procedures, the calculation "d:= a + b + c", where a, b and c are complex numbers, whose values are given by the arrays *a*, *b* and *c*, can be programmed as:

```
    d[0]:= S(T(a),S(T(b),T(c))); d[1]:= imp;
```

A draw-back of this approach is the frequent and unelegant appearance of *T*.
Another possibility, which does not have this draw-back and which is chosen for this chapter, is to use a stack, in which the complex numbers are stored. Let this stack be formed by the arrays *R* and *I*, declared by *array R,I[0 : bound]*, where *bound* is some large enough integer. Let the stack pointer be *p*.
The actual values of the parameters of *S* and *P* are indexes, defining the locations in the stack, where the values of the corresponding complex numbers are stored.
The procedure identifiers themselves are assigned to the indexes of the locations in the stack where the calculated sum or product is stored.

The procedure *S* then has the form:

```
integer procedure S(a,b); integer a, b;
begin integer u, v; u:= a; p:= p + 1; v:= b; p:= p - 1;
      R[p]:= R[u] + R[v]; I[p]:= I[u] + I[v]; S:= p
end S;
```

If the values of the complex variables a, b and c are given by *R[a]* + *I[a]* × i, *R[b]* + *I[b]* × i and *R[c]* + *I[c]* × i, respectively, and if the value of d = a + b + c should be stored into *R[d]* and *I[d]*, then the following statements

```
      S(a,S(b,c)); R[d]:= R[p]; I[d]:= I[p];
```

have the desired result.
Introduction of

```
integer procedure ASSIGN(z,a); value z, a; integer z, a;
begin ASSIGN:= z; R[z]:= R[a]; I[z]:= I[a] end
```

leads to the following, more elegant, "complex assignment statement":

```
      ASSIGN(d,S(a,S(b,c)))
```

which has the same result. The statement *ASSIGN:= z*, within the procedure body of *ASSIGN*, serves to make possible repeated assignment; e.g.

ASSIGN(e, ASSIGN(d, S(a, S(b,c)))). In this approach it is necessary that for each complex number an integral variable is declared, which should get a value defining the index of the locations in the stack reserved for the storage of that complex number. In order to program this conveniently, a second stack, declared as <u>*integer*</u> <u>*array*</u> *POINTER*[*1:50*], with pointer *kp* is introduced.
On block entry, the value of kp has to be increased by 1; the current value of *p* has to be stored in *POINTER*[*kp*] and space in *R* and *I* has to be reserved for the complex variables to be used.
Using the procedure *DE*, declared by:

```
Boolean procedure DE(z,first); integer z; Boolean first;
begin if first then again kp:= kp + 1; POINTER[kp]:= p; p:= p + 1; end;
      z:= p; p:= p + 1; DE:= false
end DE;
```

it is easy to program the desired operations:
suppose *a*, *b*, *c* and *d* have to be used as the complex numbers a, b, c and d, then the following "block head" is possible:

```
begin integer a, b, c, d; DE(a,DE(b,DE(c,DE(d,true))));
```

After this "block head" the statements defining the computations may be written.

On block exit the effect of *DE* should be cancelled; this may be performed by a call of *ERASE*, which is declared as:

```
procedure ERASE; begin p:= POINTER[kp]; kp:= kp - 1 end;
```

A disadvantage of the above approach is that the computations are rather time consuming due to the use of array elements.

4.2.2. Directions for use

A program using the system of procedures, described in the following section, should have the following structure:

```
begin <declaration of the variables and procedures as given in
      section 4.3>;

      INITIALIZE;

      <the declarations and statements defining the desired complex
      arithmetic computations>
end
```

The data on input tape should start with a number defining the upper bound of the arrays *R* and *I*.

All complex variables should be declared either as simple variables or as subscripted variables of type integer. If only simple variables occur, then the declaration should be followed by a declaration statement defined by:

```
<declaration statement>::= DE(<variable>, true) |
              DE(<variable>, <declaration statement>)
```

Examples: *DE(a, true)*

DE(a, DE(b, DE(c, true)))

If subscripted variables also occur, then the declaration should be followed, first by a declaration statement and second by a for statement in which *DE* is used with second parameter *false*.

Example: the variables a, *v*[*1*], ..., *v*[*100*] are declared by:

```
     integer i, a; integer array v[1 : 100]; DE(a, true);
     for i:= 1 step 1 until 100 do DE(v[i], false);
```

A calculation can be performed through the use of a complex assignment statement, a modulus calculation statement, or an argument calculation statement.

Using the abbreviation "cae" for complex-arithmetic expression we have the following definitions:

```
<complex assignment statement>::= ASSIGN(<variable>, <cae>) |
              ASSIGN(<variable>, <complex assignment statement>)
```

```
<modulus calculation statement>::= mod(<cae>)
<argument calculation statement>::= arg(<cae>)
<cae>::= <variable>|0|1|iu|<complex number>|<real number>|<sum>|
     <difference>|<product>|<quotient>|<power>|<integral power>|
     <sum with real lhs>|<difference with real lhs>|<product with real lhs>|
     <quotient with real lhs>|<elementary-function designator>
<complex-arithmetic expression>::= <cae>
<complex number>::= CN(<real part>,<imaginary part>)
<real part>::= <arithmetic expression>
<imaginary part>::= <arithmetic expression>
<real number>::= RN(<arithmetic expression>)
<sum>::= S(<cae>,<cae>)
<difference>::= D(<cae>,<cae>)
<product>::= P(<cae>,<cae>)
<quotient>::= Q(<cae>,<cae>)
<power>::= POWER(<cae>,<cae>)
<integral power>::= INT POW(<cae>,<exponent>)
<exponent>::= <arithmetic expression>
<sum with real lhs>::= SR(<arithmetic expression>,<cae>)
<difference with real lhs>::= DR(<arithmetic expression>,<cae>)
<product with real lhs>::= PR(<arithmetic expression>,<cae>)
<quotient with real lhs>::= QR(<arithmetic expression>,<cae>)
<elementary-function designator>::= <el func id>(<cae>)
<el func id>::= EXP|SIN|COS|TAN|LN|ARCSIN|ARCCOS|ARCTAN|SQRT
```

Remark: Complex arithmetic expressions occur in the ALGOL 60 program as particular expressions of type integer.

An example of a complex assignment statement is

ASSIGN(a,P(CN(0,-.5),LN(Q(SR(1,P(iu,z)),DR(1,P(iu,z))))))

which corresponds to $a := -i/2 \times \ln((1+iz)/(1-iz))$.

The variable *iu* stands for imaginary unit with *R*[*iu*] = 0 and *I*[*iu*] = 1. Moreover, *R*[*0*] = 0, *I*[*0*] = 0, *R*[*1*] = 1 and *I*[*1*] = 0, so that the complex-arithmetic expressions *0* and *1* refer to the complex numbers 0+0i and 1+0i.

Every block in which complex variables are declared should end with a procedure statement *ERASE*.

In using the elementary multivalued-functions one may specify which value should be calculated by means of the non-local real variable *LOWER BOUND ON ARGUMENT*. This variable should be used in connection with the Boolean variable *SPECIAL ARGUMENT*.
If *SPECIAL ARGUMENT* has the value *false*, then the principal values are calculated; if not, the values are calculated according to the wishes of the user specified by *LOWER BOUND ON ARGUMENT*.
More information is given in subsection 4.3.2.
Typical examples, of how complex-arithmetical computations may be programmed, are the procedure body of *ARCTAN* (section 4.3.2.3) and the program of section 4.4.

4.3. The system of complex arithmetic procedures

This section contains the procedure declarations.
The real variables *pi* and *null* are used with the values 3.14159265359 and $_{10}-600$,respectively.

4.3.1. The elementary operations and singlevalued functions

```
begin
  comment CALCULATION PROGRAM FOR COMPLEX ARITHMETIC
    R 1050 RPR 221266/06;
  real pi,null,LOWER BOUND ON ARGUMENT;
  integer p,kp,iu;
  real array R,I[0:READ]; integer array POINTER[1:50];
  comment The MC standard function READ reads a number from input tape;
  Boolean SPECIAL ARGUMENT;
  Boolean procedure DE(z,first); integer z; Boolean first;
  comment DE should be used in declaring complex variables,
    for example a and b are declared by: DE(a,DE(b,true));
  begin if first then begin kp:= kp + 1; POINTER[kp]:= p; p:= p + 1 end;
    z:= p; p:= p + 1; DE:= false
  end DE;
  procedure ERASE; comment ERASE should occur just before block exit;
  begin p:= POINTER[kp]; kp:= kp - 1 end;
  integer procedure ASSIGN(a,z); value a,z; integer a,z;
  comment ASSIGN is used in an assignment statement a:= z;
  begin ASSIGN:= z; R[a]:= R[z]; I[a]:= I[z] end ASSIGN;
  integer procedure RN(x); value x; real x;
  comment RN(x) is the real number x;
  begin RN:= p; R[p]:= x; I[p]:= 0 end RN;
  integer procedure CN(x,y); value x,y; real x,y;
  comment CN(x,y) is the complex number x + i y;
  begin CN:= p; R[p]:= x; I[p]:= y end CN;
```

```
integer procedure S(u,v); integer u,v; comment S:= u + v;
begin integer a,b; a:= u; p:= p + 1; b:= v; S:= p:= p - 1;
   R[p]:= R[a] + R[b]; I[p]:= I[a] + I[b]
end S;
integer procedure D(u,v); integer u,v; comment D:= u - v;
begin integer a,b; a:= u; p:= p + 1; b:= v; D:= p:= p - 1;
   R[p]:= R[a] - R[b]; I[p]:= I[a] - I[b]
end D;
integer procedure P(u,v); integer u,v; comment P:= u × v;
begin integer a,b; real ra,rb,ia,ib; a:= u; p:= p + 1; b:= v;
   P:= p:= p - 1; ra:= R[a]; rb:= R[b]; ia:= I[a]; ib:= I[b];
   R[p]:= ra × rb - ia × ib; I[p]:= ra × ib + ia × rb
end P;
integer procedure Q(u,v); integer u,v; comment Q:= u/v;
begin integer a,b; real ra,rb,ia,ib,r,den; a:= u; p:= p + 1; b:= v;
   Q:= p:= p - 1; ra:= R[a]; rb:= R[b]; ia:= I[a]; ib:= I[b];
   comment The following statement tends to avoid over- or
      underflow (see R.L.Smith [24]).;
   if abs(rb) ≥ abs(ib) then
   begin r:= ib/rb; den:= rb + r × ib; R[p]:= (ra + ia × r)/den;
      I[p]:= (ia - ra × r)/den
   end else
   begin r:= rb/ib; den:= ib + r × rb; R[p]:= (ra × r + ia)/den;
      I[p]:= (ia × r - ra)/den
   end
end Q;
integer procedure POWER(u,v); integer u,v; comment POWER:= u ⋀ v;
POWER:= EXP(P(LN(u),v));
integer procedure SR(r,z); value r,z; real r; integer z;
comment SR:= r + z, where r is real;
begin SR:= p; R[p]:= r + R[z]; I[p]:= I[z] end SR;
integer procedure DR(r,z); value r,z; real r; integer z;
comment DR:= r - z, where r is real;
begin DR:= p; R[p]:= r - R[z]; I[p]:= - I[z] end DR;
```

```
integer procedure PR(r,z); value r,z; real r; integer z;
comment PR:= r × z, where r is real;
begin PR:= p; R[p]:= r × R[z]; I[p]:= r × I[z] end PR;
integer procedure QR(r,z); real r; integer z;
comment QR:= r / z, where r is real; QR:= Q(RN(r),z);
integer procedure INT POW(z,n); value z,n; integer z,n;
comment INT POW:= z ↑ n;
if abs(I[z]) ≤ null then INT POW:= RN(R[z] ↑ n) else
if n < 0 then INT POW:= Q(1,INT POW(z,- n)) else
if n = 0 then INT POW:= 1 else
if n = 1 then INT POW:= z else
if n = 2 then INT POW:= CN(R[z] × R[z] - I[z] × I[z], 2 × R[z] × I[z]) else
begin integer k; DE(k,true); ASSIGN(k,INT POW(INT POW(z,n ÷ 2),2));
  if (n ÷ 2) × 2 - n ≠ 0 then ASSIGN(k,P(k,z));
  ERASE; ASSIGN(p,k); INT POW:= p
end INT POW;
integer procedure EXP(z); value z; integer z;
begin real r,i; r:= exp(R[z]); i:= I[z];
  EXP:= if abs(i) ≤ null then RN(r) else CN(r × cos(i),r × sin(i))
end EXP;
procedure hyperbolic function(sinh,cosh,y); value y; real sinh,cosh,y;
comment this procedure is due to P.WYNN [27];
begin real y1; y1:= exp(y); cosh:= .5 × (y1 + 1/y1);
  if abs(y) ≥ 1.0 then sinh:= .5 × (y1 - 1/y1) else
  begin integer r; real br,brplus1,brplus2; array CWC[0:5];
    CWC[0]:= 1.13031 82079 8497; CWC[1]:= 4.43368 49848 66$_{10}$-2;
    CWC[2]:= 5.42926 31191$_{10}$-4; CWC[3]:= 3.19843 646$_{10}$-6;
    CWC[4]:= 1.10367 7$_{10}$-8; CWC[5]:= 2.498$_{10}$-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 hyperbolic function;
```

```
integer procedure SIN(z); value z; integer z;
if abs(I[z]) ≤ null then SIN:= RN(sin(R[z])) else
begin real r,s,c; r:= R[z]; hyperbolic function(s,c,I[z]);
   SIN:= CN(sin(r) × c,cos(r) × s)
end SIN;
integer procedure COS(z); value z; integer z;
if abs(I[z]) ≤ null then COS:= RN(cos(R[z])) else
begin real r,s,c; r:= R[z]; hyperbolic function(s,c,I[z]);
   COS:= CN(cos(r) × c,- sin(r) × s)
end COS;
integer procedure TAN(z); value z; integer z;
if abs(I[z]) ≤ null then TAN:= RN(sin(R[z])/cos(R[z])) else
begin real s,c,si,co,r; r:= R[z];
   hyperbolic function(s,c,I[z]); si:= sin(r); co:= cos(r);
   TAN:= Q(CN(si × c,co × s),CN(co × c,-si × s))
end TAN;
```

4.3.2. The multivalued functions

The multivalued functions are defined by:

$LOWER\ BOUND\ ON\ ARGUMENT < arg(z) \le LOWER\ BOUND\ ON\ ARGUMENT + 2 \times pi$,

$LN(z) = \ln|z| + i\ arg(z)$,

$SQRT(z) = EXP(\frac{1}{2}\ LN(z))$,

$ARCSIN(z) = \frac{1}{i}\ LN(iz + SQRT(1 - z^2))$,

$ARCCOS(z) = \frac{1}{i}\ LN(z + i\ SQRT(1 - z^2))$,

$ARCTAN(z) = \frac{1}{2i}\ LN(1 + iz)/(1 - iz))$.

We do not always use these definitions, however, to calculate the functions.
By means of certain transformations, e.g. addition of a multiple of $2 \times pi$, the resulting values are adapted in order to satisfy the above definitions. For this the procedure *mult of 2pi* is used.

Remark: It may occur that little changes of *z* result in completely different values for the multivalued functions. Therefore, one should be careful in choosing *LOWER BOUND ON ARGUMENT*. It may be desirable to choose it somewhat too large (say $_{10}-10$).

```
real procedure mult of 2pi(a); value a; real a;
mult of 2pi:= (entier((LOWER BOUND ON ARGUMENT - a)/
  6.28318530717958) + 1) × 6.28318530717958;

real procedure arg(z); value z; integer z;
begin real r,i,a; r:= R[z]; i:= I[z];
  a:= if abs(r) ≤ null then sign(i) × pi/2 else
      if r > null then arctan(i/r) else
      if abs(i) ≤ null then pi else
      sign(i) × pi/2 - arctan(r/i);
  if SPECIAL ARGUMENT then a:= a + mult of 2pi(a);
  arg:= a
end arg;
```

```
real procedure mod(z); value z; integer z;
comment This procedure tends to avoid over- or underflow
  (see Paul Friedland: Algorithm 312, Absolute value and
    sqare root of a complex number.
    Comm. ACM 10-10-1967, p 665).;
begin real r,i; r:= abs(R[z]); i:= abs(I[z]);
  mod:= if r = 0 ∧ i = 0 then 0 else
    if r ≥ i then r × sqrt(1 + (i/r) ↑ 2) else
                  i × sqrt((r/i) ↑ 2 + 1)
end mod;
```

4.3.2.1. The logarithmic function

The logarithm is,in general,calculated by means of the expression

$$\ln(z) = \ln|z| + i \arg(z).$$

However, when either $z = a + i\varepsilon$ or $z = \varepsilon + ia$, where $|a| = 1$ and $|\varepsilon| < 0.1$, the logarithm is calculated from the expression

$$\ln(1+z) = 2i \arctan \frac{z}{i(z+2)}.$$

The reason for this is that the first expression gives an absolute error which is of the order of the relative error (δ) of ε, whereas the second formula gives a relative error of the order δ. Obviously,no improvement can be obtained from the second formula if $|a|$ is not exactly equal to 1.

```
integer procedure LN(z); value z; integer z;
begin real r,i,n; r:= R[z]; i:= I[z];
  if (abs(r - 1) ≤ null ∧ abs(i) < .1) ∨ (abs(i - 1) ≤ null ∧ abs(r) < .1) then
  begin if abs(i) < .1 then
    begin n:= 4 + i × i; LN:= if sign(r) = +1 then
      P(CN(0,2),ARCTAN(CN(2 × i/n, - i × i/n))) else
      S(CN(0,pi × sign(i + null)),P(CN(0,2),ARCTAN(CN(- 2 × i/n,- i × i/n))))
    end else
    begin n:= 4 + r × r; LN:= if sign(i) = +1 then
      S(CN(0,pi/2),P(CN(0,2),ARCTAN(CN(- 2 × r/n, - r × r/n)))) else
      S(CN(0,-pi/2),P(CN(0,2),ARCTAN(CN(2 × r/n,- r × r /n))))
    end; if SPECIAL ARGUMENT then I[p]:= I[p] + mult of 2pi(I[p])
  end else LN:= CN(.5 × ln(r × r + i × i),arg(z))
end LN;
```

4.3.2.2. The square root function

```
integer procedure SQRT(z); value z; integer z;
begin real a,b,r,i; a:= R[z]; b:= I[z];
  r:= mod(z); if r ≤ null then
  begin r:= 0; i:= 0; goto END end;
  r:= sqrt((abs(a) + r)/2); i:= b:= b/(2 × r);
  if a < 0 then
  begin i:= if b ≥ 0 then r else  - r; r:= abs(b) end;
  if SPECIAL ARGUMENT then
  begin real phi; integer k;
    SPECIAL ARGUMENT:= false;
    phi:= arg(CN(r,i));
    k:= entier((phi - LOWER BOUND ON ARGUMENT/2)/pi);
    if k : 2 × 2 - k ≠ 0 then begin r:= - r; i:= - i end;
    SPECIAL ARGUMENT:= true
  end;
END: SQRT:= CN(r,i)
end SQRT;
```

4.3.2.3. <u>The arctan function</u>

For $|z| >$ the *ARCTAN* is calculated by means of the formula:

$$\arctan(z) = \pi/2 - \arctan(1/z);$$

for $.5 \le |z| \le 2$ it is calculated by means of the formula:

$$\arctan(z) = \frac{1}{2i} \ln((1 + iz)/(1 - iz)).$$

Finally, for $|z| < .5$ the *ARCTAN* is calculated as follows:

Consider two numbers a and b; let the numbers a_n and b_n be defined by:

$$a_0 = a, \qquad b_0 = b,$$

$$a_{n+1} = (a_n + b_n)/2, \quad b_{n+1} = (a_{n+1} \cdot b_n)^{\frac{1}{2}}, \; n = 0, 1, 2, \ldots,$$

where the square root with positive real part is chosen (i.e. *SPECIAL ASSIGNMENT* = <u>*false*</u>).

Introducing $z_n = a_n/b_n$ and $\phi = \arccos(a/b)$, it follows that

$$z_{n+1}^2 = (1 + z_n)/2;$$

hence,

$$z_n = \cos(\phi/2^n).$$

Therefore,

$$b_n = z_n b_{n-1} = \prod_{k=1}^{n} \cos(\phi/2^k) \cdot b$$

and

$$b_n = \frac{2^{-n} \sin \phi}{\sin(\phi/2^n)} \cdot b.$$

Denoting the limit of a_n and b_n by $M(a,b)$, we have

$$M(a,b) = \frac{\sin \phi}{\phi} \cdot b = \frac{(b^2 - a^2)^{\frac{1}{2}}}{\arccos(a/b)} \cdot \qquad (4.2)$$

For $a = 1$ and $b = (1 + z^2)^{\frac{1}{2}}$, we then have

$$\arctan(z) = z/M(1,(1 + z^2)^{\frac{1}{2}}). \qquad (4.3)$$

The process, although very elegant, converges only linearly and e.g. for $z = .5$, 20 iterations are needed to obtain $|a_n - b_n| < 10^{-12}$. It will be shown, however, that if the process is discontinued as soon as $|a_n - b_n| < 10^{-3}$, which occurs for $z = .5$ already for $n = 4$, then $M(a,b)$ can be approximated (without extracting the square root) within the desired accuracy of 10^{-12}.

Let $\delta = a_n - b_n$ and $|\delta| < 10^{-3}$.
We observe that

$$M(a,b) = M(a_{n+1},b_{n+1}) = b_{n+1}M(a_{n+1},1). \qquad (4.4)$$

As well b_{n+1} as $M(a_{n+1}/b_{n+1},1)$ will be approximated.

Instead of calculating b_{n+1} as the geometric mean of a_{n+1} and b_n, we calculate b^*_{n+1} as the arithmetic mean:

$$b^*_{n+1} = (a_{n+1} + b_n)/2. \qquad (4.5)$$

A direct calculation shows that

$$b^*_{n+1} - b_{n+1} = \left(\frac{1 - (1 - \delta^2)^{\frac{1}{2}}}{16\, b^{*2}_{n+1}}\right)$$

$$= \frac{\delta^2}{32b^*_{n+1}} - \frac{\delta^4}{2^{11}b^{*3}_{n+1}}\left(1 - \frac{\theta\delta^2}{16\, b^{*2}_{n+1}}\right)^{-3/2}, \qquad (4.6)$$

where $0 < \theta < 1$.

If we approximate b_{n+1} by

$$\beta = b^*_{n+1} - \frac{\delta^2}{32b^*_{n+1}} \qquad (4.7)$$

then $|\beta - b_{n+1}| < 10^{-12}$, provided $|b^*_{n+1}| \gg 2^{-19/6}$.

It will be shown now that $|b^*_{n+1}| > \frac{1}{2}\sqrt{3}$.

Consider the region $G(\kappa) = \{z : \mathrm{Re}(z) \geq \kappa > 0\}$.

Due to the convexity of $G(\kappa)$, it follows that:

if $a_m \in G(\kappa)$ and $b_m \in G(\kappa)$, then $a_{m+1} \in G(\kappa)$.

Consider next the image $H(\kappa)$ of $G(\kappa)$ in the complex W-plane, with $W = \ln(z)$, where $\ln(z)$ assumes its principal value. Since $H(\kappa)$ is also a convex region, it follows that if $a_{m+1} \in G(\kappa)$ and thus $\ln(a_{m+1}) \in H(\kappa)$ and if $b_m \in G(\kappa)$ and thus $\ln(b_m) \in H(\kappa)$, then $\frac{1}{2}\{\ln(a_{m+1}) + \ln(b_m)\} =$
$= \ln(b_{m+1}) \in H(\kappa)$ and thus $b_{m+1} \in G(\kappa)$.
From $|z| < .5$, it follows that $\mathrm{Re}(b_0) > \frac{1}{2}\sqrt{3}$; therefore, $\mathrm{Re}(a_n) > \frac{1}{2}\sqrt{3}$ and $\mathrm{Re}(b_n) > \frac{1}{2}\sqrt{3}$, from which it follows that $|b_{n+1}| > \frac{1}{2}\sqrt{3}$; we have, moreover, $|b^{*}_{n+1}| > \frac{1}{2}\sqrt{3}$.

In equation (4.4), b_{n+1} is now estimated by β as defined in (4.7); we proceed with estimating $M(a_{n+1}/b_{n+1}, 1)$.

We have

$$M(x,1) = \frac{y}{\arcsin(y)} \quad , \tag{4.8}$$

with $y = (1 - x^2)^{\frac{1}{2}}$.
Expanding $\arcsin(y)$ in a power series, we have

$$\arcsin(y) = y(1 + \frac{1}{6}y^2 + \frac{3}{40}y^4 + \frac{5}{112}y^6 + R\,y^8), \tag{4.9}$$

where $$R = \frac{1}{9!}\arcsin^{(9)}(\nu y), \tag{4.10}$$

and $0 < \nu < 1$.

From $$1 - (a_{n+1}/b_{n+1})^2 = \frac{-\delta}{2b_n} = \xi \tag{4.11}$$

it follows that, within the accuracy desired:

$$\arctan(z) = \frac{z}{\beta}\,(1 + \frac{1}{6}\xi + \frac{3}{40}\xi^2 + \frac{5}{112}\xi^3 + R\,\xi^4). \tag{4.12}$$

It remains to estimate $R\,\xi^4$.

Let

$$\frac{d^k \arcsin(x)}{dx^k} = P_k(x)/(1 - x^2)^{(2k-1)/2},$$

where $P_k(x)$ is a polynomial in x.

From

$$P_1(x) = 1$$

$$P_{k+1}(x) = \frac{dP_k(x)}{dx} \cdot (1 - x^2) + P_k(x) \cdot (2k - 1)x,$$

$P_9(x)$ can be determined by means of the formula program below:
Since the coefficients of $P_9(x)$ are all positive, we obtain the following estimate for $R\ \xi^4$:

$$|R\ \xi^4| = |\frac{1}{9!} P_9(\nu y)(1 - (\nu y)^2)^{-17/2}|\ |\xi^4|$$

$$\leq \frac{1}{9!} P_9(\alpha)(1 - \alpha^2)^{-17/2} \cdot \alpha^8 = E,$$

where $\alpha = (10^{-3}/\sqrt{3})^{\frac{1}{2}}$. (Note that $|y| = \sqrt{|\xi|} < \alpha$).

Besides, $P_9(x)$, the formula program determines E(RROR).
From E $\approx .3455_{10}-14$, it follows that $R\ \xi^4$ may be omitted from (4.12).

```
Formula program RPR 100168/02 derivatives of arcsin(x)
(8192,  100,   0,     0,     0,    10-20,  10-12,  3,     0)
NLCR; PR STRING(Results RPR100168/02);
EXPAND; FIX; P:= 1;

P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x);
OUTPUT R(P2:= P); ER B RET(P); FIX;
P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x × 3);
OUTPUT R(P3:= P); ER B RET(P); FIX;
P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x × 5);
OUTPUT R(P4:= P); ER B RET(P); FIX;
P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x × 7);
OUTPUT R(P5:= P); ER B RET(P); FIX;
P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x × 9);
OUTPUT R(P6:= P); ER B RET(P); FIX;
P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x × 11);
```

```
OUTPUT R(P7:= P); ER B RET(P); FIX;
P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x × 13);
OUTPUT R(P8:= P); ER B RET(P); FIX;
P:= SIMPL(DER(P,x) × (1 - x↑2) + P × x × 15);
OUTPUT R(P9:= P);

alpha:= sqrt(⏨-3/sqrt(3));
OUTPUT R(ERROR:=
SUBST(P,x,alpha)×(1 - alpha↑2)↑(-17/2) × alpha↑8/(9×8×7×6×5×4×3×2));
END;
```

```
Results RPR100168/02
P2:= x;
P3:= 2×x↑2+1;
P4:= 6×x↑3+9×x;
P5:= 24×x↑4+72×x↑2+9;
P6:= 120×x↑5+600×x↑3+225×x;
P7:= 720×x↑6+.54⏨4×x↑4+4050×x↑2+225;
P8:= .504⏨4×x↑7+.5292⏨5×x↑5+.6615⏨5×x↑3+.11025⏨5×x;
P9:= .4032⏨5×x↑8+.56448⏨6×x↑6+.10584⏨7×x↑4+.3528⏨6×x↑2+.11025⏨5;
ERROR:= .345516723697⏨-14;
ready
line number = 24 execution time = 16 sec
```

We, finally, give the procedure *ARCTAN*:

```
integer procedure ARCTAN(z); value z; integer z;
begin real modz; integer a; DE(a,true); modz:= mod(z);
  if modz > 2 then
  begin ASSIGN(a,D(RN(pi/2),ARCTAN(Q(1,z))));
    if SPECIAL ARGUMENT then R[a]:=  R[a] + mult of 2pi(2 × R[a])/2
  end else
  if modz < .5 then
  begin integer b,delta,ksi,bstar; Boolean SA; SA:= false;
    DE(b,DE(delta,DE(ksi,DE(bstar,true))));
    if SPECIAL ARGUMENT then
    begin SA:= true; SPECIAL ARGUMENT:= false end;
    ASSIGN(a,1); ASSIGN(b,SQRT(SR(1,P(z,z))));
  again: ASSIGN(delta,D(a,b));
    if mod(delta) ≤ 10-3 then goto out;
    ASSIGN(a,PR(.5,S(a,b)));
    ASSIGN(b,SQRT(P(a,b))); goto again;
  out: ASSIGN(ksi,PR(-.5,Q(delta,b)));
    ASSIGN(bstar,S(PR(.25,a),PR(.75,b)));
    ASSIGN(bstar,D(bstar,Q(P(delta,delta),PR(32,bstar))));
    ASSIGN(a,Q(P(z,SR(1,P(ksi,SR(1/6,P(ksi,SR(3/40,
    PR(5/112,ksi)))))),bstar));
  if SA then
    begin SPECIAL ARGUMENT:= true; R[a]:= R[a] + mult of 2pi(2 × R[a])/2
    end; ERASE
  end else ASSIGN(a,P(CN(0,-.5),LN(Q(SR(1,P(iu,z)),D(1,P(iu,z))))));
  ERASE; ASSIGN(p,a); ARCTAN:= p
end ARCTAN;
```

4.3.2.4. The arcsin function

For $|z| < .5$, *ARCSIN* is calculated by means of the expression:

$$\arcsin z = \arctan(a/(1 - z^2)^{\frac{1}{2}}).$$

For $|z| \geq .5$, *ARCSIN* is calculated by means of two different expressions. If $|iz + \sqrt{1 - z^2}| \geq 1$, it is calculated from

$$\arcsin z = \frac{1}{i} \ln(iz + \sqrt{1 - z^2}),$$

and in the other case from

$$\arcsin z = -\frac{1}{i} \ln(-iz + \sqrt{1 - z^2}).$$

```
integer procedure ARCSIN(z); value z; integer z;
begin real modz; integer a,b,k; Boolean SA; DE(a,DE(b,true)); modz:= mod(z);
  if modz < .5 then
  begin SA:= false; if SPECIAL ARGUMENT then
    begin SA:= true; SPECIAL ARGUMENT:= false end;
    ASSIGN(a,ARCTAN(Q(z,SQRT(D(1,P(z,z))))));
    if SA then
    begin SPECIAL ARGUMENT:= true;
      k:= entier(LOWER BOUND ON ARGUMENT/(2 × pi));
      ASSIGN(a,if k - (k ÷ 2) × 2 = 0 then
        D(RN(pi + mult of 2pi(pi - R[a])),a) else
        SR(mult of 2pi(R[a]),a))
    end
  end else
  begin ASSIGN(a,SQRT(D(1,P(z,z))));
    ASSIGN(b,S(P(iu,z),a)); if mod(b) ≥ 1 then
    ASSIGN(a,P(CN(0,-1),LN(b))) else
    begin ASSIGN(a,P(iu,LN(D(a,P(iu,z)))));
      if SPECIAL ARGUMENT then ASSIGN(a,SR(mult of 2pi(R[a]),a))
    end
  end; ERASE; ASSIGN(p,a); ARCSIN:= p
end ARCSIN;
```

4.3.2.5. The arccos function

For $|z + i\sqrt{1 - z^2}| \geq 1$, *ARCCOS* is calculated from

$$\arccos z = \frac{1}{i} \ln(z + i\sqrt{1 - z^2}),$$

and in the other case from

$$\arccos z = -\frac{1}{i} \ln(z - i\sqrt{1 - z^2}).$$

To use a special formula for small $|z|$ is not necessary since

$$\arccos\ (z(1 + \delta)) \simeq \arccos z - z\delta,$$

and $$\arccos(z) \simeq \pi/2 - z;$$

thus, the relative error δ in z is also the relative error in arccos(z).

```
integer procedure ARCCOS(z); value z; integer z;
begin integer a,b; DE(a,DE(b,true));
  ASSIGN(a,P(iu,SQRT(D(1,P(z,z)))));
  ASSIGN(b,S(z,a)); if mod(b) ≥ 1 then
  ASSIGN(a,P(CN(0,-1),LN(b))) else
  begin ASSIGN(a,P(iu,LN(D(z,a))));
    if SPECIAL ARGUMENT then
    ASSIGN(a,SR(mult of 2pi(R[a]),a))
  end; ERASE; ASSIGN(p,a); ARCCOS:= p
end ARCCOS;
```

4.3.2.6. The procedure *INITIALIZE*

```
procedure INITIALIZE;
begin p:= 0; pi:= 3.14159265358979; null:= ₁₀-600;
  CN(0,0); p:= 1; CN(1,0); p:= iu:= 2; CN(0,1); kp:= 1; POINTER[1]:= p:= 3;
  LOWER BOUND ON ARGUMENT:= - pi + ₁₀-12; SPECIAL ARGUMENT:= false
end;
```

4.4. A test program

In this section some results with regard to error analysis are given.

Moreover,a test program is reproduced together with its output. This test program performs the main part of the error analysis.

It calculates for

LOWER BOUND ON ARGUMENT = $-$ *pi*, $-$ *pi*/2, 0, *pi*/2, *pi*, 3/2×*pi* and 2×*pi*,

for

Re(z) = -1000, -10, -1, -.1, -.001, 0, .001, .1, 1, 10 and 1000,

and for

Im(z) = -1000, -10, -1, -.1, -.001, 0, .001, .1, 1, 10 and 1000,

in each quadrant of the complex z-plane the maximum of the following expressions:

$|z - EXP(LN(z))|/|z|$,

$|z - (z^{1/3})^3|/|z|$,

$|z - (SQRT(z))^2|/|z|$,

$|z - SIN(ARCSIN(z))|/|z|$,

$|z - COS(ARCCOS(z))|/|z|$ and

$|z - TAN(ARCTAN(z))|/|z|$.

No calculations were performed in the last but one case for $|z| \leq .1$ and in the last case for $|z| \geq 10$.

The output of the program appears in five columns. The first column indicates which function was tested, the next four columns contain for each quadrant of the z plane the following numbers: *max error*, *real part of z max*, *imaginary part of z max*, where *max error* is the maximum of all errors calculated in the corresponding quadrant and where *z max* is the value of z for which the maximal error occurred.

```
INITIALIZE;
begin comment Test program for Complex Arithmetic RPR 190367/01;
  integer case,i,z; real real part of z, imag part of z, mod of z;
  array max error[1:4,1:6]; integer array z max[1:4,1:6];
  procedure ERROR(f); integer f;
  begin real error; i:= i + 1;
    error:= mod(D(z,f))/mod of z;
    if max error[case,i] < error then
    begin max error[case,i]:= error; ASSIGN(z max[case,i],z) end;
  end ERROR;
  procedure OUTPUT(string); string string;
  begin i:= i + 1; NLCR; PRINTTEXT(string);
    for case:= 1,2,3,4 do
    begin FLOT(2,2,max error[case,i]);
      FLOT(1,1,R[z max[case,i]]);
      FLOT(1,1,I[z max[case,i]]); PRINTTEXT(⟨  ⟩)
    end
  end OUTPUT;
  DE(z,true); for case:= 1,2,3,4 do
  for i:= 1,2,3,4,5,6 do DE(z max[case,i],false);
  NLCR; PRINTTEXT(⟨Results from test program RPR 190367/01⟩);
  for LOWER BOUND ON ARGUMENT:= -pi+⏨-10, -pi/2+⏨-10, ⏨-10,
    pi/2+⏨-10, pi+⏨-10, 3/2×pi+⏨-10, 2×pi+⏨-10 do
  begin for case:= 1,2,3,4 do for i:= 1,2,3,4,5,6 do
    begin ASSIGN(z max[case,i],0); max error[case,i]:= 0 end;
    for real part of z:=
      -1000, -10, -1, -.1, -.001, 0, .001, .1, 1, 10, 1000 do
    for imag part of z:=
      -1000, -10, -1, -.1, -.001, 0, .001, .1, 1, 10, 1000 do
      begin if abs(real part of z) + abs(imag part of z) < ₁₀-5
        then goto END;
        i:= 0; ASSIGN(z,CN(real part of z, imag part of z));
        mod of z:= mod(z);
        case:= entier(arg(z)/(pi/2) - ₁₀-5);
        case:= case - entier(case/4) × 4 + 1;
        ERROR(EXP(LN(z)));
        ERROR(INT POW(POWER(z,RN(1/3)),3));
        ERROR(INT POW(SQRT(z),2));
        ERROR(SIN(ARCSIN(z)));
        if mod of z > .1 then ERROR(COS(ARCCOS(z))) else i:= i + 1;
        if mod of z < 10 then ERROR(TAN(ARCTAN(z)));
      END: end; SPECIAL ARGUMENT:= true; i:= 0;
      NLCR; PRINTTEXT(⌈LOWER BOUND ON ARGUMENT =⌉);
      FIXT(1,1,LOWER BOUND ON ARGUMENT/pi); PRINTTEXT(⌈× pi⌉);
      OUTPUT(
          ⌈EXP(LN(z))      ⌉);
      OUTPUT(
          ⌈(z↑1/3)↑3       ⌉);
      OUTPUT(
          ⌈SQRT(z)↑2       ⌉);
      OUTPUT(
          ⌈SIN(ARCSIN(z)) ⌉);
      OUTPUT(
          ⌈COS(ARCCOS(z))          ⌉);
      OUTPUT(
          ⌈TAN(ARCTAN(z))          ⌉)
    end
  end; ERASE
end
```

The input tape contains the number 100.

```
Results from test program RPR 190367/01
LOWER BOUND ON ARGUMENT =-1.0 × pi
EXP(LN(z))      +.56⏨-11 +.1⏨+1 +.1⏨+4   +.58⏨-11 -.1⏨+4 +.1⏨+1   +.58⏨-11 -.1⏨+4 -.1⏨+1   +.56⏨-11 +.1⏨+1 -.1⏨+4
(z↑1/3)↑3       +.13⏨-10 +.1⏨-2 +.1⏨+4   +.13⏨-10 -.1⏨+4 +.1⏨-0   +.13⏨-10 -.1⏨+4 -.1⏨-0   +.14⏨-10 +.1⏨+4 -.1⏨-2
SQRT(z)↑2       +.35⏨-11 +.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 -.1⏨+2   +.35⏨-11 +.1⏨-0 -.1⏨+2
SIN(ARCSIN(z))  +.84⏨-11 +.0⏨-0 +.1⏨+4   +.75⏨-11 -.1⏨+4 +.1⏨+1   +.75⏨-11 -.1⏨+2 -.1⏨+4   +.75⏨-11 +.1⏨+2 -.1⏨+4
COS(ARCCOS(z))  +.20⏨-10 +.1⏨-0 +.1⏨-2   +.34⏨-10 -.1⏨-0 +.1⏨-2   +.34⏨-10 -.1⏨-0 -.1⏨-2   +.20⏨-10 +.1⏨-0 -.1⏨-2
TAN(ARCTAN(z))+.39⏨-11 +.1⏨+1 +.1⏨+1   +.57⏨-11 -.1⏨-0 +.0⏨-0   +.37⏨-11 -.1⏨+1 -.1⏨+1   +.57⏨-11 +.1⏨-0 +.0⏨-0
LOWER BOUND ON ARGUMENT = -.5 × pi
EXP(LN(z))      +.56⏨-11 +.1⏨+1 +.1⏨+4   +.58⏨-11 -.1⏨+4 +.1⏨+1   +.82⏨-11 -.1⏨+4 -.1⏨+2   +.56⏨-11 +.1⏨+1 -.1⏨+4
(z↑1/3)↑3       +.13⏨-10 +.1⏨-2 +.1⏨+4   +.13⏨-10 -.1⏨+4 -.1⏨-2   +.14⏨-10 -.1⏨-2 -.1⏨+4   +.14⏨-10 +.1⏨+4 -.1⏨-2
SQRT(z)↑2       +.35⏨-11 +.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 -.1⏨+2   +.35⏨-11 +.1⏨-0 -.1⏨+2
SIN(ARCSIN(z))  +.84⏨-11 +.0⏨-0 +.1⏨+4   +.75⏨-11 -.1⏨+4 +.1⏨+1   +.10⏨-10 -.1⏨+4 -.1⏨+1   +.75⏨-11 +.1⏨+2 -.1⏨+4
COS(ARCCOS(z))  +.20⏨-10 +.1⏨-0 +.1⏨-2   +.34⏨-10 -.1⏨-0 +.1⏨-2   +.34⏨-10 -.1⏨-0 -.1⏨-2   +.20⏨-10 +.1⏨-0 -.1⏨-2
TAN(ARCTAN(z))+.39⏨-11 +.1⏨+1 +.1⏨+1   +.57⏨-11 -.1⏨-0 +.0⏨-0   +.37⏨-11 -.1⏨+1 -.1⏨-2   +.57⏨-11 +.1⏨-0 +.0⏨-0
LOWER BOUND ON ARGUMENT = +.0 × pi
EXP(LN(z))      +.62⏨-11 +.1⏨-2 +.1⏨+1   +.58⏨-11 -.1⏨+4 +.1⏨+1   +.98⏨-11 +.1⏨-2 -.1⏨+4   +.12⏨-10 +.1⏨+4 -.1⏨+1
(z↑1/3)↑3       +.13⏨-10 +.1⏨-2 +.1⏨+4   +.13⏨-10 -.1⏨+4 -.1⏨-2   +.16⏨-10 +.1⏨-2 -.1⏨+4   +.15⏨-10 +.1⏨-0 -.1⏨+1
SQRT(z)↑2       +.35⏨-11 +.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 -.1⏨+2   +.35⏨-11 +.1⏨-0 -.1⏨+2
SIN(ARCSIN(z))  +.29⏨- 8 +.0⏨-0 +.1⏨-2   +.41⏨- 8 -.1⏨-2 +.0⏨-0   +.29⏨- 8 +.0⏨-0 -.1⏨-2   +.45⏨- 8 +.1⏨-2 +.0⏨-0
COS(ARCCOS(z))  +.12⏨- 9 +.1⏨-0 +.1⏨-2   +.34⏨-10 -.1⏨-0 +.1⏨-2   +.56⏨-10 -.1⏨-0 -.1⏨-0   +.47⏨-10 +.1⏨-0 -.1⏨-2
TAN(ARCTAN(z))+.29⏨- 8 +.0⏨-0 +.1⏨-2   +.16⏨- 8 -.1⏨-2 +.0⏨-0   +.29⏨- 8 +.0⏨-0 -.1⏨-2   +.57⏨-11 +.1⏨-0 +.0⏨-0
```

LOWER BOUND ON ARGUMENT = +.5 × pi

EXP(LN(z))	+.13$_{10}$-10	+.1$_{10}$+4	+.1$_{10}$+2	+.58$_{10}$-11	-.1$_{10}$+4	+.1$_{10}$+1	+.98$_{10}$-11	+.1$_{10}$-2	-.1$_{10}$+4	+.12$_{10}$-10	+.1$_{10}$+4	-.1$_{10}$+1
(z$\uparrow$1/3)$\uparrow$3	+.17$_{10}$-10	+.0$_{10}$-0	+.1$_{10}$+4	+.13$_{10}$-10	-.1$_{10}$+4	-.1$_{10}$-2	+.16$_{10}$-10	+.1$_{10}$-2	-.1$_{10}$+4	+.15$_{10}$-10	+.1$_{10}$-0	-.1$_{10}$+1
SQRT(z)$\uparrow$2	+.35$_{10}$-11	+.1$_{10}$-0	+.1$_{10}$+2	+.35$_{10}$-11	-.1$_{10}$-0	+.1$_{10}$+2	+.35$_{10}$-11	-.1$_{10}$-0	-.1$_{10}$+2	+.35$_{10}$-11	+.1$_{10}$-0	-.1$_{10}$+2
SIN(ARCSIN(z))	+.29$_{10}$- 8	+.0$_{10}$-0	+.1$_{10}$-2	+.41$_{10}$- 8	-.1$_{10}$-2	+.0$_{10}$-0	+.29$_{10}$- 8	+.0$_{10}$-0	-.1$_{10}$-2	+.45$_{10}$- 8	+.1$_{10}$-2	+.0$_{10}$-0
COS(ARCCOS(z))	+.12$_{10}$- 9	+.1$_{10}$-0	+.1$_{10}$-2	+.56$_{10}$-10	-.1$_{10}$-0	+.1$_{10}$-0	+.56$_{10}$-10	-.1$_{10}$-0	-.1$_{10}$-0	+.12$_{10}$- 9	+.1$_{10}$-0	-.1$_{10}$-2
TAN(ARCTAN(z))	+.29$_{10}$- 8	+.0$_{10}$-0	+.1$_{10}$-2	+.16$_{10}$- 8	-.1$_{10}$-2	+.0$_{10}$-0	+.29$_{10}$- 8	+.0$_{10}$-0	-.1$_{10}$-2	+.12$_{10}$- 8	+.1$_{10}$-2	+.0$_{10}$-0

LOWER BOUND ON ARGUMENT =+1.0 × pi

EXP(LN(z))	+.13$_{10}$-10	+.1$_{10}$+4	+.1$_{10}$+2	+.12$_{10}$-10	-.1$_{10}$+4	+.1$_{10}$+1	+.98$_{10}$-11	+.1$_{10}$-2	-.1$_{10}$+4	+.12$_{10}$-10	+.1$_{10}$+4	-.1$_{10}$+1
(z$\uparrow$1/3)$\uparrow$3	+.17$_{10}$-10	+.0$_{10}$-0	+.1$_{10}$+4	+.16$_{10}$-10	-.1$_{10}$+4	+.1$_{10}$-0	+.16$_{10}$-10	+.1$_{10}$-2	-.1$_{10}$+4	+.15$_{10}$-10	+.1$_{10}$-0	-.1$_{10}$+1
SQRT(z)$\uparrow$2	+.35$_{10}$-11	+.1$_{10}$-0	+.1$_{10}$+2	+.35$_{10}$-11	-.1$_{10}$-0	+.1$_{10}$+2	+.35$_{10}$-11	-.1$_{10}$-0	-.1$_{10}$+2	+.35$_{10}$-11	+.1$_{10}$-0	-.1$_{10}$+2
SIN(ARCSIN(z))	+.72$_{10}$- 8	+.1$_{10}$-2	+.1$_{10}$-2	+.41$_{10}$- 8	-.1$_{10}$-2	+.0$_{10}$-0	+.88$_{10}$- 9	-.1$_{10}$-2	-.1$_{10}$-2	+.72$_{10}$- 8	+.1$_{10}$-2	-.1$_{10}$-2
COS(ARCCOS(z))	+.12$_{10}$- 9	+.1$_{10}$-0	+.1$_{10}$-2	+.56$_{10}$-10	-.1$_{10}$-0	+.1$_{10}$-0	+.56$_{10}$-10	-.1$_{10}$-0	-.1$_{10}$-0	+.12$_{10}$- 9	+.1$_{10}$-0	-.1$_{10}$-2
TAN(ARCTAN(z))	+.29$_{10}$- 8	+.0$_{10}$-0	+.1$_{10}$-2	+.16$_{10}$- 8	-.1$_{10}$-2	+.0$_{10}$-0	+.29$_{10}$- 8	+.0$_{10}$-0	-.1$_{10}$-2	+.12$_{10}$- 8	+.1$_{10}$-2	+.0$_{10}$-0

```
LOWER BOUND ON ARGUMENT =+1.5 × pi
EXP(LN(z))       +.13⏨-10 +.1⏨+4 +.1⏨+2   +.15⏨-10 -.1⏨+4 -.1⏨-2   +.19⏨-10 -.1⏨-2 -.1⏨+4   +.12⏨-10 +.1⏨+4 -.1⏨+1
(z↑1/3)↑3        +.17⏨-10 +.0⏨-0 +.1⏨+4   +.21⏨-10 -.1⏨+4 -.1⏨-2   +.19⏨-10 -.1⏨+1 -.1⏨-2   +.15⏨-10 +.1⏨-0 -.1⏨+1
SQRT(z)↑2        +.35⏨-11 +.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 -.1⏨+2   +.35⏨-11 +.1⏨-0 -.1⏨+2
SIN(ARCSIN(z))   +.72⏨- 8 +.1⏨-2 +.1⏨-2   +.98⏨- 8 -.1⏨-2 +.0⏨-0   +.88⏨- 9 -.1⏨-2 -.1⏨-2   +.72⏨- 8 +.1⏨-2 -.1⏨-2
COS(ARCCOS(z))   +.12⏨- 9 +.1⏨-0 +.1⏨-2   +.64⏨-10 -.1⏨-2 +.1⏨-0   +.64⏨-10 -.1⏨-2 -.1⏨-0   +.12⏨- 9 +.1⏨-0 -.1⏨-2
TAN(ARCTAN(z))+.29⏨- 8 +.0⏨-0 +.1⏨-2      +.16⏨- 8 -.1⏨-2 +.0⏨-0   +.29⏨- 8 +.0⏨-0 -.1⏨-2   +.12⏨- 8 +.1⏨-2 +.0⏨-0
LOWER BOUND ON ARGUMENT =+2.0 × pi
EXP(LN(z))       +.13⏨-10 +.1⏨+4 +.1⏨+2   +.15⏨-10 -.1⏨+4 -.1⏨-2   +.19⏨-10 -.1⏨-2 -.1⏨+4   +.18⏨-10 +.1⏨+4 -.1⏨-0
(z↑1/3)↑3        +.17⏨-10 +.0⏨-0 +.1⏨+4   +.21⏨-10 -.1⏨+4 -.1⏨-2   +.19⏨-10 -.1⏨+1 -.1⏨-2   +.14⏨-10 +.1⏨+4 +.1⏨-2
SQRT(z)↑2        +.35⏨-11 +.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 +.1⏨+2   +.35⏨-11 -.1⏨-0 -.1⏨+2   +.35⏨-11 +.1⏨-0 -.1⏨+2
SIN(ARCSIN(z))   +.11⏨- 7 +.0⏨-0 +.1⏨-2   +.13⏨- 7 -.1⏨-2 +.0⏨-0   +.11⏨- 7 +.0⏨-0 -.1⏨-2   +.98⏨- 8 +.1⏨-2 +.0⏨-0
COS(ARCCOS(z))   +.64⏨-10 +.1⏨-2 +.1⏨-0   +.64⏨-10 -.1⏨-2 +.1⏨-0   +.62⏨-10 -.1⏨-0 -.1⏨-2   +.62⏨-10 +.1⏨-0 -.1⏨-2
TAN(ARCTAN(z))+.57⏨- 8 +.0⏨-0 +.1⏨-2      +.10⏨- 7 -.1⏨-2 +.0⏨-0   +.57⏨- 8 +.0⏨-0 -.1⏨-2   +.12⏨- 8 +.1⏨-2 +.0⏨-0
```

Some other tests were performed for *ARCTAN*, *ARCSIN* and *ARCCOS*.
z assumed the values

$$z = \rho\, e^{i\phi}$$

where $\phi = 0 \quad (pi/10) \quad 1.9\ pi$.

Only the principal values were calculated, i.e. *SPECIAL ARGUMENT* = *false*.

The results are:

$|z - TAN(ARCTAN(z))|/\rho$

reached	for	$\rho = .49$	the	maximal	value	$.87_{10}-11$	for	$\phi = 1.9$ pi
"	"	$\rho = .1$	"	"	"	$.69_{10}-11$	"	$\phi = 1.8$ pi
"	"	$\rho = .001$	"	"	"	$.36_{10}-11$	"	$\phi = 0$
"	"	$\rho = 10$	"	"	"	$.36_{10}-10$	"	$\phi = .8$ pi

$|z - SIN(ARCSIN(z))|/\rho$

reached	for	$\rho = .49$	the	maximal	value	$.59_{10}-11$	for	$\phi = 1.7$ pi
"	"	$\rho = .1$	"	"	"	$.57_{10}-11$	"	$\phi = 1.1$ pi
"	"	$\rho = .001$	"	"	"	$.36_{10}-11$	"	$\phi = .2$ pi
"	"	$\rho = 10$	"	"	"	$.39_{10}-11$	"	$\phi = 1.2$ pi
"	"	$\rho = 100$	"	"	"	$.54_{10}-11$	"	$\phi = 1.7$ pi

$|z - COS(ARCCOS(z))|/\rho$

reached	for	$\rho = 10$	the	maximal	value	$.47_{10}-11$	for	$\phi = .2$ pi
"	"	$\rho = 100$	"	"	"	$.52_{10}-11$	"	$\phi = 1.7$ pi

Chapter 5

THE CAUCHY PROBLEM

5.1. Introduction

Many physical problems lead to the so-called Cauchy problem (see [5] p. 39); i.e. the determination of functions $U_k(x_1, \ldots, x_d)$, $k = 1,\ldots,K$, satisfying the partial-differential equations:

$$G_l\left(\frac{\partial U_1}{\partial x_1}, \ldots, \frac{\partial U_1}{\partial x_d}, \ldots, \frac{\partial U_K}{\partial x_1}, \ldots, \frac{\partial U_K}{\partial x_d},\right.$$
$$\left.U_1, \ldots, U_K, x_1, \ldots, x_d\right) = C_l, \qquad (5.1.1)$$

for $l = 1, \ldots, K$, where the C_l's are constant,

and the initial conditions:

$$U_k(x_1, \ldots, x_{d-1}, 0) = U_k^0(x_1, \ldots, x_{d-1}),\ k=1,\ldots,K. \qquad (5.1.2)$$

Sometimes,it is possible to develop the functions U_k in Taylor series:

$$U_k(x_1, \ldots, x_d) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_d=0}^{\infty} u_{k,n_1,\ldots,n_d}\, x_1^{n_1} \cdots x_d^{n_d}, \qquad (5.1.3)$$

for $k=1,\ldots,K$.

In this chapter,we shall be concerned with the calculation of the Taylor coefficients $u_{k,n_1,\ldots,n_d}$.
(Here and in the sequel,we denote Taylor coefficients by small letters and the functions to which they belong by corresponding capital letters.)

In general, the calculation of the $u_{k,n_1,\ldots,n_d}$ is elementary but tedious; therefore, it is perfectly suitable for treatment by means of a formula-manipulation system. The calculation is performed in two steps:
first, the differential equations (5.1.1) are given to a formula-manipulation ALGOL 60 program which investigates the differential equations

algebraically and which outputs special ALGOL 60 statements; second, another ALGOL 60 program, composed of the ALGOL 60 statements produced by the first program, calculates the Taylor coefficients (in a numerical manner). The latter program is (automatically) made as efficient as possible with respect to the storage space, in which numerous Taylor coefficients are stored.

We require in the sequel that the Cauchy problem satisfies the following conditions:

1. The left-hand sides of (5.1.1) are expressions built up with real or integral numbers, variables (as $\partial U_i/\partial x_k$, or U_i, or x_i, or some parameters), the dyadic operators: $+$, $-$, $*$, $/$ and $\uparrow$, and the functions: exp, ln, sin, cos, arctan and sqrt, only.

2. The functions $U_k^0(x_1, \ldots, x_{d-1})$ have a known Taylor-series expansion:

$$U_k^0(x_1, \ldots, x_{d-1}) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_{d-1}=0}^{\infty} u^0_{k,n_1,\ldots,n_{d-1}} x_1^{n_1} \cdots x_{d-1}^{n_{d-1}},$$

for $k=1,\ldots,K$. (5.1.4)

3. If we substitute: for $k = 1, \ldots, K$,

$$u_{k,1,0,\ldots,0,0} = u^0_{k,1,0,\ldots,0},$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$u_{k,0,0,\ldots,1,0} = u^0_{k,0,0,\ldots,1}$$

and

$$u_{k,0,0,\ldots,0,0} = u^0_{k,0,0,\ldots,0},$$

then the equations:

$$G_l(u_{1,1,\ldots,0,0}, \ldots, u_{1,0,\ldots,0,1}, \ldots, u_{K,1,\ldots,0,0}, \ldots, u_{K,0,\ldots,0,1},$$
$$u_{1,0,\ldots,0,0}, \ldots, u_{K,0,\ldots,0,0}, 0, \ldots, 0) = C_l, l = 1,\ldots,K,$$

are explicitly solvable for $u_{k,0,\ldots,0,1}$, $k = 1, \ldots, K$.

These equations are, in general, nonlinear in $u_{k,0,\ldots,0,1}$; we require, therefore, that they are solved already by hand or by computer and that their solution is given together with the initial conditions (5.1.2). An alternative way of looking at these equations is, that they determine the constants C_l, $l = 1, \ldots, K$ in terms of the, now independent, initial data.

4. The left-hand sides of (5.1.1) are analytic functions of their $(Kd + K + d)$ variables in the neighbourhood of the origin defined by:

$$x_1 = \ldots = x_d = 0,$$

$$\left.\begin{aligned} \partial U_k/\partial x_j &= u_{k,0,\ldots,\underset{(j)}{1},\ldots,0}, \quad j=1,\ldots,d, \\ U_k &= u_{k,0,\ldots,0} \end{aligned}\right\} k=1,\ldots,K.$$

5. For $k = 1, \ldots, K$ we introduce $Q_k = \partial U_k/\partial x_d$; it is required that in the origin:

$$\det\{\partial G_l/\partial Q_k\} \neq 0. \qquad (5.1.5)$$

The above conditions are sufficient for the Taylor coefficients to be computable by means of the process as described in this chapter. The conditions are also sufficient for proving that the obtained Taylor series converge (see section 5.3.1).

The problem (5.1.1) seems to be very general, however, it does not enclose the case that for some k the derivative $\partial U_k/\partial x_d$ does not occur in (5.1.1); then, namely, condition 5 is not satisfied. Therefore, condition 5 is weakened: Let t_k be such that $\partial U_k/\partial x_{t_k}$ occurs, but $\partial U_k/\partial x_j$ with $j > t_k$ does not occur. (Note, that t_k may be equal to 0; then $\partial U_k/\partial x_{t_k}$ should be understood as U_k itself.)
We now define $\overline{Q}_k$ by:

$$\overline{Q}_k = \partial U_k/\partial x_{t_k}, \quad k = 1, \ldots, K,$$

and require that in the origin:

$$\det\{\partial G_l/\partial \overline{Q}_k\} \neq 0. \qquad (5.1.5a)$$

The number t_k will be called the <u>type</u> of U_k.

It is obvious that if, for some k, $t_k < d$, then the initial conditions (5.1.2) should be changed appropriately (see section 5.8.5).

In order to simplify the investigations we shall assume in the sequel that $d = 2$ and that $t_k = 2$, for all k. The variables x_1 and x_2 will be denoted by x and y, respectively.
The final ALGOL 60 programs of section 5.7 and 5.8 are, however, made for the general case; moreover, the origin may be chosen arbitrarily.

In the sequel, we shall, moreover, assume that the constants C_l, $l = 1, \ldots, K$, are all zero; this does not harm generality, as they may easily be taken into account by the left-hand sides of equation (5.1.1).

The outline of this chapter is as follows:
Section 5.2 describes the theoretical process for calculating the Taylor coefficients. The purpose of this section is to show the recursion, with respect to the indexes n and m, for calculating $u_{k,n,m}$.
Section 5.3 describes a process for solving the explicit Cauchy problem; i.e. the quantities $\partial U_k/\partial y$ occur explicitly at the left-hand sides of the equations: $\partial U_k/\partial y = H_k(\partial U_1/\partial x, \ldots, \partial U_K/\partial x, U_1, \ldots, U_K, x, y)$.
Section 5.4 gives an outline of the calculation process by means of a simple ALGOL 60 program.
Section 5.5 discusses the separation of the process into an algebraic part and a computational part.
Section 5.6 investigates a more efficient approach in which e.g. the differential equations are transformed, in order to obtain a more efficient computational program.
Section 5.7 discusses the ALGOL 60 program for executing the algebraic part of the problem.
Section 5.8 discusses the computational ALGOL 60 program.
Section 5.9 is devoted to an example from the theory of "blunt-body problems" in aerodynamics.
Finally, section 5.10 lists some devices to overcome certain shortcomings of the system, and it mentions some problems for future investigations.

5.2. A simple but unpractical calculation process

By means of a simple, but, as will turn out, unpractical calculation process, it will be shown that the Taylor coefficients are indeed computable. This process, in principle described in [5] p. 39, demonstrates one important facet of the process to be described in sections 5.6 and 5.7, namely the recursion with respect to the indexes n and m of $u_{k,n,m}$.

The functions G_1 of (5.1.1) are differentiated n times with respect to x, and m times with respect to y (we assume $n+m > 0$). The result is:

$$\sum_{k=1}^{K} \left(\frac{\partial G_1}{\partial P_k} \frac{\partial^{n+m} P_k}{\partial x^n \partial y^m} + \frac{\partial G_1}{\partial Q_k} \frac{\partial^{n+m} Q_k}{\partial x^n \partial y^m} + \frac{\partial G_1}{\partial U_k} \frac{\partial^{n+m} U_k}{\partial x^n \partial y^m}\right) + \ldots = 0 \qquad (5.2.1)$$

where $P_k = \partial U_k/\partial x$ and $Q_k = \partial U_k/\partial y$, and where the dots represent terms involving:

$$\left.\begin{aligned} \overline{P}_{k,i,j} &= \frac{\partial^{i+j} P_k}{\partial x^i \partial y^j} \\ \overline{Q}_{k,i,j} &= \frac{\partial^{i+j} Q_k}{\partial x^i \partial y^j} \end{aligned}\right\} \quad \text{with } i \le n,\ j \le m \text{ and } i+j \ne n+m,$$

$$\overline{U}_{k,i,j} = \frac{\partial^{i+j} U_k}{\partial x^i \partial y^j}, \qquad \text{with } i \le n \text{ and } j \le m,$$

and terms involving partial derivatives of G_1 with respect to x, y, P_k, Q_k and U_k. In the origin we have:

$$\overline{P}_{k,i,j} = (i+1)!j!u_{k,i+1,j},$$

$$\overline{Q}_{k,i,j} = i!(j+1)!u_{k,i,j+1},$$

$$\overline{U}_{k,i,j} = i!\,j!\,u_{k,i,j}.$$

Thus, equations (5.2.1) are, in the origin, equations for the Taylor coefficients $u_{k,i,j}$ with $k = 1, \ldots, K$ and (i,j) in $R_{n,m}$ defined by

$$R_{n,m} = \{(i,j):\ i \le n+1 \wedge j \le m+1 \wedge i+j \ne n+m+2\}. \qquad (5.2.2)$$

Next, we observe that equations (5.2.1) are linear in the $u_{k,n,m+1}$. Due to condition (5.1.5) we may solve the equations for the $u_{k,n,m+1}$, thus expressing the $u_{k,n,m+1}$ in terms of the $u_{k,i,j}$ with (i,j) in $S_{n,m}$ defined by:

$$S_{n,m} = R_{n,m} \setminus \{(n,m+1)\}, \qquad (5.2.3)$$

(See figure 1.)

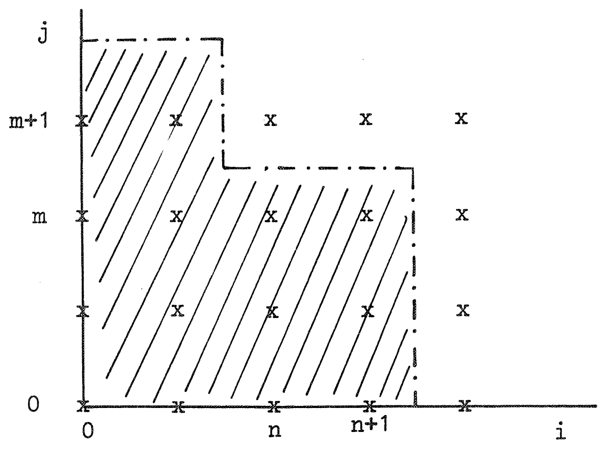

fig. 1. The shaded region is $S_{n,m}$

By hypothesis, the $u_{k,i,0}$ and the $u_{k,0,1}$ are known from the initial conditions.

The following calculation scheme presents itself:

First, let $n = 1$, $m = 0$; then $S_{n,m}$ consists of the points: (0,0), (1,0), (2,0) and (0,1) in which the $u_{k,i,j}$ are known; thus, we can calculate the $u_{k,1,1}$.

Next, let $n = 0$, $m = 1$; then $S_{n,m}$ consists of the points: (0,0), (1,0), (0,1) and (1,1), in which the $u_{k,i,j}$ are known; thus, we can calculate the $u_{k,0,2}$.

In general, if index pair (n_1,m_1) has been treated, then the next index pair (n_2,m_2) is defined by:

if $n_1 > 0$, then $n_2 = n_1 - 1$ and $m_2 = m_1 + 1$;

otherwise, $n_2 = m_1 + 1$ and $m_2 = 0$,

and the u_{k,n_2,m_2+1}, $k = 1, \ldots, K$ can be calculated. (See figure 2.)

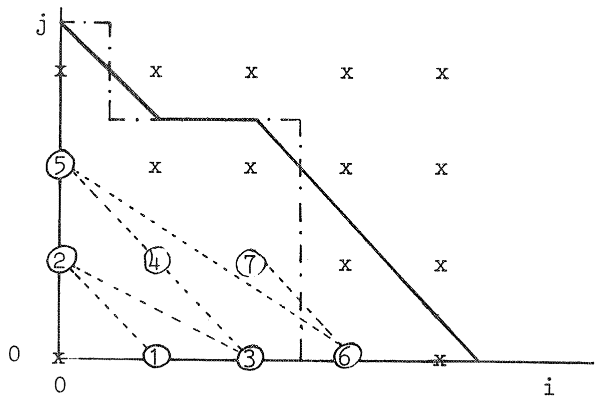

fig. 2. The successive values of (n,m)

The -.- line encloses the region $S_{n,m}$

The —— line encloses the region of $u_{k,i,j}$'s already calculated.

A direct and easy conclusion from fig. 2 is that, if it is the turn for (n,m) to be treated, then all the points of $S_{n,m}$ ly in the region of already calculated $u_{k,i,j}$; thus, the $u_{k,n,m+1}$ can indeed be calculated.

Note, that if the quantity $\partial U_k/\partial y$ does not occur in the partial-differential equations, but, instead, either $\partial U_k/\partial x$, or U_k itself occurs as "highest" derivative, then the same process as above can be used; now, however, the equations (5.2.1) should be solved for either $u_{k,n+1,m}$, or $u_{k,n,m}$ which occur also linearly in (5.2.1).

The method just described can be laid down in an ALGOL 60 formula-manipulation program which differentiates the functions G_1; this is done, for example, by Perlis, Itturiaga and Standish [12] . There is, however, a major disadvantage attached to this approach, namely, that the derivatives of the differential equations, on the form of which we do not want to impose severe restrictions, will,in general,become very lengthy so that the memory capacity of the computer may easily become exhausted.

We choose therefore an alternative method, which is more arduous to describe but which has not the above disadvantage.

In short, this method consists of calculating the equations (5.2.1) in a numerical way, instead of the analytic way by differentiation.

5.3. The explicit Cauchy problem

5.3.1. Introduction

The ordinary, explicit Cauchy problem, as given in [5] p. 39, has the form:

Determine the functions U_k satisfying

$$\frac{\partial U_k}{\partial y} = H_k(\frac{\partial U_1}{\partial x}, \ldots, \frac{\partial U_K}{\partial x}, U_1, \ldots, U_K, x, y), \qquad k=1,\ldots,K, \tag{5.3.1}$$

and the initial conditions (5.1.2).

For the functions H_k we require the same conditions as for the functions G_l in (5.1.1), except for conditions 3 and 5 which are now meaningless.

There are several reasons for treating problem (5.3.1):

1. An historical reason; already in 1960, R.D. Richtmyer [13] as well as A. Gibbons [6] described (machine-code) programs which calculate the Taylor coefficients of the functions U_k by means of a process to be discussed in this section. Also R.E. Moore [10] has investigated the problem as part of a study on error analysis.

2. A practical reason; the calculation scheme for the more complicated problem (5.1.1) is a direct generalization of the calculation scheme which will be developed in this section. The latter scheme is, apart for the treatment of elementary functions and integral powers, already given by Gibbons.

3. A mathematical reason; from the known theorem of Cauchy-Kowalewski, which asserts the existence and convergence of the Taylor series for the functions U_k satisfying (5.3.1) and (5.1.2),it will be proved that the Taylor series of the functions U_k satisfying (5.1.1) and (5.1.2) exist and converge also.

Let us first give the convergence proof mentioned under 3. Introduce K new functions V_k with

$$\frac{\partial U_k}{\partial y} = V_k; \tag{5.3.2}$$

differentiate (5.1.1) with respect to y, obtaining:

$$\sum_{k=1}^{K} \left\{\frac{\partial G_l}{\partial P_k}\frac{\partial V_k}{\partial x} + \frac{\partial G_l}{\partial Q_k}\frac{\partial V_k}{\partial y} + \frac{\partial G_l}{\partial U_k} V_k\right\} + \frac{\partial G_l}{\partial y} = 0 \qquad (5.3.3)$$

where $P_k = \partial U_k/\partial x$, $Q_k = \partial U_k/\partial y$.

Due to condition (5.1.5),the equations (5.3.3) may be written as

$$\frac{\partial V_k}{\partial y} = \overline{H}_k(\frac{\partial V_1}{\partial x}, \ldots, \frac{\partial V_K}{\partial x}, \frac{\partial U_1}{\partial x}, \ldots, \frac{\partial U_K}{\partial x}, V_1, \ldots, V_K, U_1, \ldots, U_K, x, y). \qquad (5.3.4)$$

For the initial conditions of the functions V_k we choose

$$V_k(x,0) = \frac{\partial U_k}{\partial y}(x,0), \qquad (5.3.5)$$

where the $\frac{\partial U_k}{\partial y}(x,0)$ are calculated from (5.1.1) in which we have substituted:

$$\frac{\partial U_k}{\partial x}(x,0) = \frac{dU_k^0}{dx}, \; U_k(x,0) = U_k^0(x), \text{ and } y = 0.$$

The equations (5.3.2) and (5.3.4) together are of the form (5.3.1); moreover, the functions $\overline{H}_k$ and the initial conditions (5.3.5) are analytic in a neighbourhood of the origin; therefore,the Cauchy-Kowalewski theorem asserts the existence and convergence of the Taylor series for the functions U_k and V_k.

Remarks

1. The proof that the U_k obtained from (5.3.2) and (5.3.4) is also the solution of (5.1.1), is trivial.
2. One might ask: it is shown that problem (5.1.1) may be treated as a problem of the simpler form (5.3.1); why should problem (5.1.1) be investigated at all? The answer is fourfold:
   a. The differentiation and the inversion of the matrix $\partial G_l/\partial Q_k$ must be done beforehand, which may be difficult.

b. The resulting left-hand sides of (5.3.4) will, in general, be more lengthy than the formulae G_l. (See section 5.9.1.)

c. The extra initial conditions for V_k should be calculated beforehand, which may be difficult.

d. The treatment of problems of the form (5.3.1) does not necessarily involve a formula-manipulation system and would, therefore, be of little interest in the scope of this treatise; in the treatment of the problem (5.1.1), however, formula manipulation will turn out to be necessary in a natural way.

5.3.2. The calculation scheme for the explicit Cauchy problem

We now proceed in defining the process for calculating the Taylor coefficients of functions U_k satisfying (5.3.1) and (5.1.2).

The functions H_k are formulae as defined in chapter 2; hence, they can be stored in the computer as trees consisting of subformulae. Each of the subformulae and the formulae H_k themselves can be written in a simple form using at most one operand and at most two subformulae.

Example: Let

$$H_1 = U_1 \times \partial U_1/\partial x + x; \qquad (5.3.6)$$

this formula gives rise to the following subformulae, written in the simple form:

$$\left.\begin{aligned} A_1 &= U_1 , \\ A_2 &= \partial U_1/\partial x , \\ A_3 &= A_1 \times A_2 , \\ A_4 &= x , \\ (H_1 =) A_5 &= A_3 + A_4 . \end{aligned}\right\} \qquad (5.3.7)$$

As already indicated in the example, we denote the subformulae of the H_k by A_i, $i = 1, 2, \ldots$.

Each A_i is an analytic function of x and y. (This does not follow from condition 4 of section 5.1; one may have e.g. the function sin(x)×(1/x) which is analytic for x = 0, whereas the subformula 1/x is not. We shall discard, however, such pathological cases.) Thus each A_i has a Taylor series:

$$A_i(x,y) = \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} a_{i,n,m}\, x^n\, y^m. \tag{5.3.8}$$

The calculation process is as follows:

Let (n,m) = (0,0), (1,0), (0,1), (2,0), (1,1), ..., i.e. the sequence of index pairs as given in section 5.2. For given (n,m), $a_{i,n,m}$ is calculated according to the following table. (Note that elementary functions and powers will be treated separately):

if A_i =	then $a_{i,n,m}$ =
$A_j \pm A_k$	$a_{j,n,m} \pm a_{k,n,m}$
$A_j \times A_k$	$\sum_{p=0}^{n} \sum_{q=0}^{m} a_{j,p,q}\, a_{k,n-p,m-q}$
A_j/A_k	$\{a_{j,n,m} - \sum_{p=0}^{n} \sum_{\substack{q=0 \\ p+q \neq n+m}}^{m} a_{i,p,q}\, a_{k,n-p,m-q}\}/a_{k,0,0}$
$\partial U_k/\partial x$	$(n+1)u_{k,n+1,m}$
U_k	$u_{k,n,m}$
x	if $n = 1 \wedge m = 0$ then 1,otherwise 0
y	if $n = 0 \wedge m = 1$ then 1,otherwise 0
a number N	if $n = 0 \wedge m = 0$ then N,otherwise 0

table 1. Calculation process for elementary operations.

The value of $h_{k,n,m}$ can thus be calculated and $u_{k,n,m+1}$ follows from

$$u_{k,n,m+1} = \frac{1}{m+1} h_{k,n,m}. \qquad (5.3.9)$$

For the example (5.3.7) we have

$$a_{1,n,m} := u_{1,n,m};$$

$$a_{2,n,m} := (n+1)u_{1,n+1,m};$$

$$a_{3,n,m} := \sum_{p=0}^{n} \sum_{q=0}^{m} a_{1,p,q}\; a_{2,n-p,m-q};$$

$$a_{4,n,m} := \text{if } n = 1 \wedge m = 0 \text{ then } 1 \text{ otherwise } 0;$$

$$(h_{1,n,m} :=) a_{5,n,m} := a_{3,n,m} + a_{4,n,m}$$

$$\text{and } u_{1,n,m+1} := h_{1,n,m}/(m+1).$$

Starting with known values for $u_{1,i,0}$ (from the initial conditions), we first calculate $u_{1,0,1}$ in terms of $u_{1,0,0}$; next, we calculate $u_{1,1,1}$ in terms of $u_{1,0,0}$, $u_{1,2,0}$ and $u_{1,0,1}$; then $u_{1,0,2}$ is calculated, etc.

5.3.3. The recurrence relations for elementary functions and powers

Consider the subformula $A_i = \Phi(A_j)$, where the function Φ may stand for: exp, ln, sin, cos, arctan, and sqrt. Note that a power f↑g, where "g" is not an integer, may be treated as exp(g*ln(f)).
For each possible choice of the function Φ, we can introduce three functions Ψ_1, Ψ_2 and Φ_2 such that

$$\left.\begin{aligned} d\Phi(z)/dz &= \Psi_1(\Phi(z),\ \Phi_2(z),\ z), \\ d\Phi_2(z)/dz &= \Psi_2(\Phi(z),\ \Phi_2(z),\ z). \end{aligned}\right\} \qquad (5.3.10)$$

In the sequel we shall write

Φ_1^j for $\Phi(A_j)$,

Φ_2^j for $\Phi_2(A_j)$,

Ψ_k^j for $\Psi_k(\Phi(A_j),\ \Phi_2(A_j),\ A_j)$, $k = 1, 2$,

and z_j for A_j.

The definition of the functions Φ_ν^j and Ψ_ν^j is given in the following table:

Φ_1^j	Φ_2^j	Ψ_1^j	Ψ_2^j
$\exp(z_j)$		Φ_1^j	
$\ln(z_j)$		$1/z_j$	
$\sin(z_j)$	$\cos(z_j)$	Φ_2^j	$(-1) \times \Phi_1^j$
$\cos(z_j)$	$\sin(z_j)$	$(-1) \times \Phi_2^j$	Φ_1^j
$\arctan(z_j)$		$1/(1+z_j \times z_j)$	
$\mathrm{sqrt}(z_j)$		$.5/\Phi_1^j$	

table 2. The differential equations for elementary functions

The coefficient $a_{i,n,m}$ of $A_i = \Phi(A_j)$ can be calculated if the $a_{j,p,q}$ are known for $0 \le p \le n$, $0 \le q \le m$, as follows.
From (5.3.10) it follows, after differentiation with respect to x, that

$$\frac{\partial \Phi_\nu^j}{\partial x} = \Psi_\nu^j(\Phi_1^j, \Phi_2^j, A_j) \frac{\partial A_j}{\partial x} \quad ; \qquad (5.3.11)$$

thus, if the Taylor coefficients of Φ_ν^j and of Ψ_ν^j are denoted by $\phi_{\nu,n,m}^j$ and $\psi_{\nu,n,m}^j$, respectively, then we have the following equation:

$$(n+1)\phi^j_{\nu,n+1,m} = \sum_{p=0}^{n} \sum_{q=0}^{m} \psi^j_{\nu,p,q}(n-p+1)a_{j,n-p+1,m-q};$$

or,changing n+1 into n and dividing by n:

$$\phi^j_{\nu,n,m} = \frac{1}{n} \sum_{p=0}^{n} \sum_{q=0}^{m} \psi^j_{\nu,p,q}(n-p)a_{j,n-p,m-q}. \qquad (5.3.12)$$

In the case that $n = 0$,we get a similar formula by differentiating A_j with respect to y:

$$\phi^j_{\nu,n,m} = \frac{1}{m} \sum_{p=0}^{n} \sum_{q=0}^{m} \psi^j_{\nu,p,q}(m-q)a_{j,n-p,m-q}. \qquad (5.3.13)$$

As can be seen from table 2, the structure of the functions ψ^j_ν is the structure of a formula in which no special functions occur; their variables are: ϕ^j_1, ϕ^j_2 and z_j.

This means that if we add to table 1 the following table:

If A_i =	then $a_{i,n,m}$ =
ϕ^j_1	$\phi^j_{1,n,m}$
ϕ^j_2	$\phi^j_{2,n,m}$
z_j	$a_{j,n,m}$

table 3. The calculation of elementary functions

then the $\psi^j_{\nu,n,m}$ can be calculated using this table and table 1.
Finally, by means of (5.3.12) and (5.3.13) the Taylor coefficients $a_{i,n,m}$ of $A_i = \Phi(A_j)$ can be calculated.
The equations (5.3.12) and (5.3.13) can be used only if both n and m differ from zero. In the case that n = 0 and m = 0, we may calculate the $\phi^j_{\nu,0,0}$ (and thus the $a_{i,0,0}$) from

$$\phi^j_{\nu,0,0} = \Phi^j_\nu(a_{j,0,0}). \tag{5.3.14}$$

Remark: It is possible to introduce direct recursion relations for the special functions; for example if $A_i = \exp(A_j)$, then

$$a_{i,n,m} = \frac{1}{n} \sum_{p=0}^{n} \sum_{q=0}^{m} a_{i,p,q}(n-p)a_{j,n-p,m-q},$$

which would be more simple than the above sketched procedure; in fact, this is the method used by Gibbons [6].
Our method has, however, the advantage that other special functions can easily be implemented in the program. Take for example the Bessel-function $J_n(z)$ satisfying $z^2J''_n + zJ'_n + (z^2 - n^2)J_n = 0$. Introducing $\Phi_2(z) = J'_n(z)$ we have

$$\frac{d\Phi_1}{dz} = \Psi_1 = \Phi_2,$$

$$\frac{d\Phi_2}{dz} = \Psi_2 = -\Phi_2/z + (n \times n/(z \times z) - 1)\Phi_1.$$

We apply the calculation process to the differential equation

$$\partial U_2/\partial y = \sin(\partial U_2/\partial x) + \ln(U_2). \tag{5.3.15}$$

Besides the subformulae (5.3.7) the following subformulae are introduced:

$$\left.\begin{aligned} A_6 &= \partial U_2/\partial x, \\ A_7 &= \sin(A_6), \\ A_8 &= U_2, \end{aligned}\right|$$

$$\left.\begin{aligned} A_9 &= \ln(A_8)\ , \\ H_2 = A_{10} &= A_7 + A_9\ , \\ \Phi_2^6 = A_{11} &= \cos(A_6)\ , \\ \Psi_1^6 = A_{12} &= A_{11}, \\ A_{13} &= -1\ , \\ \Psi_2^6 = A_{14} &= A_{13} \times A_{17}, \\ A_{15} &= 1\ , \\ A_{16} &= A_8\ , \\ \Psi_1^8 = A_{17} &= A_{15}/A_{16}. \end{aligned}\right\} \qquad (5.3.16)$$

The zeroth Taylor coefficients may be calculated by:

$$\begin{aligned} a_{6,0,0} &:= u_{2,1,0}; \\ a_{7,0,0} &:= \sin(a_{6,0,0}); \\ a_{8,0,0} &:= u_{2,0,0}; \\ a_{9,0,0} &:= \ln(a_{8,0,0}); \\ h_{2,0,0} := a_{10,0,0} &:= a_{7,0,0} + a_{9,0,0}; \\ \phi^6_{2,0,0} := a_{11,0,0} &:= \cos(a_{6,0,0}); \\ \psi^6_{1,0,0} := a_{12,0,0} &:= a_{11,0,0}; \\ a_{13,0,0} &:= -1; \\ \psi^6_{2,0,0} := a_{14,0,0} &:= a_{13,0,0} \times a_{7,0,0}; \\ a_{15,0,0} &:= 1; \end{aligned}$$

$a_{16,0,0} := a_{8,0,0};$

$\psi^{8}_{1,0,0} := a_{17,0,0} := a_{15,0,0}/a_{16,0,0}.$

From the calculated value of $h_{2,0,0}$ the value of $u_{2,0,1}$ can be calculated:

$u_{2,0,1} := h_{2,0,0}.$

The (n,m) Taylor coefficients are defined by:

$a_{6,n,m} := (n+1)u_{2,n+1,m};$

$a_{7,n,m} := (\sum_{p=0}^{n} \sum_{q=0}^{m} a_{12,p,q}(n-p)a_{6,n-p,m-q})/n;$

(if n = 0, we have to use formula (5.3.13))

$a_{8,n,m} := u_{2,n,m};$

$a_{9,n,m} := (\sum_{p=0}^{n} \sum_{q=0}^{m} a_{17,p,q}(n-p)a_{8,n-p,m-q})/n;$

(if n = 0, we have to use formula (5.3.13))

$h_{2,n,m} := a_{10,n,m} := a_{7,n,m} + a_{9,n,m};$

$\phi^{6}_{1,n,m} := a_{11,n,m} := (\sum_{p=0}^{n} \sum_{q=0}^{m} a_{14,p,q}(n-p)a_{6,n-p,m-q})/n;$

(if n = 0, we have to use formula (5.3.13'))

$\psi^{6}_{1,n,m} := a_{12,n,m} := a_{11,n,m};$

$a_{13,n,m} := 0;$

$\psi^{6}_{2,n,m} := a_{14,n,m} := \sum_{p=0}^{n} \sum_{q=0}^{m} a_{13,p,q}\ a_{17,p,q};$

$a_{15,n,m} := 0;$

$a_{16,n,m} := a_{8,n,m};$

$$\psi^8_{1,n,m} := a_{17,n,m} := (a_{15,n,m} - \sum_{\substack{p=0\\p+q\neq n+m}}^{n} \sum_{q=0}^{m} a_{17,p,q}\, a_{16,n-p,m-q})/a_{16,0,0}.$$

We see that $h_{2,n,m}$ is calculated in terms of: $u_{2,n+1,m}$, $u_{2,n+1,m-1}$, ..., $u_{2,n+1,0}$, $u_{2,n,m}$, ..., $u_{2,0,0}$. (These quantities appear in the sums for defining $a_{7,n,m}$ and $a_{9,n,m}$.) Thus, according to section 5.2, $h_{2,n,m}$ is indeed computable.

From $h_{2,n,m}$, the $u_{2,n,m+1}$ may be calculated by means of

$$u_{2,n,m+1} := h_{2,n,m}/(m+1).$$

Remarks

1. It seems that for the calculation of $a_{11,n,m}$ the value of the not yet calculated $a_{14,n,m}$ is needed; this is, however, not the case since $a_{14,n,m}$ is multiplied by 0.
2. For $n \neq 0$ and $m \neq 0$ we may use two different formulae (5.3.12) and (5.3.13) to calculate the coefficient of $\Phi(A_j)$. The results are, however, identical. This can be seen easily. We shall only treat the case $A_i = \exp(A_j)$. Equation (5.3.12) is equivalent with

$$\frac{\partial A_i}{\partial x} = A_i \frac{\partial A_j}{\partial x}$$

which has the solution $A^1_i = C_1(y)\exp(A_j)$.
In the same way equation (5.3.13) is equivalent with $A^2_i = C_2(x)\exp(A_j)$.
For $n = 0$, only equation (5.3.13) can be used, this means for $x = 0$: A^1_i and A^2_i are equal. The same reasoning gives that for $y = 0$: A^1_i and A^2_i are equal; thus $C_1(y) = C_2(x) =$ constant. For $n = 0$ and $m = 0$ we have $A_i(0,0) = \exp(A_j(0,0))$;thus,the constant is unity.

Integral powers:$A_i = \Phi(A_j) = A_j \uparrow n$, where n is an integer,can not be treated by means of the differential equation:

$$\frac{d\Phi}{dz} = n \times \Phi/z,$$

since the constant term of the Taylor series for z may be zero.
We, therefore, treat $A_j \uparrow n$ by means of an efficiently formed product: let $A_p = A_j \uparrow (n \div 2)$, then

$$A_i = (A_p \times A_p) \times (\underline{\text{if}}\ n \div 2 \times 2 = n\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ A_j).$$

5.3.4. A sketch of an ALGOL 60 procedure

The process, as described in the previous subsections, can be defined by a simple ALGOL 60 program. The functions H_k are stored as trees to be used as the threads along which a recursive procedure

can perform the calculations. Let the index i of the subformula $f = A_i$ be called the *Taylor index* of *f*. Introduce a nonlocal variable *next Taylor index* which initially gets the value 0; introduce, moreover, the array elements *a*[*i*,*n*,*m*]; then, a sketch of the procedure for calculating the *a*[*i*,*n*,*m*] might be

```
integer procedure Taylor index (f); value f; integer f;
begin integer type, A, B, T, TA, TB, p, q;
      type:= TYPE(f,A,B);

      if type = sum ∨ type = difference ∨ type = product ∨ type = quotient
      then begin TA:= Taylor index(A); TB:= Taylor index(B) end else
      if type = function ∨ type = integral power
      then TB:= Taylor index(B) else EXIT;

      T:= Taylor index:= next Taylor index:= next Taylor index + 1;

      a[T,n,m]:=
        if type = sum then a[TA,n,m] + a[TB,n,m] else
        if type = difference then a[TA,n,m] - a[TB,n,m] else
        if type = product then
           SUM(p,1,n,SUM(q,1,m,a[TA,p,q] × a[TB,n-p,m-q]))
           else
           ............... else
        if type = unknown function then u[index of unknown function(f),n,m]
           else ..........
end Taylor index;
```

If *H*[*k*] is the formula H_k, then the Taylor coefficients *u*[*k*,*n*,*m*+1] may be calculated by means of:

```
n:= m:= 0;
comment the values of u[k,i,0], i = 0, 1, ..., highest degree,
should be given beforehand;
for next Taylor index:= 0 while n < highest degree do
begin for k:= 1 step 1 until K do
      u[k,n,m+1]:= a[Taylor index(H[k]),n,m]/(m+1);
      n:= n - 1; m:= m + 1;
      if n = -1 then begin n:= m; m:= 0 end
end;
```

Note that if also elementary functions occur, then the Taylor coefficients for the Φ and Ψ functions should also be calculated. We shall discuss this and other details in section 5.7.

5.4. A simple calculation process for the implicit Cauchy problem

We return to the implicit Cauchy problem (5.1.1), which, for the present section and section 5.5, will have the simple form of an ordinary differential equation:

$$G(x,U,U') = 0, \qquad (5.4.1)$$

(where $U' = dU/dx$)

with initial conditions:

$$U(0) = u_0, \; U'(0) = u_1. \qquad (5.4.2)$$

We require, for this section and section 5.5, that the only arithmetic operators occurring in "G" are the operators + and ×; furthermore, "G" does not contain function symbols or integral powers.

The investigations will be illustrated by means of the following example:

$$G = (((U \times U) + (U' \times U')) + (-1)) = 0, \qquad (5.4.3)$$

with

$$U(0) = 0 \text{ and } U'(0) = 1; \qquad (5.4.4)$$

thus, $U = \sin(x)$.

The formula "G" determines a set of subformulae:

$$\begin{aligned} A_1 &= U, \\ A_2 &= U, \\ A_3 &= A_1 \times A_2, \\ A_4 &= U', \\ A_5 &= U', \qquad\qquad (5.4.5) \\ A_6 &= A_4 \times A_5, \\ A_7 &= A_3 + A_6, \end{aligned}$$

$$A_8 = -1,$$

$$A_9 = A_7 + A_8.$$

Table 1 of section 5.3 applied to these subformulae would give the following calculation scheme:

$$\begin{aligned}
&a_{1,n} := u_n;\\
&a_{2,n} := u_n;\\
&a_{3,n} := \sum_{p=0}^{n} a_{1,p} \times a_{2,n-p};\\
&a_{4,n} := (n+1)u_{n+1};\\
&a_{5,n} := (n+1)u_{n+1}; \qquad\qquad (5.4.6)\\
&a_{6,n} := \sum_{p=0}^{n} a_{4,p} \times a_{5,n-p};\\
&a_{7,n} := a_{3,n} + a_{4,n};\\
&a_{8,n} := \text{if } n = 0 \text{ then } -1, \text{otherwise } 0;\\
&a_{9,n} := a_{7,n} + a_{8,n}.
\end{aligned}$$

The process ends, however, after the $a_{i,0}$ have been calculated; as, for the calculation of $a_{4,1}$,we need the unknown coefficient u_2.

<u>We now make the important observation that the coefficient u_2 may be considered as an algebraic variable and that the coefficients $a_{i,1}$ may be calculated in terms of u_2 in an algebraic way</u>.

In particular,$a_{9,1}$ may be calculated as a formula dependent on u_2. As follows from (5.4.1),$a_{9,1} = 0$; hence,we have found an equation for the unknown u_2. Since this equation is linear in u_2, we can calculate u_2 in a simple way:

Let $a_{9,1} = f(u_2)$ then

$$u_2 := - f(0)/(\frac{df}{du_2}).$$

The two operations: substituting $u_2 = 0$ in $f(u_2)$ and differentiating $f(u_2)$ with respect to u_2 are already available in the general system of chapter 2. We are thus led, in a natural way, to perform the calculation by means of formula manipulations.

If u_2 has been calculated, its value should be substituted in the formulae $a_{i,1}$ such that the $a_{i,1}$ become numbers; next, the $a_{i,2}$ may be calculated as formulae in u_3 which may then be calculated by equating $a_{9,2}$ to zero, etc.

The procedure for calculating the $a_{i,n}$ can be made very similar to the procedure *Taylor index* of section 5.3.4; the only significant difference is that the arithmetic operators + and × should be replaced by the operators *S* and *P*. For example, "*a[TA, n, m] + a[TB, n, m]*" should be replaced by "*S(a[TA, n], a[TB, n])*".

We now give a complete ALGOL 60 program, which should be imbedded as an "actual program" in the general system of chapter 2.
This program calculates the first 8 Taylor coefficients of the function U defined by (5.4.3) and (5.4.4). The main part of the program is the procedure *CALC TAYLOR COEFFICIENTS* which is independent of the particular problem; it consists of:

1. The procedure *count number of Taylor indexes* which is used to calculate the length of the array *a*.
2. The procedure *Taylor index*.
3. The procedure body defining the calculation of u_{n+1} and for substituting the calculated value of u_{n+1} in the $a_{i,n}$.

Remark: in this program and in the programs of the next sections, a difference a - b is transformed into a + (-1) × b. This is automatically executed after a redefinition of the procedure *D*. The efficiency of the computational program of section 5.8 is not affected by this transformation.

```
ACTUAL PROGRAM: INITIALIZE; expand:= false;
   begin integer U,Uprime,x,i; integer array u[0:8];
      procedure OUTPUT(f); value f; integer f;
      begin integer t,a,b; t:= TYPE(f,a,b);
         if t = number ∧ a = integer then
         begin VAL OF INT NUM(f,a); FIXP(2,0,a) end
         else if t = number ∧ a = real then
         begin real r; VAL OF REAL NUM(f,r);
            FLOP(12,3,r)
         end else PUTEXT(⌈wrong formula⌉)
      end OUTPUT;
      integer procedure D(a,b); value a,b; integer a,b;
      D:= S(a,P(minone,b));

      procedure CALC TAYLOR COEFFICIENTS
         (diff eq,u0,u1,highest degree,u);
         value diff eq,u0,u1,highest degree;
         integer diff eq,u0,u1,highest degree; integer array u;
      begin integer next Taylor index,i,k,n,unplus1,eq in unplus1;
         integer procedure count number of Taylor indexes(f);
            value f; integer f;
         begin integer t,A,B; t:= TYPE(f,A,B);
            count number of Taylor indexes:=
               if t = sum ∨ t = product then
               1 + count number of Taylor indexes(A) +
                   count number of Taylor indexes(B)
               else 1
         end count number of Taylor indexes;

         begin integer array a[1:count number of Taylor indexes (diff eq),
               0:highest degree - 1];

            integer procedure Taylor index(f); value f; integer f;
            begin integer t,A,B,T,TA,TB,p; t:= TYPE(f,A,B);
            if t = sum ∨ t = product then
            begin TA:= Taylor index(A); TB:= Taylor index(B) end;
            T:= Taylor index:= next Taylor index:= next Taylor index + 1;
            a[T,n]:= if t = product then
               Sum(p,0,n,P(a[TA,p],a[TB,n-p])) else
            if t = sum then
               S(a[TA,n],a[TB,n]) else
            if f = U then u[n] else
            if f = Uprime then (if n = 0 then u[1] else
               P(IN(n + 1),unplus1)) else
            if f = x then (if n = 1 then one else zero) else
            (if n = 0 then f else zero)
         end Taylor index;

      BODY OF CALC TAYLOR COEFFICIENTS:
         unplus1:= STORE(0,algebraic variable,0);
         u[0]:= u0; u[1]:= u1;
         for n:= 0 step 1 until highest degree - 1 do
         begin next Taylor index:= 0; eq in unplus1:=
            a[Taylor index(diff eq),n];
            if n = 0 then goto out;
            u[n+1]:= Q(P(minone,SUBSTITUTE(eq in unplus1,
            k,1,1,unplus1,zero)),DER(eq in unplus1,unplus1));
            for i:= 1 step 1 until next Taylor index do
               a[i,n]:= SUBSTITUTE(a[i,n],k,1,1,unplus1,u[n+1]);
         out: end
   end end CALC TAYLOR COEFFICIENTS;

   U:= STORE(0,algebraic variable,0);
   Uprime:= STORE(0,algebraic variable,0);
   x:= STORE(0,algebraic variable,0);
   CALC TAYLOR COEFFICIENTS(
      S(S(P(U,U),P(Uprime,Uprime)),minone),
      zero,one,8,u);
```

```
        for i:= 0,1,2,3,4,5,6,7,8 do
        begin PUNLCR; OUTPUT(u[i]) end
    end;comment the next two ends correspond to the two begins of the
    general system;
end end
The input tape consists of
    2048     500       0       0       0
    ₁₀-10    ₁₀-10
```

The above program, which gives the following results:

```
            0
            1
            0
            -.166666666667₁₀-0
            0
            +.833333333335₁₀-2
            0
            -.198412698410₁₀-3
            0,
```

is very simple and straightforward,but has four shortcomings:

1. The program itself (together with the, not reproduced, general system) needs much storage space, which cannot be used for storing the numerous Taylor coefficients in a realistic problem.
2. The program needs much storage space for storing the Taylor coefficients as formulae. (An approach in which the Taylor coefficients are stored as real numbers is described in [15]).
3. The program uses the storage space for the Taylor coefficients very inefficiently (it will turn out,later on,that the same storage space may be used for different Taylor coefficients).
4. The program needs much time, since the algebraic analysis for calculating the Taylor coefficients is performed repeatedly.

5.5. Divide et impera

The process, as described in the foregoing section, can be improved considerably by splitting it into two programs:

1. The algebraic program, which has as input the differential equations and which defines output in the form of ALGOL 60 statements to be used in
2. the computational program, which has as input the initial values and computes in an ordinary arithmetic way the Taylor coefficients.

In order to get an idea of the tasks for the algebraic program we shall propose some forms of the computational program.

The heading of the computational program should consist of:
the declaration of the real-array elements *a[i,n]* and the declaration

```
real procedure CONV PRODUCT(i,j,p,q); value i, j, p, q;
integer i, j, p, q;
comment
                       n-q
        CONV PRODUCT:=  ∑  a[i,n-k] × a[j,k].
                       k=p
        Note that if p = 1, then a[i,n] × a[j,0] is not included
              and if q = 1, then a[i,0] × a[j,n] is not included.;
begin integer k; CONV PRODUCT:= SUM(k,p,n-q,a[i,n-k] × a[j,k])
end;
```

Next,the computational program should contain the ALGOL 60 statement for calculating the zeroth Taylor coefficients.
Finally, it should contain,separately,a set of ALGOL 60 statements for calculating the nonzeroth Taylor coefficients. This separation is necessary if we want to introduce elementary functions, later on.

It is a simple matter to change the procedure *Taylor index* of section 5.4, such that it defines the desired output.

Before doing this, we shall declare some procedures for outputting purposes. The aim is to use concatenated strings as actual procedure parameters. To that purpose,the following procedures are declared of integral type.

```
integer procedure PR(s); string s;
begin PUTEXT(s); PR:= 1 end;
integer procedure PR integral number(i); value i; integer i;
begin integer n; n:= if i = 0 then 1 else
      entier(ln(abs(i)) × .4343 + 1);
      if i ≥ 0 then ABSFIXP(n,0,i) else FIXP(n,0,i);
      PR integral number:= 1
end PR integral number;
integer procedure PR nlcr; begin PUNLCR; PR nlcr:= 1 end;

integer procedure PR array element(T); value T; integer T;
begin PR(⟨a[⟩); PR integral number(T); PR(⟨,n]⟩);
      PR array element:= 1
end PR array element;

integer procedure PR zeroth array element(T); value T; integer T;
begin PR(⟨a[⟩); PR integral number(T); PR(⟨,0]⟩);
      PR zeroth array element:= 1
end PR zeroth array element;
```

In the declaration of the next output procedures, we have deliberately used the fact that the X8 ALGOL 60 system evaluates an expression from left to right; so that, e.g., the effect of the statement:

x:= PR array element(5) × PR(⟨:= ⟩) × PR array element(10);

is, that *x* gets the (uninteresting) value *1* and that the following (interesting) output is produced:

a[5,n]:= a[10,n].

It is admitted that this way of concatenating strings is far from decent; a more neat, but also more laborious way would be to declare the following procedure *conc*:

```
integer procedure conc(head, tail); value head;
   integer head, tail; conc:= tail;
```

and to change the ""×"-concatenated string":

$$S_1 \times S_2 \times \ldots \times S_{n-1} \times S_n$$

where S_i, i = 1, ..., n, may stand for a call of any output procedure, declared above, into the following "decently-concatenated string":

$$conc(s_1, conc(s_2, \ldots, conc(s_{n-1}, s_n) \ldots)).$$

The above statement would then read:

```
x:= conc(PR array element(5), conc(PR(⦉:= ⦊),
         PR array element(10)));
```

If, in the sequel, we use the more elegant ""×"-concatenated string", we shall indicate such by a note and refer to the remarks on these pages.

We now define some more output procedures:

```
integer procedure PR ass st for arr el(T,right hand side);
value T; integer T, right hand side;
      PR ass st for arr el:= PR nlcr × PR array element(T) ×
      PR(⦉:=⦊) × right hand side × PR(⦉;⦊);                    *)

integer procedure PR conv product(a,b,c,d); value a,b,c,d;
   integer a,b,c,d;
PR conv product:= PR(⦉CONV PRODUCT(⦊) × PR integral number(a) ×
   PR(⦉,⦊) × PR integral number(b) × PR(⦉,⦊) ×                  *)
   PR integral number(c) × PR(⦉,⦊) × PR integral number(d) ×
   PR(⦉)⦊);
```

*) See the above remarks.

We now give the new procedure *Taylor index*.

```
integer procedure Taylor index(f); value f; integer f;
begin integer t, A, B, T, TA, TB; t:= TYPE(f,A,B);
   if t = sum ∨ t = product then
   begin TA:= Taylor index(A); TB:= Taylor index(B) end;
   T:= Taylor index:= next Taylor index:= next Taylor index +1;
   PR ass st for arr el(T,                                          *)
   if t = sum then PR array element(TA) × PR(⌈+⌉) ×
                   PR array element(TB) else
   if t = product then PR conv product(TA,TB,0,0) else
   if f = U then PR(⌈u[n]⌉) else
   if f = U prime then PR(⌈(n+1) × u[n+1]⌉) else
   if f = x then PR(⌈COEFF OF X⌉) else
   if t = number then PR(⌈NUMBER(⌉) × OUTPUT(f) × PR(⌈)⌉)
   else 1)
end Taylor index;
```

Note first, that the procedure *OUTPUT* should be of integral type and second, that the computational program should also be provided with procedures *NUMBER* and *COEFF OF X* with obvious meanings.

This procedure *Taylor index* when applied to the example (5.4.3) gives the following output:

```
a[1,n]:= u[n];                                   ⎫
a[2,n]:= u[n];                                   ⎪
a[3,n]:= CONV PRODUCT(1,2,0,0);                  ⎪
a[4,n]:= (n+1) × u[n+1];                         ⎪
a[5,n]:= (n+1) × u[n+1];                         ⎬          (5.5.1)
a[6,n]:= CONV PRODUCT(4,5,0,0);                  ⎪
a[7,n]:= a[3,n] + a[6,n];                        ⎪
a[8,n]:= NUMBER(-1);                             ⎪
a[9,n]:= a[7,n] + a[8,n];                        ⎭
```

*) See the remarks on pp. 59-60.

For $n = 0$ the above statements define $a[i,0]$.

For $n > 0$, the following calculation process is possible:

1. make $u[n+1]$ equal to zero;
2. calculate all $a[i,n]$;
3. equate α to $a[9,n]$;
4. make $u[n+1]$ equal to one;
5. calculate all $a[i,n]$;
6. equate β to $a[9,n]$;
7. calculate $u[n+1]$ by means of $u[n+1] := \alpha/(\alpha-\beta)$;
8. calculate all $a[i,n]$.

Although very simple, the above process is also very inefficient from the standpoint of computing time: in a real problem with K unknown functions the $a[i,n]$ should be calculated K(K+1)+1 times.

There is a more efficient, but also more difficult, way to perform the calculations by means of "more sophisticated" ALGOL 60 statements.

As in section 5.4 we treat the quantity $u[n+1]$ as an unknown algebraic variable and separate the subformulae A_i into two classes: C_d for the subformulae dependent on U' and C_{ind} for the subformulae A_i independent on U'. Three sorts of statements are outputted: first, statements defining the Taylor coefficients $a[i,n]$, for which $A_i \in C_{ind}$; second, statements defining special array elements $coeff[j]$ which originate from expressions for Taylor coefficients $a[i,n]$ for which $A_i \in C_d$, and which are used to calculate $u[n+1]$ immediately; third, statements for defining the Taylor coefficients $a[i,n]$, with $A_i \in C_d$, in terms of the already calculated $u[n+1]$.

Let us first give this output and then discuss how the algebraic program should be made to produce this output.

```
a[1,n]:= u[n];
a[2,n]:= u[n];
a[3,n]:= CONV PRODUCT(1,2,0,0);
coeff[1]:= CONV PRODUCT(4,5,1,1);
coeff[2]:= a[5,0];
coeff[3]:= a[4,0];
coeff[4]:= a[3,n];
```

```
a[8,n]:= NUMBER(-1);
coeff[5]:= a[8,n];

COEFF[0]:= -((coeff[4] + coeff[1]) + coeff[5]);
COEFF[1]:= (coeff[2] + coeff[3]);

JUMP TO CALC OF UNKNOWNS;
FILL IN OPEN PLACES:

a[4,n]:= (n+1) × u[n+1];                                          (5.5.2)
a[5,n]:= a[4,n];
a[6,n]:= (coeff[1] + ((a[5,n] × coeff[2]) + (coeff[3] × a[5,n])));
a[7,n]:= (coeff[4] + a[6,n]);
a[9,n]:= (a[7,n] + coeff[5]);
```

As one sees immediately, the coefficients *coeff*[*i*] are defined in terms of already known quantities; for example, *coeff*[*1*], *coeff*[*3*] and *coeff*[*2*] originate from the formula:

$$a_{6,n} = \sum_{p=1}^{n-1} a_{4,p}a_{5,n-p} + a_{4,0}a_{5,n} + a_{4,n}a_{5,0},$$

in which the $a_{4,n}$ and $a_{5,n}$ are unknown.
Therefore, the coefficients *COEFF*[*0*] and *COEFF*[*1*] are calculatable; it is easily seen that *u*[*n+1*] can be calculated by means of

u[*n+1*]:= *COEFF*[*0*]/*COEFF*[*1*]/(*n+1*);

this statement should occur within the body of the procedure *JUMP TO CALC OF UNKNOWNS*. After completion of the procedure call *JUMP TO CALC OF UNKNOWNS*, the program continues with the calculation of the Taylor coefficients *a*[*i*,*n*], with $A_i \epsilon C_d$.
We shall now show how the algebraic program can be made such that it defines the above statements.

First, we introduce the procedure *COEFF* which defines output of the form "*coeff*[...]:= ...;".

```
integer procedure  COEFF(right hand side); integer right hand side;
begin integer x; pointer of coeff:= pointer of coeff +1;
   PR nlcr; PR(⌈coeff[⌉); PR integral number(pointer of coeff);
   PR(⌈]:= ⌉); x:= right hand side;
   PR(⌈;⌉);
   COEFF:= STORE(coefficient,algebraic variable,pointer of coeff)
end COEFF;
```

The *pointer of coeff* should have got the initial value 0. The last statement of the procedure body is very important; here *COEFF* becomes an algebraic variable of the type *coefficient*.

If later on, the output procedure hits upon an algebraic variable whose lhs is equal to *coefficient* and whose rhs is equal to e.g. 10, then it can produce the output: "*coeff*[*10*]".

Next, we introduce the integer-array elements *formula for* [*i*].

If $A_i \in C_{\mathrm{ind}}$, then the program makes *formula for* [*i*] equal to *-1*; if, on the other hand, $A_i \in C_{\mathrm{d}}$, it builds up a formula by means of the procedure *COEFF* and assigns this formula to *formula for* [*i*] (note that *-1* can not be the value of a formula).

Finally, we introduce the algebraic variable *unknown* by means of

```
            unknown:= STORE(Unknown,algebraic variable,0);
```

If the output procedure comes across this algebraic variable, it has to produce the output "*(n+1)* × *u*[*n+1*]".

Now, we are ready to construct the new procedure

```
integer procedure Taylor index(f); value f; integer f;
begin integer t, A, B, T, TA, TB; t:= TYPE(f,A,B);
   if t = sum ∨ t = product then
   begin TA:= Taylor index(A); TB:= Taylor index(B) end;
   T:= Taylor index:= next Taylor index:= next Taylor index +1;
```

```
    if t = sum then
    begin if formula for [TA]= -1 ∧ formula for [TB] = -1 then
          begin formula for [T]:= -1;
                PR ass st for arr el(T,PR array element(TA) × PR(‹+›) × *)
                PR array element(TB))
          end else
          if formula for [TA] = -1 then formula for [T]:=
                S(COEFF(PR array element(TA)),formula for [TB])
          else
          if formula for [TB] = -1 then formula for [T]:=
                S(formula for [TA],COEFF(PR array element(TB)))
          else
          formula for [T]:= S(formula for [TA],formula for [TB])
    end else
    if t = product then
    begin if formula for [TA] = -1 ∧ formula for [TB] = -1 then
       begin formula for [T]:= -1;
          PR ass st for arr el(T,PR conv product(TA,TB,0,0))
       end else
       if formula for [TA] = -1 then formula for [T]:=
          S(COEFF(PR conv product(TA,TB,0,1)),
             P(COEFF(PR zeroth array element(TA)),formula for [TB]))
          else
       if formula for [TB] = -1 then formula for [T]:=
          S(COEFF(PR conv product(TA,TB,1,0)),
             P(formula for [TA],COEFF(PR zeroth array element(TB))))
          else
       formula for [T]:= S(COEFF(PR conv product(TA,TB,1,1)),
          S(P(formula for [TA],COEFF(PR zeroth array element(TB))),
             P(COEFF(PR zeroth array element(TA)),formula for [TB])))
end else
if f = U then
begin formula for [T]:= -1;
      PR ass st for arr el(T,PR(‹u[n]›))
end else
```

*) See the remarks on pp. 59-60.

```
   if f = U prime then
   begin if n = 0 then
         begin formula for [T]:= -1;
               PR ass st for arr el(T,PR(⊀u[1]⊁))
         end else formula for [T]:= unknown
   end else
   if f = x then
   begin formula for [T]:= -1;
         PR ass st for arr el(T,PR(⊀COEFF OF X⊁))
   end else
   if t = number then
   begin formula for [T]:= -1;                          *)
         PR ass st for arr el(T,PR(⊀NUMBER(⊁) ×
            OUTPUT(f) × PR(⊀)⊁))
   end
end Taylor index;
```

The procedure body of *CALC TAYLOR COEFFICIENTS* (see section 5.4) may now be replaced by:

```
BODY OF CALC TAYLOR COEFFICIENTS:
   unknown:= STORE(Unknown,algebraic variable,0);
   n:= 0;
AGAIN: next Taylor index:= pointer of coeff:= 0;
   T:= Taylor index(diff eq);
   if n = 0 then begin n:= 1; goto AGAIN end; PR nlcr;
   PR nlcr; PR(⊀COEFF[0]:= -⊁);
   OUTPUT(SUBSTITUTE(formula for [T],k,1,1,unknown,zero));
   PR(⊀;⊁);
   PR nlcr; PR(⊀COEFF[1]:= ⊁);
   OUTPUT(DER(formula for [T],unknown)); PR(⊀;⊁); PR nlcr;
   PR nlcr; PR(⊀JUMP TO CALC OF UNKNOWNS;⊁);
   PR nlcr; PR(⊀FILL IN OPEN PLACES:⊁);
   for T:= 1 step 1 until next Taylor index do
   begin if formula for [T] ≠ -1 then
```

*) See the remarks on pp. 59-60.

```
            begin PR ass st for arr el(T,OUTPUT(formula for [T]));
               REPLACE(formula for [T],
                  STORE(Taylor coefficient,algebraic variable,T))
            end
      end
end end CALC TAYLOR COEFFICIENTS;
```

Within this context we observe that the procedure *REPLACE*, declared in the general system (section 2.6), is very useful; since by means of *REPLACE* we are able to change the internal representation of *formula for* [*T*]. That this changing is necessary may be seen from the preceding example (5.5.2) where we have:

```
   formula for [4] = unknown,
   formula for [5] = unknown,
   formula for [6] = coeff[1] + ((formula for [4] × coeff[2])
                              + (coeff[3] × formula for [5])),
   formula for [7] = coeff[4] + formula for [6], and
   formula for [8] = formula for [7] + coeff[5].
```

Without changing the internal representation of *formula for* [*T*], the following output would be produced:

```
   a[4,n]:= (n+1) × u[n+1];
   a[5,n]:= (n+1) × u[n+1];
   a[6,n]:= (coeff[1] + (((n+1) × u[n+1]×coeff[2] +
                  (coeff[3] × (n+1) × u[n+1])));
   a[7,n]:= (coeff[4] + (coeff[1] + (((n+1) × u[n+1]×coeff[2] +
                  (coeff[3] × (n+1) × u[n+1]))));
   a[9,n]:= ((coeff[4] + (coeff[1] + (((n+1) × u[n+1]×coeff[2] +
                  (coeff[3] × (n+1) × u[n+1])))) + coeff[5]);
```

The new procedure *CALC TAYLOR COEFFICIENTS*, when applied to the differential equation (5.4.1), will define output as given by (5.5.1) followed by (5.5.2); the expressions "*(n+1)* × *u*[*n+1*]"occurring in (5.5.1) are output as "*u*[*1*]".

5.6. Saving storage space

5.6.1. Introduction

The intelligence of the algebraic program can be enlarged in several directions:

1. Detection of common subformulae.
   The formulae U and U, and U' and U' in:

   $$(((U \times U) + (U' \times U')) + (-1)) = 0 \qquad (5.4.3)$$

   are identical; the procedures *Taylor index* of the preceding sections do not recognize this fact and treat both U's and both U''s separately; introducing thus two superfluous Taylor series. Detection of common subformulae and a suitable treatment will lead to computational programs which are more efficient with respect to computation time and storage space.

2. Detection of equivalent subformulae.
   Using the simplification routines of the second chapter, it is possible to detect equivalent subformulae; such as x+y and y+x. Which may be treated once also; thus reducing storage space and computation time. We encounter, however, some difficult problems:
   a) how far should the simplification be done;
   b) in simplifying a formula, the procedures of chapter 2 require the formulae to be stored in expanded form; but expanding a formula will mostly lead to a more lengthy formula with more products (involving expensive convolution products) in it; the algebraic program should detect this situation.

3. Detection of the fact that array elements may be used more than once.
   Returning to the statements (5.5.1) and (5.5.2), we see that after *a*[*3*,*n*] has been used in evaluating *a*[*7*,*n*] or *coeff*[*4*], the array element *a*[*3*,*n*] can be used again for storing another Taylor coefficient. In the same way we see that *a*[*6*,*n*], *a*[*7*,*n*] and *a*[*8*,*n*] can be used again after they have been used in evaluating *a*[*7*,*n*] and *a*[*9*,*n*].

Moreover, it is not necessary to introduce the array elements *u*[*n*], since the Taylor coefficients of U are already stored in *a*[*1*,*n*], *n* = 0, 1, 2, ...
It is thus possible to change the statements (5.5.1) into:

```
    a[2,n]:= CONV PRODUCT(1,1,0,0);
    a[3,n]:= a[1,1];
    a[4,n]:= a[1,1];
    a[5,n]:= CONV PRODUCT(3,4,0,0);              (5.6.1a)
    a[5,n]:= a[2,n] + a[5,n];
    a[2,n]:= NUMBER(-1);
    a[5,n]:= a[5,n] + a[2,n];
```

which, for *n* = 0, deliver a value for *a*[*5*,*n*] equal to the value of *a*[*9*,*n*] in (5.5.1).
And it is possible to change the statements (5.5.2) into:

```
    a[2,n]:= CONV PRODUCT(1,1,0,0);
    coeff[1]:= CONV PRODUCT(3,4,1,1);
    coeff[2]:= a[4,0];
    coeff[3]:= a[3,0];
    coeff[4]:= a[2,n];
    a[2,n]:= NUMBER(-1);
    coeff[5]:= a[2,n];                                              (5.6.1b)

    COEFF[0]:= -((coeff[4] + coeff[1]) + coeff[5]);
    COEFF[1]:= (coeff[2] + coeff[3]);

    JUMP TO CALC OF UNKNOWNS;
    FILL IN OPEN PLACES:

    a[3,n]:= (n+1) × a[1,n+1];
    a[4,n]:= a[3,n];
```

which deliver values for *COEFF*[*0*] and *COEFF*[*1*] equal to the values they got by the statements (5.5.2). Hence, the calculated value of u_{n+1} is also the same.

Note, that the statements defining *a[6,n]*, *a[7,n]* and *a[9,n]* in (5.5.2) have been skipped, since the values of the corresponding Taylor coefficients have become useless.

It is a rather simple matter to construct a new procedure *Taylor index* which defines the output sketched above. The only significant difference with the old procedure *Taylor index* is that the determination of the Taylor index, i.e. the index i in *a*[i,*n*], cannot be performed by the simple statement:
Taylor index:= next Taylor index:= next Taylor index + 1;
but has to be performed by a more complicated mechanism using a set of Taylor indexes; a new Taylor index is taken from this set, while a Taylor index, i, is added to this set when it turns out that *a*[i,*n*] may be used again for storing another Taylor coefficient, as *a[2,n]* and *a[5,n]* above.

4. Transformation of the differential equations.
Employing the strategy as sketched under point 3, we even can go a step further and construct the algebraic program such that it transforms the differential equations into mathematically equivalent differential equations which are such that a minimum of array elements *a*[i,*n*] are involved.
For example, equation (5.6.1) involves *a*[i,*n*], with i = 1, ..., 5, whereas

$$((U \times U) + ((U' \times U') + (-1))) = 0$$

involves *a*[i,*n*], with i = 1, ..., 6, as can easily be seen.
It will be shown in section 5.6.3 that it is possible to prove rigorously that a special form of the differential equations involves a minimal number of array elements.

Above, we have sketched four directions in which it is possible to enlarge the intelligence of the algebraic program. We have chosen for the latter two directions instead of the first two or a combination of all four.

The, possibly not completely evident, reasons are:

a) If there is a laborious subformula occurring at a number of places in the differential equations, then the user has the means (by introducing an extra dependent variable, see section 5.10) to make full profit of this situation and still use our algebraic program.

b) Apart from the problems mentioned at the end of point 2, it would not be possible to combine the simplifying procedures of chapter 2 and the algebraic program into one ALGOL 60 program which is still manageable by the available MC-X8 computer system; it would be too large.

c) The user cannot, or at least it would be very difficult, perform the storage space optimizations, mentioned under point 3 and 4, by himself; therefore, these optimizations have to be performed automatically and should be built in.

d) A combination of point 1 and points 3 and 4 is difficult, since in this case it will be very hard to prove, if possible anyhow, that some special form of the differential equations involves minimal storage space.

5.6.2. More efficient use of the array elements

Let us investigate precisely when an array element $a[i,n]$ may be used again, or, what is equivalent, when the Taylor coefficient, T, stored into $a[i,n]$ will not be used any more during the calculations.
As far as the calculations with $n > n_0$ are concerned, T will not be used if it does not occur in a convolution product; that means the subformula f of which T is the n_0-th Taylor coefficient is not:

i. a factor in a product,

ii. a quotient,

iii. the numerator of a quotient,

iv. a special function;

in this case f is called free, whereas, if it is of a form mentioned under i, ii, iii or iv, then it is called not free.

A first conclusion is, therefore: if $a[i,n]$ is used to store the Taylor coefficient, T, of a subformula, f, which is not free, then it may not be used again. By the way, we shall call i the <u>Taylor index</u> of f and the set of array elements $a[i,n]$, $n = 0, 1, 2, \ldots$, the <u>Taylor array</u> of f.
We next investigate the possible use of $a[i,n_0]$ during the course of the calculations when $n = n_0$.
Either f is itself the left-hand side of one of the differential equations, in which case $a[i,n_0]$ will not be used during the calculations with $n = n_0$, or f is a direct subformula of some formula g (with "direct" we mean: there is not another subformula of g of which f is a subformula).
As soon as the statements for the Taylor coefficient of g have been produced, the Taylor coefficient of f will not be used again except in the case that f is a direct subformula of other formulae; then, however, we have the case that f occurs as common subformula in several places and then each occurrence of f is treated anew, i.e. each occurrence of f leads to another Taylor series, to another Taylor index and to another Taylor array (under circumstances it may be possible that these Taylor indexes happen to be the same if f is a free formula, but this would be mere chance).
A second conclusion is now that $a[i,n_0]$ is not used during the calculation with $n = n_0$ after the Taylor coefficient is treated of that formula of which f is a direct subformula, where i is the Taylor index of f.

<u>Remarks</u>: 1. If *highest degree* defines the number of wanted Taylor coefficients, then we need for problem (5.6.1) the array elements $a[i,n]$, with $i = 1, \ldots, 5$ and $n = 0, 1, \ldots,$ *highest degree* $- 1$.
A problem then occurs in storing the calculated $u_{highest\ degree}$ into the non-existent $a[1, highest\ degree]$.
This problem is fictituous, since after $u_{highest\ degree}$ has been calculated it can be output and it is not necessary to store this quantity since the computation is finished.

2. After the label: *FILL IN OPEN PLACES* only those array elements *a*[i,*n*] should obtain a value for which the corresponding formula is not free. It would be superfluous, and even wrong, if a value to *a*[i,*n*] were assigned while its corresponding formula were free; since i could have become the Taylor index of a not-free formula. (Consider the example:

$$((x + U_{1x}) + (U_{2x} \times U_{2x})) = 0.)$$

3. Considering the statements (5.6.1a,b), we see that, without damaging the calculations, the last four statements of (5.6.1a) may be skipped, since the values of *a*[*5*,*n*] and *a*[*2*,*n*] are not used in the statements (5.6.1b).
This is a remarkable fact; it follows that *NUMBER(-1)* may be replaced by *NUMBER(*C*)*, where C is an arbitrary constant; and it follows that the "-1" in the differential equation (5.4.3) may be replaced by C, without any effect on the calculated Taylor series of U. The reason is that the constant "-1" has implicitly been used for determining u_0 and u_1, beforehand. Afterwards it does not play any role. One should compare the remarks about the constants C_1 in section 5.1.
It were possible to let the algebraic program detect such a situation; it has not been built in, however, since the profit does not seem to be large for two reasons:
a) If there are more than one differential equations, then the Taylor indexes concerned (2 and 5 above) become free and will be used for a next differential equation.
b) The superfluous statements concern only the statements for the calculation of the zeroth Taylor coefficients; in the calculation of the non-zeroth Taylor coefficients such superfluous statements do not occur.

5.6.3. Efficiency by means of transformations

As already announced in the introduction of section 5.6 we want to transform the differential equation such that the necessary number of Taylor arrays in the computational program is as few as possible.
This is achieved as follows:

1. the constant terms in a sum are, as much as possible, combined into one constant term (constant means:not depending on the variables x and y or the dependent functions);
2. the constant factors in a product are combined into one constant factor;
3. the associative and commutative laws for a sum or a product are applied.

Note that sum and product means here: sum of several terms and product of several factors.
Only the last point deserves special attention.
Consider the example:

$$F = (U' + (x + (U' \times U))).$$

(The numbers: (2, 2, 3,3, 4, 5,1) give the Taylor indexes of the subformulae.)

The number of necessary Taylor arrays is 5, while this number for

$$\overline{F} = ((U' + x) + (U' \times U))$$

$$(2, 2,3, 2, 3, 4,1)$$

is 4; thus, application of the associative law may be advantageous.
Consider next the example:

$$F = ((U' \times x) + (U' + x)).$$

$$(2, 4,3, 4, 5, 5,6)$$

The number of necessary Taylor arrays is 6 (including the Taylor arrays for U itself), while this number for

$$\overline{F} = ((U' + x) + (U' \times x))$$

$$(2, 2,3, 2, 3, 5,4)$$

is 5; thus, application of the commutative law may be advantageous.

In this section some theorems are proved which clear up the situation and by means of which we are able to construct an efficient computational program.

Notations: In order to tie up the form of a formula, we introduce the dyadic, noncommutative adding operator $\oplus$ and the dyadic, noncommutative multiplying operator $\otimes$.

The first two formulae F and $\overline{F}$, above, are then written in the forms:

$$F = U' \otimes (x \oplus (U' \otimes U)),$$

and $$\overline{F} = (U' \oplus x) \oplus (U' \otimes U), \text{ respectively.}$$

The functions $\pi(x)$ and $\beta(B)$ are defined as follows:

$$\pi(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \qquad \text{(x is an integer)},$$

$$\beta(B) = \begin{cases} 1 & \text{if } B \\ 0 & \text{if } \neg B \end{cases} \qquad \text{(B is a Boolean expression)}.$$

Abbreviation: If a formula F is free, we write $F \epsilon fr$;
if a formula F is not free, we write $F \not\epsilon fr$.

5.6.3.1. Commutative laws

Consider $F = A \oplus B$ and $F' = B \oplus A$. We calculate:

1. N_F = number of Taylor arrays, used to calculate the Taylor coefficients of F;
2. R_F = number of Taylor arrays, which may be used again after the calculation of the Taylor coefficients of F(the Taylor array for F itself is not included in this number).

In the same way N_A, N_B, $N_{F'}$, R_A, R_B and $R_{F'}$ are defined.
Let us calculate N_F in terms of N_A and R_A by following the process:

Treatment of:		leads to:
A	:	$N := N_A;\ R := R_A;$
B	:	$N := N + \pi(N_B - R);$
		$R := R_B + \pi(R - N_B);$
		$R := R + \beta(A \epsilon fr) + \beta(B \epsilon fr);$
F	:	$N := N + \beta(R = 0);\ R := \pi(R - 1).$

Thus, finally:

$$\left.\begin{array}{l} N_F = N = N_A + \pi(N_B - R_A) + \beta(\overline{R}_F = 0), \text{ with} \\ \overline{R}_F = R_B + \pi(R_A - N_B) + \beta(A\epsilon fr) + \beta(B\epsilon fr) \text{ and} \\ R_F = \pi(\overline{R}_F - 1) \end{array}\right\} \qquad (5.6.2)$$

<u>Theorem 1</u>. If $F = A \oplus B$, $F' = B \oplus A$ and $R_A \geq R_B$, then $N_F \leq N_{F'}$.

<u>Proof</u>. From $N_A > R_A \geq R_B$ it follows:

$$\pi(N_A - R_B) = N_A - R_B$$

and $\pi(R_B - N_A) = 0$.

Thus ,

$$N_{F'} = N_B + N_A - R_B + \beta(\overline{R}_{F'} = 0),$$

$$\overline{R}_{F'} = R_A + \beta(B\epsilon fr) + \beta(A\epsilon fr),$$

$$R_{F'} = \pi(\overline{R}_{F'} - 1)$$

and

$$N_F - N_{F'} = \pi(N_B - R_A) - (N_B - R_B) + \beta(\overline{R}_F = 0) - \beta(\overline{R}_{F'} = 0). \quad (5.6.3)$$

1. Assume $\overline{R}_{F'} = 0$,then

$$R_A = 0 \wedge B\notin fr \wedge A\notin fr,$$

thus $R_B = 0$ and $N_F - N_{F'} = \beta(\overline{R}_F = 0) - 1 = 0$.

2. Assume $\overline{R}_{F'} > 0$.

2.1. Let $\overline{R}_F = 0$,then

$R_B = 0 \wedge (R_A \leq N_B) \wedge A\notin fr \wedge B\notin fr$,

thus $R_A > 0$.
It now follows: $N_F - N_{F'} = - R_A + 1 \leq 0$.

2.2. Let $\overline{R}_F > 0$.
If $R_A \geq N_B$ then $N_F - N_{F'} < 0$,

if $R_A < N_B$ then $N_F - N_{F'} = R_B - R_A \leq 0$.

From the theorem proved, it follows that it is in general advantageous to transform $B \oplus A$ into $A \oplus B$ if $R_A > R_B$.
It should be remarked that if $N_{B\oplus A} > N_{A\oplus B}$ then $R_{B\oplus A} < R_{A\oplus B}$, thus, if $B \oplus A$ is changed into $A \oplus B$, there remain less Taylor arrays to be used again.
At first sight it is not clear, therefore, that changing $B \oplus A$ into $A \oplus B$ will, in general, result in a more efficient computational program. One should remind that $A \oplus B$ may be just a subformula.
The following theorem, however, asserts that changing $B \oplus A$ into $A \oplus B$ never leads to a less efficient computational program.

Theorem 2. $N_F - N_{F'} = R_F - R_{F'}$

where $F = A \oplus B$ and $F' = B \oplus A$.

Proof. Consider the number of Taylor arrays to be used in calculating the Taylor coefficients of F, and corresponding to subformulae which are not free. This number only depends on the types of the subformulae of F and is thus independent of the order $A \oplus B$ or $B \oplus A$.

On the other hand, this number is equal to $N_F - R_F - 1$. Thus $N_F - R_F - 1 = N_{F'} - R_{F'} - 1$. Q.e.d.

Investigation of the commutative law with respect to multiplication leads to almost the same results as above. The only difference is that now $A \notin fr$ and $B \notin fr$.
Thus:

Theorem 3. Let $F = A \otimes B$, $F' = B \otimes A$ and $R_A \le R_B$ then $N_F \le N_{F'}$.

Theorem 4. $N_F - N_{F'} = R_F - R_{F'}$

where $F = A \otimes B$ and $F' = B \otimes A$.

5.6.3.2. The associative law

Theorem 5. If $F = A \oplus (B \oplus C)$ and $F' = (A \oplus B) \oplus C$, then

$N_F \ge N_{F'}$ if $A \in fr$ and

$N_F \le N_{F'}$ if $A \notin fr$.

<u>Proof</u>. We calculate N_F, R_F, $N_{F'}$ and $R_{F'}$ by means of the following processes:

Treatment of:	leads to:
A:	$N_1 := N_A;\ R_1 := R_A;$
B:	$N_2 := N_1 + \pi(N_B - R_1);$
	$R_2 := R_B + \pi(R_1 - N_B);$
C:	$N_3 := N_2 + \pi(N_C - R_2);$
	$R_3 := R_C + \pi(R_2 - N_C);$
	$R_4 := R_3 + \beta(B \epsilon fr) + \beta(C \epsilon fr);$
$B \oplus C$:	$N_4 := N_3 + \beta(R_4 = 0);$
	$R_5 := \pi(R_4 - 1);$
	$R_6 := R_5 + 1 + \beta(A \epsilon fr);$
F:	$N_F := N_4;$
	$R_F := R_6 - 1.$

Treatment of:	leads to:
A:	$N'_1 := N_A;$
	$R'_1 := R_A;$
B:	$N'_2 := N'_1 + \pi(N_B - R'_1);$
	$R'_2 := R_B + \pi(R'_1 - N_B);$
	$R'_3 := R'_2 + \beta(A \epsilon fr) + \beta(B \epsilon fr);$
$A \oplus B$:	$N'_3 := N'_2 + \beta(R'_3 = 0);$

$R_4' := \pi(R_3' - 1);$

C: $N_4' := N_3' + \pi(N_C - R_4');$

$R_5' := R_C + \pi(R_4' - N_C);$

$R_6' := R_5' + 1 + \beta(C \epsilon fr);$

F': $N_F' := N_4';$

$R_F' = R_6' - 1.$

1. Assume $R_2 = R_2' = R_B + \pi(R_A - N_B) = 0$, then
   $R_B = 0 \wedge R_A \leq N_B$ and

   $N_F = \beta(R_4 = 0) + \pi(N_C - R_2) + \pi(N_B - R_1) + N_A$

   $\quad = \beta(R_4 = 0) + N_C + N_B - R_A + N_A;$

   moreover,

   $N_{F'} = \pi(N_C - R_4') + \beta(R_3' = 0) + \pi(N_B - R_1') + N_A$

   $\quad = \pi(N_C - R_4') + \beta(R_3' = 0) + N_B - R_A + N_A.$

   Thus,

   $N_F - N_{F'} = \beta(R_4 = 0) + N_C - \beta(R_3' = 0) - \pi(N_C - R_4')$

   $\quad = \beta(R_4 = 0) - \beta(R_3' = 0) + N_C - \pi(N_C - \pi(-1 + \beta(A\epsilon fr) + \beta(B\epsilon fr))).$

   From $- 1 + \beta(A\epsilon fr) + \beta(B\epsilon fr) \leq 1$ and $N_C \geq 1$,it follows that we may omit the first π. Hence,

$$N_F - N_{F'} = \beta(\beta(B\epsilon fr) + \beta(C\epsilon fr) + R_C = 0) - \beta(\beta(A\epsilon fr) + \beta(B\epsilon fr) = 0) + \pi(\beta(A\epsilon fr) + \beta(B\epsilon fr) - 1). \qquad (5.6.4)$$

1.1. Assume $A\epsilon fr$, then

$$N_F - N_{F'} = \beta(\beta(B\epsilon fr) + \beta(C\epsilon fr) + R_C = 0) + \beta(B\epsilon fr) \geq 0.$$

1.2. Assume $A\not\epsilon fr$, then

$$N_F - N_{F'} = \beta(\beta(B\epsilon fr) + \beta(C\epsilon fr) + R_C = 0) - \beta(\beta(B\epsilon fr) = 0)$$

and it is easily seen that $N_F - N_{F'} \leq 0$; for, if $N_F - N_{F'}$ were equal to 1 then $\beta(\beta(B\epsilon fr) = 0) = 0$ and $B\epsilon fr$, but then the first term is zero.

Thus, in the case that $R_2 = R_2' = 0$, the theorem is proved.

2. Assume now that $R_2 = R_2' > 0$.

2.1. If $R_2 > N_C$ and thus $R_2' > N_C$, then

$$N_3 = N_2,\ R_3 = R_C + R_2 - N_C,$$

$$N_F = N_A + \pi(N_B - R_A) + \beta(\beta(B\epsilon fr) + \beta(C\epsilon fr) + R_C + R_2 - N_C = 0)$$

$$= N_A + \pi(N_B - R_A),$$

$$N_3' = N_A + \pi(N_B - R_A) + \beta(R_2 + \beta(A\epsilon fr) + \beta(B\epsilon fr) = 0)$$

$$= N_A + \pi(N_B - R_A),$$

$$N_F' = N_A + \pi(N_B - R_A) + \pi(N_C - \pi(R_2 + \beta(A\epsilon fr) + \beta(B\epsilon fr) - 1))$$

$$= N_A + \pi(N_B - R_A) + \pi(N_C - (R_2 + \beta(A\epsilon fr) + \beta(B\epsilon fr) - 1))$$

$$= N_A + \pi(N_B - R_A),$$

and $N_F = N_F'$.

2.2. Next we assume $R_2 \leq N_C$, thus also $R_2' \leq N_C$; then

$$N_3 = N_A + \pi(N_B - R_A) + N_C - R_2,\ R_3 = R_C,$$

$$N_F = N_A + \pi(N_B - R_A) + N_C - R_2 + \beta(R_C + \beta(B\epsilon fr) + \beta(C\epsilon fr) = 0),$$

and

$$N'_3 = N_A + \pi(N_B - R_A) + \beta(R_2 + \beta(A\epsilon fr) + \beta(B\epsilon fr) = 0),$$

$$N'_F = N'_3 + \pi(N_C - \pi(R_2 + \beta(A\epsilon fr) + \beta(B\epsilon fr) - 1)).$$

Thus,

$$\begin{aligned} N_F - N'_F = N_C - R_2 &+ \beta(R_C + \beta(B\epsilon fr) + \beta(C\epsilon fr) = 0) \\ &- \beta(R_2 + \beta(A\epsilon fr) + \beta(B\epsilon fr) = 0) \\ &- \pi(N_C - \pi(R_2 + \beta(A\epsilon fr) + \beta(B\epsilon fr) - 1)). \end{aligned} \quad (5.6.5)$$

2.2.1. Assume $A\epsilon fr$, then

$$\begin{aligned} N_F - N'_F = N_C - R_2 &+ \beta(R_C + \beta(B\epsilon fr) + \beta(C\epsilon fr) = 0) \\ &- \pi(N_C - R_2 - \beta(B\epsilon fr)). \end{aligned}$$

2.2.1.1. Let $N_C = R_2$, then

$$N_F - N'_F = \beta(R_C + \beta(B\epsilon fr) + \beta(C\epsilon fr) = 0) \geq 0.$$

2.2.1.2. Let $N_C > R_2$, then

$$N_F - N'_F = \beta(R_C + \beta(B\epsilon fr) + \beta(C\epsilon fr) = 0) + \beta(B\epsilon fr) \geq 0.$$

2.2.2. Now we assume $A\notin fr$.

2.2.2.1. A next assumption is $N_C = R_2$; thus $R_2 \geq 1$ and

$$\begin{aligned} N_F - N'_F = \beta(R_C &+ \beta(B\epsilon fr) + \beta(C\epsilon fr) = 0) \\ &-\pi(-\beta(B\epsilon fr) + 1) \end{aligned}$$

2.2.2.1.1. If $B\epsilon fr$, then $N_F - N'_F = 0$

2.2.2.1.2. If $B\notin fr$, then

$$N_F - N'_F = \beta(R_C + \beta(C\epsilon fr) = 0) - 1 \leq 0$$

2.2.2.2. We now assume $N_C > R_2$.

2.2.2.2.1. Let $B \in fr$, then

$$N_F - N_F' = N_C - R_2 - \pi(N_C - R_2) = 0.$$

2.2.2.2.2. Let, finally, $B \notin fr$, then

$$N_F - N_F' = N_C - R_2 + \beta(R_C + \beta(C \in fr) = 0)$$
$$- \beta(R_2 = 0) - \pi(N_C - \pi(R_2 - 1)).$$

2.2.2.2.2.1. If $R_2 = 0$, then

$$N_F - N_F' = \beta(R_C + \beta(C \in fr) = 0) - 1 \leq 0.$$

2.2.2.2.2.2. If $R_2 > 0$, then

$$N_F - N_F' = \beta(R_C + \beta(C \in fr) = 0) - 1 \leq 0.$$

Theorem 6. If $F = A \otimes (B \otimes C)$ and $F' = (A \otimes B) \otimes C$ then $N_F \leq N_{F'}$.

Proof. If we substitute in theorem 5: $A \notin fr$, $B \notin fr$ and $C \notin fr$ and change N_F into $N_4 + \beta(R_5 = 0)$ and N_F' into $N_4' + \beta(R_5' = 0)$, then we know from theorem 5 that $N_4 \leq N_4'$ since $A \notin fr$. We,therefore,have to investigate whether the following may occur:
$\beta(R_5 = 0) = 1$ and $\beta(R_5' = 0) = 0$ or $R_5 = 0$ and $R_5' > 0$.
From $R_5 = 0$ it follows $R_C + \pi(R_2 - N_C) \leq 1$.
From $R_5' > 0$ it follows $R_C + \pi(\pi(R_2 - 1) - N_C) \geq 1$.
Hence $\pi(\pi(R_2 - 1) - N_C) \geq \pi(R_2 - N_C)$,
which is only possible if $R_2 \leq N_C$; we then have $R_C = 1$.
In this case: $N_F - N_{F'} = 1 + N_4 - N_4'$;
if $R_2 = 0$, we have from equation (5.6.4): $N_F - N_{F'} = 0$;
and if $R_2 \geq 0$, we have from equation (5.6.5)

$$N_F - N_{F'} = 1 + N_C - R_2 - \pi(N_C - R_2 + 1) = 0.$$

Q.e.d.

5.6.3.3. A sum of n terms

We shall now treat the more difficult case of a formula F being a sum of n terms f_i, i = 1, ..., n (where each term is not itself a sum) and investigate the manner in which the terms f_i should be ordered and bracketed such that the number of Taylor arrays needed for F will be minimal.

Definition 1: The length of a formula F, denoted by L(F), is defined to be one, if F is not a sum; otherwise, it is equal to the number of the terms of F.

Definition 2: A permutation of a formula F, denoted by P(F), is a formula consisting of the same terms of F, but not necessarily ordered in the same way and not necessarily bracketed in the same way.

Definition 3: Let the formula F be the sum of the terms f_i, i = 1, ..., n; the ordering of F is defined to be the sequence of terms f_{i_1}, f_{i_2}, ..., f_{i_n} which are read from left to right if F were written out.

Remark: If F has the ordering f_1, f_2, ..., f_n, then F would be printed as $f_1 + f_2 + \dots + f_n$, if the printing of brackets were suppressed.

Definition 4: Let the ordering of the formula F be f_1, f_2, ..., f_n, then F has the standard ordering, or $F \epsilon SO$, if

$$\begin{aligned}&(R_{f_i} > R_{f_{i+1}}) \vee \\ &(R_{f_i} = R_{f_{i+1}}) \wedge ((f_i \epsilon fr \wedge f_{i+1} \not\epsilon fr) \vee \\ &\qquad ((f_i \epsilon fr = f_{i+1} \epsilon fr) \wedge (N_{f_i} \leq N_{f_{i+1}}))),\end{aligned} \qquad (5.6.6)$$

for all i = 1, ..., n-1.

Remark: The standard ordering is not always defined uniquely.

Definition 5: Let the ordering of the formula F be f_1, f_2, ..., f_n; let k be the index of the first term which is free; i.e., $f_i \not\epsilon fr$, i = 1, ..., k-1 and $f_k \epsilon fr$; if none of the terms are free, then $k \overset{d}{=} n+1$. F has the standard bracketing, or $F \epsilon SB$, if

$$F = \begin{cases} (f_1 \oplus (f_2 \oplus (\ldots \oplus (f_{n-1} \oplus f_n) \ldots))), \text{ if } k = n+1, \\ ((\ldots((f_1 \oplus f_2) \oplus f_3) \oplus \ldots) \oplus f_n), \text{ if } k = 1, \\ (f_1 \oplus (f_2 \oplus (\ldots \oplus (f_{k-1} \oplus ((\ldots((f_k \oplus f_{k+1}) \oplus f_{k+2}) \oplus \ldots) \\ \qquad \oplus f_n)) \ldots))), \text{ if } 1 < k \le n. \end{cases} \quad (5.6.7)$$

Remark: For a given ordering of f_i, $i = 1, \ldots, n$, the standard bracketing is defined uniquely.

Definition 6: F has the minimum property, or $F \epsilon MP$, if for all possible permutations $F' = P(F)$, $N_F \le N_{F'}$.

We now state the main theorem:

Theorem 7: If F has the standard ordering and the standard bracketing, then F has the minimum property.
(Or $F \epsilon SO \wedge F \epsilon SB \implies F \epsilon MP$.)

The proof of this theorem is an immediate consequence of the following lemmas. During the proofs of these lemmas, we shall use the induction argument on the length of a formula; in particular, we shall assume that theorem 7 holds for all formulae with a length less than n. From theorem 1 we conclude that n may be taken ≥ 3.

Lemma 1: Let $F_1 \epsilon MP$ then an F_2 can be constructed from F_1, such that $F_2 \epsilon MP \wedge F_2 \epsilon SO$.

Lemma 2: Let $F_2 \epsilon MP \wedge F_2 \epsilon SO$, then an F_3 can be constructed from F_2, such that $F_3 \epsilon MP \wedge F_3 \epsilon SO \wedge F_3 \epsilon SB$.

Lemma 3: Let $F_3 \epsilon SO \wedge F_3 \epsilon SB$ and $F \epsilon SO \wedge F \epsilon SB$, then $N_{F_3} = N_F$.

Before proving the above lemmas, we first state and prove the following (auxiliary) lemma:

<u>Lemma 4</u>: Let $A^* = P(A)$ and $B^* = P(B)$; let, moreover, $N_A \leq N_{A^*}$ and $N_B \leq N_{B^*}$, then

$$N_{A \oplus B} \leq N_{A^* \oplus B^*}.$$

<u>Proof of lemma 4</u>: As in the proof of theorem 2, we use the fact that

$N_A = C_A + R_A + 1$, where

C_A = the sum of Taylor arrays which are used for sub-formulae of A which are not free.

R_A = the sum of Taylor arrays which may be used again.

Obviously, $C_A = C_{A^*}$ (a permutation does not change the property of a subformula of A being free or not).

Hence $N_A - N_{A^*} = R_A - R_{A^*}$; and in the same way

$N_B - N_{B^*} = R_B - R_{B^*}$.

Furthermore, we shall use the facts that $A\epsilon fr = A^*\epsilon fr$ and $B\epsilon fr = B^*\epsilon fr$; for, if $L(A) = 1$ then $A = A^*$ and if $L(A) > 1$ then both A and A^* are a sum (similar reasoning for B).

From equation (5.6.2) we have:

$$N_{A \oplus B} = N_A + \pi(N_B - R_A) + \beta(R_B + \pi(R_A - N_B) + \beta(A\epsilon fr) + \beta(B\epsilon fr) = 0).$$

Assume $N_B \geq R_A$ and $N_{B^*} \geq R_{A^*}$, then

$$\begin{aligned} N_{A \oplus B} - N_{A^* \oplus B^*} = {} & N_A - N_{A^*} + N_B - N_{B^*} - R_A + R_{A^*} \\ & + \beta(R_B + \beta(A\epsilon fr) + \beta(B\epsilon fr) = 0) \\ & - \beta(R_{B^*} + \beta(A^*\epsilon fr) + \beta(B^*\epsilon fr) = 0). \end{aligned}$$

Disregarding the trivial case: $A\epsilon fr \vee B\epsilon fr$, we obtain

$$N_{A \oplus B} - N_{A^* \oplus B^*} = N_B - N_{B^*} + \beta(R_B = 0) - \beta(R_{B^*} = 0).$$

If $\beta(R_B = 0) = 1$ and $\beta(R_{B^*} = 0) = 0$, then

$N_{B^*} > N_B$ and $N_{A \oplus B} \leq N_{A^* \oplus B^*}$.

If $\beta(R_B = 0) = 0$, or $\beta(R_{B^*} = 0) = 1$, we immediately have

$N_{A \oplus B} \leq N_{A^* \oplus B^*}$.

Assume $N_B < R_A$ and $N_{B^*} \geq R_{A^*}$, then

$$\begin{aligned} N_{A \oplus B} - N_{A^* \oplus B^*} = {} & N_A - N_{A^*} - N_{B^*} + R_{A^*} - \\ & - \beta(R_{B^*} + \beta(A^*\epsilon fr) + \beta(B^*\epsilon fr) = 0) \leq 0. \end{aligned}$$

Assume $N_B \geq R_A$ and $N_{B^*} < R_{A^*}$, then

$$\begin{aligned} N_{A \oplus B} - N_{A^* \oplus B^*} &= N_A - N_{A^*} + N_B - R_A + \\ &+ \beta(R_B + \beta(A \epsilon fr) + \beta(B \epsilon fr) = 0) \\ &= R_A - R_{A^*} + N_B - R_A + \beta(R_B + \beta(A\epsilon fr) + \beta(B\epsilon fr) = 0) \\ &= N_{B^*} - R_{A^*} + (N_B - N_{B^*}) + \\ &+ \beta(R_B + \beta(A\epsilon fr) + \beta(B\epsilon fr) = 0) \le 0 \end{aligned}$$

Assume, finally, $N_B < R_A$ and $N_{B^*} < R_{A^*}$, then

$N_{A \oplus B} - N_{A^* \oplus B^*} = N_A - N_{A^*} \le 0$,

which proves lemma 4.

<u>Proof of lemma 1</u> (let $F_1 \epsilon MP \Longrightarrow \exists F_2, F_2 \epsilon MP \wedge F_2 \epsilon SO$).

Since $L(F_1) \ge 3$, $F_1 = A \oplus B$.
Let A^* and B^* be permutations of A and B, respectively, such that $A^* \epsilon SO \wedge A^* \epsilon SB$ and $B^* \epsilon SO \wedge B^* \epsilon SB$. Using the induction argument we may assume that $A^* \epsilon MP$ and $B^* \epsilon MP$.
From $F_1 \epsilon MP$ and lemma 4, it now follows that

$$F_1^* = A^* \oplus B^* \epsilon MP.$$

The form of $A^* \oplus B^*$, when written out, is,in general:

$$\begin{aligned} &A^* \oplus B^* = \\ &(a_1 \oplus (a_2 \oplus (\dots \oplus (a_{k_a-1} \oplus ((\dots ((a_{k_a} \oplus a_{k_a+1}) \oplus a_{k_a+2}) \oplus \dots) \oplus a_l))\dots))) \\ &\oplus (b_1 \oplus (b_2 \oplus (\dots \oplus (b_{k_b-1} \oplus ((\dots ((b_{k_b} \oplus b_{k_b+1}) \oplus b_{k_b+2}) \oplus \dots) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \oplus b_m)) \dots))). \end{aligned}$$

This is the form of $A^* \oplus B^*$ when $1 < k_a \le l$ and $1 < k_b \le m$; we shall investigate this case only, since the other cases can be treated similarly.

We define C to be:

$C = (\dots ((a_{k_a} \oplus a_{k_a+1}) \oplus a_{k_a+2}) \oplus \dots \oplus a_{l-1})$;

and shall omit the indexes l and 1 of a_l and b_1.
(Notice,that C may be considered to be "void" if $k_a = l + 1$.)

Let the number of Taylor arrays used, just before the treatment of "a" be N_0, and the corresponding number of remaining free Taylor arrays be R_0.

The current values of the number of Taylor arrays needed and the number of free Taylor arrays will be denoted by N and R, respectively.

After the treatment of a:

$N := N_0 + \pi(N_a - R_0);$

$R := R_a + \pi(R_0 - N_a).$

After the treatment of $C \oplus a$:

$R := R + \{\beta(a \in fr) + \beta(C \in fr)\} \times \beta(l > 1);$

$N := N + \beta(R = 0) \times \beta(l > 1);$

$R := \pi(R - \beta(l > 1)).$

After the treatment of the $k_a - 1$ sums $(a_{k_a - 1} \oplus (\ldots)), \ldots, (a_1 \oplus (\ldots))$, if they are present, N and R do not change since $a_i \notin fr$ for $0 < i < k_a$.

Finally, b is treated and

$N := N + \pi(N_b - R).$

If the value of N is denoted by N_1, then

$$\begin{aligned} N_1 = N_0 &+ \pi(N_a - R_0) + \\ &+ \beta(l > 1) \times \beta(R_a + \pi(R_0 - N_a) + \{\beta(a \in fr) + \beta(C \in fr)\} = 0) + \\ &+ \pi(N_b - \pi(R_a + \pi(R_0 - N_a) + \{\beta(a \in fr) + \beta(C \in fr) - 1\} \times \beta(l > 1))). \end{aligned}$$

We shall suppose that $A^* \oplus B^* \notin SO$, then

$$R_a < R_b \vee (R_a = R_b \wedge ((a \notin fr \wedge b \in fr) \vee ((a \in fr = b \in fr) \wedge N_a > N_b)))$$

Our objective is to show that after an interchange of a and b, the number of Taylor arrays N_2 then needed is not larger than N_1.

N_2 is equal to N_1 in which a and b are interchanged; hence

$$\begin{aligned} N_1 - N_2 = \pi(N_a - R_0) &- \pi(N_b - R_0) + \qquad (5.6.8) \\ &+ \beta(l > 1) \times [\beta(R_a + \pi(R_0 - N_a) + \{\beta(a \in fr) + \beta(C \in fr)\} = 0) + \\ &\qquad - \beta(R_b + \pi(R_0 - N_b) + \{\beta(b \in fr) + \beta(C \in fr)\} = 0)] + \\ &+ \pi(N_b - \pi(R_a + \pi(R_0 - N_a) + \{\beta(a \in fr) + \beta(C \in fr) - 1\} \times \beta(l > 1))) + \\ &- \pi(N_a - \pi(R_b + \pi(R_0 - N_b) + \{\beta(b \in fr) + \beta(C \in fr) - 1\} \times \beta(l > 1))). \end{aligned}$$

If $l = 1$, then

$$\begin{aligned} N_1 - N_2 = {} & \pi(N_a - R_0) - \pi(N_b - R_0) + \\ & + \pi(N_b - \pi(R_a + \pi(R_0 - N_a))) + \\ & - \pi(N_a - \pi(R_b + \pi(R_0 - N_b))). \end{aligned}$$

In the following we shall use: $N_b > R_b \geq R_a$ and $N_a > R_a$.
Let $N_a - R_0 > 0 \wedge N_b - R_0 > 0$ then

$$N_1 - N_2 = N_a - R_a - \pi(N_a - R_b) = \left\{ \begin{matrix} R_b - R_a \\ \text{or} \\ N_a - R_a \end{matrix} \right] \geq 0.$$

Let $N_a - R_0 \leq 0 \wedge N_b - R_0 > 0$ then

$$\begin{aligned} N_1 - N_2 &= R_0 - N_b + \pi(N_b - R_a - R_0 + N_a) - \pi(N_a - R_b) \\ &= N_a - R_a - \pi(N_a - R_b) \geq 0. \end{aligned}$$

Let $N_a - R_0 > 0 \wedge N_b - R_0 \leq 0$ then

$$\begin{aligned} N_1 - N_2 &= N_a - R_0 + N_b - R_a - \pi(N_a - R_b - R_0 + N_b) \\ &= R_b - R_a \geq 0. \end{aligned}$$

Let, finally, $N_a - R_0 \leq 0 \wedge N_b - R_0 \leq 0$ then

$$N_1 - N_2 = \pi(N_b - R_a - R_0 + N_a) - \pi(N_a - R_b - R_0 + N_b) \geq 0.$$

We shall now consider the case $l > 1$.
In the following we assume $N_a - R_0 > 0 \wedge N_b - R_0 > 0$.

$$\begin{aligned} N_1 - N_2 = {} & N_a - N_b + \\ & \quad \beta(R_a + \beta(a \epsilon fr) + \beta(C \epsilon fr) = 0) \\ & - \beta(R_b + \beta(b \epsilon fr) + \beta(C \epsilon fr) = 0) \\ & + \pi(N_b - \pi(R_a + \beta(a \epsilon fr) + \beta(C \epsilon fr) - 1)) \\ & - \pi(N_a - \pi(R_b + \beta(b \epsilon fr) + \beta(C \epsilon fr) - 1)). \end{aligned}$$

Consider first the case $R_b > R_a$; then

$$\begin{aligned} N_1 - N_2 = {} & N_a + \beta(R_a + \beta(a \epsilon fr) + \beta(C \epsilon fr) = 0) \\ & - \pi(R_a + \beta(a \epsilon fr) + \beta(C \epsilon fr) - 1) \\ & - \pi(N_a - R_b - \beta(b \epsilon fr) - \beta(C \epsilon fr) + 1). \end{aligned}$$

Hence,

$$N_1-N_2 = \begin{cases} N_a + \beta(R_a + \beta(a\epsilon fr) + \beta(C\epsilon fr) = 0) > 0, \\ \text{or,} \\ N_a - R_a + \beta(R_a + \beta(a\epsilon fr) + \beta(C\epsilon fr) = 0) - \beta(a\epsilon fr) - \beta(C\epsilon fr)+1 \geq 0, \\ \text{or,} \\ R_b + \beta(R_a + \beta(a\epsilon fr) + \beta(C\epsilon fr) = 0) + \beta(b\epsilon fr) + \beta(C\epsilon fr) - 1 \geq 0, \\ \text{or,} \\ R_b - R_a + \beta(R_a + \beta(a\epsilon fr) + \beta(C\epsilon fr) = 0) + \beta(b\epsilon fr) - \beta(a\epsilon fr) \geq 0. \end{cases}$$

Consider next the case $R_b = R_a \wedge a\notin fr \wedge b\epsilon fr$; then

$$\begin{aligned} N_1 - N_2 = {} & N_a + \beta(R_a + \beta(C\epsilon fr) = 0) \\ & - \pi(R_a + \beta(C\epsilon fr) - 1) \\ & - \pi(N_a - R_b - \beta(C\epsilon fr) + 1). \end{aligned}$$

Hence,

$$N_1-N_2 = \begin{cases} N_a + \beta(R_a + \beta(C\epsilon fr) = 0) > 0, \\ \text{or,} \\ N_a - R_a + \beta(R_a + \beta(C\epsilon fr) = 0) - \beta(C\epsilon fr) + 1 > 0, \\ \text{or,} \\ R_b + \beta(R_a + \beta(C\epsilon fr) = 0) + \beta(C\epsilon fr) - 1 \geq 0, \\ \text{or,} \\ R_b - R_a + \beta(R_a + \beta(C\epsilon fr) = 0) \geq 0. \end{cases}$$

Concluding, if $N_a - R_0 > 0 \wedge N_b - R_0 > 0$, then $N_1 - N_2 \geq 0$.
The other cases can be treated similarly also leading to $N_1 - N_2 \geq 0$.
Therefore, the interchange of a and b does not enlarge the number of Taylor arrays needed.
After the interchangement we obtain $A^{**} \oplus B^{**}$ which resembles the standard ordering more closely than $A^{*} \oplus B^{*}$.

Repetition of the arguments leads, in a finite number of steps, to the "q.e.d." of lemma 1.

Proof of lemma 2 ($\text{If } F_2 \epsilon MP \wedge F_2 \epsilon SO \Rightarrow \exists F_3 : F_3 \epsilon MP \wedge F_3 \epsilon SO \wedge F_3 \epsilon SB$):

Since $n = L(F_2)$ is assumed to be ≥ 3, we may write $F_2 = A \oplus B$. Applying the induction argument we can construct A^* and B^*, such that $A^* = P(A)$, $B^* = P(B)$, $A^* \epsilon MP \wedge A^* \epsilon SO \wedge A^* \epsilon SB$ and $B^* \epsilon MP \wedge B^* \epsilon SO \wedge B^* \epsilon SB$.

From $F_2 \epsilon SO$ and, hence, $A \epsilon SO$ and $B \epsilon SO$, we may assume that the orderings of A^* and B^* are the same as the orderings of A and B.

Let F_2 have the ordering $f_1, f_2, \ldots, f_n$. In constructing F_3, we shall not change the ordering of F_2; therefore, $F_3 \epsilon SO$.

1. Assume $A^* = f_1$, and, hence, $B^* = C \oplus D$.

1.1. If $f_1 \epsilon fr$, then take $F_2^* = (f_1 \oplus C) \oplus D$; using theorem 5 we have $F_2^* \epsilon MP$.

1.2. If $f_1 \notin fr$, then take $f_1 \oplus B^*$; from $B^* \epsilon SB$ it follows $F_3 \epsilon SB$; moreover, we have already $F_3 \epsilon MP$.

2. Assume $A^* = f_1 \oplus E$.

2.1. If $f_1 \epsilon fr$, then $E = f_2$ (otherwise $A^* \notin SB$).

2.1.1. If $L(B^*) = 1$, then take $F_3 = A^* \oplus B^*$; evidently $F_3 = (f_1 \oplus f_2) \oplus f_3 \epsilon SB$.

2.1.2. If $L(B^*) > 1$; hence $B^* = C \oplus D$, then take $F_2^* = (A^* \oplus C) \oplus D$. From theorem 5 we have $F_2^* \epsilon MP$.

2.2. If $f_1 \notin fr$, then take $F_2' = f_1 \oplus (E \oplus B^*)$. From theorem 5 we have $F_2' \epsilon MP$. Let G be $E \oplus B^*$ brought into the standard bracketing; then take $F_3 = f_1 \oplus G$. Evidently, $F_3 \epsilon MP$ and $F_3 \epsilon SB$.

3. Assume, finally, $A^* = E \oplus G$, where $L(E) \geq 2$.

3.1. If $L(B^*) = 1$ and, hence, $B^* = f_n$, then take $F_3 = A^* \oplus B^*$. Evidently, $F_3 \epsilon SB$.

3.2. If $L(B^*) > 1$, i.e. $B^* = C \oplus D$, then take $F_2^* = (A^* \oplus C) \oplus D$; according to theorem 5, $F_2^* \epsilon MP$.

Concluding: either an F_3 has been constructed, or an F_2^* has been constructed with the properties: $F_2^* \epsilon SO \wedge F_2^* \epsilon MP$ and if F_2^* is written as $A_2^* \oplus B_2^*$, then $L(B_2^*) < L(B^*)$. Repetition of the above arguments will, in a finite number of steps, lead to the desired F_3.
This proves lemma 2.

<u>Proof of lemma 3</u> (let $F_3 \epsilon SO \wedge F_3 \epsilon SB$ and $F_4 \epsilon SO \wedge F_4 \epsilon SB \implies N_{F_3} = N_{F_4}$):
Let F_3 and F_4 differ in the location of f; say, the i-th term of F_3.
Let the i-th term of F_4 be g.
It follows from definition 3 that

$$R_f = R_g,\ f \epsilon fr = g \epsilon fr,\ N_f = N_g.$$

(Note that only in this case the ordering is not defined uniquely.)
It is evident, however, that in, say, F_3 the term f may be interchanged with the term g, without affecting N_{F_3}. This may be done for all remaining terms in F_3 whose index in F_3 is not their index in F_4.
Hence $N_{F_3} = N_{F_4}$, which proves lemma 3.
We finally state, without proof, the analogue of theorem 7 for a product F of n factors f_i, each factor not being a product.

<u>Theorem 8</u>: Let the formula F be a product of n factors f_i, $i = 1, \ldots, n$; of all possible permutations of F, the following form involves a minimal number of Taylor arrays:

$$F = (f_{i_1} \otimes (f_{i_2} \otimes (\ldots \otimes (f_{i_{n-1}} \otimes f_{i_n}) \ldots))),$$

where the $f_{i_1}, f_{i_2}, \ldots, f_{i_n}$ satisfy:

$$(R_{f_{i_j}} > R_{f_{i_{j+1}}}) \vee$$

$$(R_{f_{i_j}} = R_{f_{i_{j+1}}}) \wedge (N_{f_i} \leq N_{f_{i_{j+1}}}), \text{ for } j = 1, \ldots, n-1.$$

5.7. The algebraic program

5.7.1. Introductory remarks

In this section we describe the algebraic program which produces the ALGOL 60 statements for the computational program of section 5.8 for calculating the Taylor coefficients of the functions $U_k(x_1, \ldots, x_d)$, $k = 1, \ldots, K$, satisfying:

$$G_l\left(\frac{\partial U_1}{\partial x_1}, \ldots, \frac{\partial U_1}{\partial x_d}, \ldots, \frac{\partial U_K}{\partial x_1}, \ldots, \frac{\partial U_K}{\partial x_d},\right.$$

$$\left. U_1, \ldots, U_K, x_1, \ldots, x_d\right) = 0 \qquad (5.1.1)$$

for $l = 1, \ldots, K$.

The numbers t_k are defined by:
$\partial U_k/\partial x_{t_k}$ occurs in the differential equations, but $\partial U_k/\partial x_j$, $j > t_k$, do not occur in the differential equations; if $t_k = 0$ then $\partial U_k/\partial x_{t_k}$ means U_k itself. t_k is called the <u>type</u> of U_k.

We want to calculate the Taylor coefficients $u_{k,n_1,\ldots,n_d}$ for the indexes lying in the region $S(d,N_k)$ defined by:

$$S(d,N_k) = \{(n_1,\ldots,n_d) : n_1 \geq 0 \wedge \ldots \wedge n_d \geq 0 \wedge n_1 + \ldots + n_d \leq N_k\},$$
$$(k = 1, \ldots, K). \qquad (5.7.1)$$

In the computational program the variable *highest degree* is used to define the numbers N_k. If $t_k > 0$ then N_k = *highest degree*, whereas if $t_k = 0$, then N_k = *highest degree* - 1.

The subformulae A_i, by means of which the left-hand sides of (5.1.1) are built up, have Taylor coefficients $a_{i,n_1,\ldots,n_d}$. It is not possible to store them in a (d+1)-dimensional array, since d is determined at run time of the computational program. Therefore, they are stored in a 2-dimensional array declared by:

array a[1 : number of Taylor series, 0 : M-1]; where *M* is the number of indexes lying in S(d,*highest degree* - 1). For the administration,use is made of an integer procedure: *integer procedure I(n sub); integer array n sub;* The values of $I(n_1, \ldots, n_d)$ are such that there is a one-to-one correspondence between the Taylor coefficients $a_{i,n_1,\ldots,n_d}$ and the array elements of $a[i,I(n_1, \ldots, n_d)]$, for $(n_1, \ldots, n_d)$ lying in S(d,*highest degree* - 1).
At execution time of the computational program the value of $I(n_1,\ldots,n_d)$ is assigned to the variable *n*; therefore, the algebraic program may output the Taylor coefficients in the simple form *a*[j,*n*], where j is some number.

The Taylor coefficients $u_{k,n_1,\ldots,n_d}$ are stored into the array elements $a[k,I(n_1,\ldots,n_d)]$. In order to treat the difficulty that, when $t_k > 0$, there are more $u_{k,n_1,\ldots,n_d}$'s than array elements $a[k,I(n_1,\ldots,n_d)]$, special array elements $u[i,M], \ldots, u[i,M_1-1]$, where M_1 is the number of indexes lying in S(d,*highest degree*), are introduced.

The main part of the algebraic program consists of the procedure *DEFINE TAYLOR COEFFICIENTS* which performs the operations for the *differential equations* given as an array. The number,K,of differential equations is given as the actual value of the parameter *order*. The heading of the procedure is now reproduced.

```
procedure DEFINE TAYLOR COEFFICIENTS
  (order,differential equations);
  value order; integer order; integer array differential equations;
begin integer i,j,k,T,next Taylor index,pointer of set of free
  indexes,nth time function occurred,N,R,pointer of coeff;
  Boolean first time,free;
  integer array unknown,type of u,N diff eq,R diff eq,new index
    [1:order];
  Boolean array diff eq is free[1:order];
```

5.7.2. The preparatory operations

According to the investigations of section 5.6.3, the differential equations are transformed into differential equations which render a more efficient computational program. This is achieved by means of the procedure *TRANSFORM* as follows: First, the constants in a formula are combined into one constant; for this, the procedures *CONSTANT* and *CONSTANT FORMULA* are introduced, for determining whether a formula is constant and for storing a formula of type *constant*, respectively. Second, the theorems for the commutative and associative laws are applied. The procedure *TRANSFORM* calculates, moreover, the number of times a function symbol occurs and assigns this number to *n th time function occurred*; finally, it calculates the numbers p_k which are assigned to *type of u[k]*.

With respect to *TRANSFORM* we remark:

1. the parameters *numb of used arr* and *numb of free arr* correspond to the symbols N and R of section 5.6.3;
2. if the quantities lhs and type of a formula f are *dependent* and *algebraic variable*, respectively, then f is of the form $\partial U_k/\partial x_j$, where j and k are given by the quantity rhs which equals $j + 32 \times k$; when $j = 0$ then f is of the form U_k;
3. integral powers with base x_j are treated in a special way;
4. in the case of a function, the number of Taylor series needed is given by *N of function* (see section 5.7.6);
5. the procedure uses the procedures *pos part* and *bool*, which are the functions π and β as introduced in section 5.6.3.

Besides the procedure *TRANSFORM*, the first part of the procedure body of *DEFINE TAYLOR COEFFICIENTS* is given which calls the procedure *TRANSFORM*, for transforming the differential equations and reordering the new differential equations. Their new indexes are given by the array elements *new index [i]*. Next, the new differential equations are being output such that one can inspect the results. Finally, the arrays *formula for*, *set of indexes*, *set of free indexes* and *TPSI* are declared (*TPSI* is used for functions and integral powers, see section 5.7.6).

```
procedure COUNT(f,type,array,bounds only);
value f,type; integer f,type; Boolean bounds only;
integer array array;
begin integer t,a,b; t:= TYPE(f,a,b);
  if t = type then
  begin COUNT(a,type,array,bounds only);
    COUNT(b,type,array,bounds only)
  end else
  begin array[0]:= array[0] + 1;
    if ¬ bounds only then array[array[0]]:= f
  end
end COUNT;

integer procedure pos part(i); value i; integer i;
pos part:= if i > 0 then i else 0;

integer procedure bool(p); value p; Boolean p;
bool:= if p then 1 else 0;

Boolean procedure CONSTANT(f); value f; integer f;
begin integer t,a,b; t:= TYPE(f,a,b);
  CONSTANT:=
    if t = constant ∨ t = number ∨ t = algebraic variable ∧
      a = complex then true else
    if t = sum ∨ t = product ∨ t = quotient then
        CONSTANT(a) ∧ CONSTANT(b)
    else if t = function ∨ t = integral power then CONSTANT(b)
  else false
end CONSTANT;

integer procedure CONSTANT FORMULA(f); value f; integer f;
CONSTANT FORMULA:= STORE(0,constant,f);

integer procedure TRANSFORM(f,numb of used arr,
  numb of free arr,free);
value f; integer f,numb of used arr,numb of free arr; Boolean free;
begin integer t,a,b,x,y; t:= TYPE(f,a,b);
```

```
if CONSTANT(f) then
begin TRANSFORM:= CONSTANT FORMULA(f); free:= true;
  numb of used arr:= 1; numb of free arr:= 0
end else

if t = algebraic variable then
begin numb of used arr:= 1; numb of free arr:= 0; free:= true;
  TRANSFORM:= f; if a = dependent then
  begin integer index of u,index of der; index of u:= b : 32;
    index of der:= b - index of u × 32; if type of u[index of u]
    < index of der then type of u[index of u]:= index of der;
    if index of der = 0 then
    begin numb of used arr:= numb of free arr:= 0; free:= false
end end end else

if t = sum ∨ t = product then
begin integer i,k,n,N,R,N1,R1,f1,g1,numb of constant terms;
  Boolean A is free; integer array A[0:0]; A[0]:= 0;
  COUNT(f,t,A,true); n:= A[0];
  begin integer array term[0:n],used arr,free arr,p[1:n];
    Boolean array term is free,term is constant[1:n];
    term[0]:= 0; COUNT(f,t,term,false); numb of constant terms:= 0;
    for i:= 1 step 1 until n do
    begin term[i]:= TRANSFORM(term[i],used arr[i],free arr[i],
      term is free[i]); p[i]:= i; if t = product then
      term is free[i]:= false;
      term is constant[i]:= TYPE(term[i],di,di) = constant;
      if term is constant[i] then numb of constant terms:=
      numb of constant terms + 1
    end;

  again1: for i:= 1 step 1 until n - 1 do
    begin if term is constant[p[i+1]] ∧ ¬ term is constant[p[i]] then
      begin k:= p[i]; p[i]:= p[i+1]; p[i+1]:= k; goto again1 end
    end; if numb of constant terms = 0 then goto again2;
    g1:= term[p[1]]; for i:= 2 step 1 until numb of constant terms do
    g1:= if t = sum then S(g1,term[p[i]]) else P(g1,term[p[i]]);
```

```
 again2: for i:= numb of constant terms + 1 step 1 until n - 1 do
   begin if (free arr[p[i]] < free arr[p[i+1]]) ∨
     ((free arr[p[i]] = free arr[p[i+1]]) ∧ ((term is free[p[i+1]] ∧
     ¬ term is free[p[i]]) ∨ (term is free[p[i+1]] ≡ term is free[p[i]]) ∧
     (used arr[p[i]] > used arr[p[i+1]])))
     then begin k:= p[i]; p[i]:= p[i+1]; p[i+1]:= k; goto again2 end
   end;
   k:= n; for i:= numb of constant terms + 1 step 1 until n - 1 do
   begin if term is free[p[i]] then begin k:= i; goto out end end;

 out: N:= used arr[p[k]]; R:= free arr[p[k]]; A is free:=
   term is free[p[k]]; f1:= term[p[k]];
   for i:= k + 1 step 1 until n do
   begin N1:= used arr[p[i]]; R1:= free arr[p[i]];
     N:= N + pos part(N1 - R); R:= R1 + pos part(R - N1);
     R:= R + bool(A is free) + bool(term is free[p[i]]);
     N:= N + bool(R = 0); R:= pos part(R - 1);
     A is free:= t = sum; f1:= if t= sum then S(f1,term[p[i]])
     else P(f1,term[p[i]])
   end; for i:= k - 1 step -1 until numb of constant terms + 1 do
   begin N1:= N; R1:= R;
     N:= used arr[p[i]] + pos part(N1 - free arr[p[i]]);
     R:= R1 + pos part(free arr[p[i]] - N1);
     R:= R + bool(term is free[p[i]]) + bool(A is free);
     N:= N + bool(R = 0); R:= pos part(R - 1);
     A is free:= t = sum; f1:= if t = sum then S(term[p[i]],f1)
     else P(term[p[i]],f1)
   end;
   numb of used arr:= N; numb of free arr:= R; free:= true;
   if numb of constant terms = n - 1 ∧ ¬ A is free then
   begin numb of used arr:= N + bool(R = 0);
     numb of free arr:= pos part(R - 1)
   end; TRANSFORM:= if numb of constant terms = 0 then f1 else
     (if t = sum then S(CONSTANT FORMULA(g1),f1) else
     P(CONSTANT FORMULA(g1),f1))
end end else
```

```
if t = quotient then
begin integer N,R,NA,RA,NB,RB; Boolean A is free,B is free;
  a:= TRANSFORM(a,NA,RA,A is free);
  b:= TRANSFORM(b,NB,RB,B is free);
  TRANSFORM:= Q(a,b);
  N:= NA + pos part(NB - RA);
  R:= RB + pos part(RA - NB) + bool(A is free);
  numb of used arr:= N + bool(R = 0);
  numb of free arr:= pos part(R - 1);
  free:= false
end else

if t = integral power then
begin if TYPE(b,x,y) = algebraic variable ∧
  x = independent then
  begin numb of used arr:= 1; numb of free arr:= 0; free:= true;
    TRANSFORM:= f
  end else
  begin integer procedure numb of aux arr(n); value n; integer n;
    numb of aux arr:= if n = 1 then 0 else
      numb of aux arr(n ÷ 2) + 1 + n - n ÷ 2 × 2;
    integer NA,RA,N; Boolean A is free;
    b:= TRANSFORM(b,NA,RA,A is free);
    N:= numb of aux arr(a); numb of used arr:= NA + pos part(N - RA);
    numb of free arr:= pos part(RA - N); free:= true;
    TRANSFORM:= STORE(a,t,b)
  end
end else

if t = function then
begin integer NA,RA,N of function; Boolean A is free;
  nth time function occurred:= nth time function occurred + 1;
  N of function:= if a = sinf ∨ a = cosf ∨ a = arctanf then 3 else
  if a = expf then 1 else 2;
  b:= TRANSFORM(b,NA,RA,A is free);
  numb of used arr:= NA + pos part(N of function - RA);
  numb of free arr:= pos part(RA - N of function);
  free:= false; TRANSFORM:= STORE(a,t,b)
end end TRANSFORM;
```

```
 comment first part of procedure body of
   DEFINE TAYLOR COEFFICIENTS;
 nth time function occurred:= 0;
 for i:= 1 step 1 until order do type of u[i]:= 0;
 PR nlcr; PR(⊀The differential equations are:⊁);
 for i:= 1 step 1 until order do
 begin PR nlcr; PR(⊀diff eq[⊁); PR integral number(i);
   PR(⊀] = ⊁); OUTPUT(differential equations[i]);
   differential equations[i]:= TRANSFORM(differential
   equations[i],N diff eq[i],R diff eq[i],diff eq is free[i]);
   new index[i]:= i
 end;
reorder: for i:= 1 step 1 until order - 1 do
 begin if R diff eq[new index[i]] < R diff eq[new index[i+1]] ∨
   R diff eq[new index[i]] = R diff eq[new index[i+1]] ∧
   ((diff eq is free[new index[i+1]] ∧ ¬ diff eq is free[new index[i]]) ∨
   (diff eq is free[new index[i]] = diff eq is free[new index[i+1]] ∧
   N diff eq[new index[i]] > N diff eq[new index[i+1]])) then
   begin k:= new index[i]; new index[i]:= new index[i+1];
     new index[i+1]:= k; goto reorder
   end
 end; N:= order + N diff eq[new index[1]];
 R:= R diff eq[new index[1]] + bool(diff eq is free[new index[1]]);
 for i:= 2 step 1 until order do
 begin N:= N + pos part(N diff eq[new index[i]] - R);
   R:= R diff eq[new index[i]] + pos part(R - N diff eq[new index[i]])
     + bool(diff eq is free[new index[i]])
 end;

 PR nlcr; PR(⊀The transformed differential equations are:⊁);
 for i:= 1 step 1 until order do
 begin PR nlcr; PR(⊀diff eq[⊁); PR integral number(new index[i]);
   PR(⊀] = ⊁); OUTPUT(differential equations[new index[i]])
 end; PR(⊀;⊁);

 begin integer array formula for,set of indexes,set of free
   indexes[1:N],TPSI[0:nth time function occurred,1:2];
```

5.7.3. Auxiliary equipment

In this subsection, the procedures are given which are used time and again in the procedures *Taylor index*, *DEF ALGOL ST FOR DYADIC OP* and *DEF ALGOL ST FOR FUNCTIONS*.
Some procedures have been introduced already in section 5.5, namely *PR*, *PR integral number*, *PR nlcr*, *COEFF*, *PR conv product*, *PR array element*, *PR zeroth array element* and *PR ass st for arr el*.
The others will be described now.
A call: *store index into free set (T)* has as effect that the Taylor index *T* is stored into the *set of free indexes*.
A call: *T:= take index from free set* has the effect that, if the *set of free indexes* is empty, a new Taylor index is introduced which is assigned to *T* and stored at the top of the array *set of indexes*. If, on the other hand, the *set of free indexes* is not empty, the top element of the array *set of free indexes* is removed and assigned to *T*; moreover, it is stored at the top of *set of indexes*, after it has been removed from *set of indexes*.

The reason for introducing the array *set of indexes* is, that the program needs the ordering later on, as given by *set of indexes*, of the Taylor indexes when it has to output statements defining Taylor coefficients which could not be outputted earlier due to their dependence on unknown Taylor coefficients of the functions U_k.
Note, that the natural ordering, as in the program of section 5.5, does not need to be preserved.

The final procedure is *KNOWN*; a call of *KNOWN(T)* gives as result *true* or *false* dependent on whether a statement defining $a[T,n]$ has already been output or not.

```
integer procedure COEFF(right hand side); integer right hand side;
begin integer x; pointer of coeff:= pointer of coeff + 1;
   PR nlcr; PR(⟨coeff[⟩); PR integral number(pointer of coeff);
   PR(⟨]:= ⟩); x:= right hand side; PR(⟨;⟩);
   COEFF:= STORE(coefficient,algebraic variable,pointer of coeff)
end COEFF;

integer procedure PR conv product(a,b,case a,case b);
value a,b,case a,case b; integer a,b,case a,case b;
begin integer i; integer array c[1:2]; c[1]:= case a; c[2]:= case b;
   for i:= 1,2 do c[i]:= if c[i] = check for zero then +1 else if c[i] =
   check for zero + diff then -2 else  if c[i] = diff then -1 else 0;
   PR(⟨CONV PRODUCT(⟩); PR integral number(a); PR(⟨,⟩);
   PR integral number(b); PR(⟨,⟩); PR integral number(c[1]);
   PR(⟨,⟩); PR integral number(c[2]); PR(⟨)⟩); PR conv product:= 1
end PR conv product;

integer procedure PR array element(TI); value TI; integer TI;
begin PR(⟨a[⟩); PR integral number(TI); PR(⟨,n]⟩);
   PR array element:= 1
end PR array element;

integer procedure PR zeroth array element(T); value T; integer T;
begin PR(⟨a[⟩); PR integral number(T); PR(⟨,0]⟩);
   PR zeroth array element:= 1
end PR zeroth array element;

integer procedure PR ass st for arr el(T,right hand side);
  value T; integer T,right hand side;
begin integer x; PR nlcr; PR array element(T); PR(⟨:= ⟩);
  x:= right hand side; PR(⟨;⟩); PR ass st for arr el:= 1;
  formula for [T]:= -1
end PR ass st for arr el;
```

```
procedure store index into free set(T); value T; integer T;
begin pointer of set of free indexes:= pointer of set of free indexes
  + 1; set of free indexes[pointer of set of free indexes]:= T
end store index into free set;

integer procedure take index from free set;
begin integer i,T; Boolean shift;
  if pointer of set of free indexes = 0 then T:= next Taylor index:=
    next Taylor index + 1 else
  begin T:= set of free indexes[pointer of set of free indexes];
    pointer of set of free indexes:= pointer of set of free indexes - 1;
    shift:= false;
    for i:= order + 1 step 1 until next Taylor index - 1 do
    begin if set of indexes[i] = T then shift:= true;
      if shift then set of indexes[i]:= set of indexes[i + 1]
    end
  end; set of indexes[next Taylor index]:= take index from free set:= T
end take index from free set;

Boolean procedure KNOWN(T); value T; integer T;
KNOWN:= formula for [T] < 0;
```

5.7.4. The procedure *Taylor index*

This procedure is an improved version of the last procedure *Taylor index* of section 5.5. It suffices for the description of the procedure to make some remarks:

1. *Taylor index* has as second parameter the Boolean *free*, which obtains the value *true* or *false* dependent on whether the formula given by the first parameter f is free or not.
2. If f is a sum or a product, only its lhs quantity is checked for being constant as a consequence of the special transformation by *TRANSFORM*.
3. In the case that the procedure *PR ass st for arr el(T,...)* is called, the array element *formula for* [*T*] obtains automatically the value -1 in the procedure body of *PR ass st for arr el*.
4. The non-local Boolean variable *first time* is used to determine whether *Taylor index* is producing statements for the zeroth Taylor coefficients or for the non-zeroth.
5. It is assumed that the computational program contains a procedure *BINOMIAL COEFFICIENT(i,j)* which becomes equal to the Taylor coefficient, with as indexes the current values of $n_1, \ldots, n_d$, of the function
$$(x_i^0 + (x_i - x_i^0))^j$$
expanded in the variable $(x_i - x_i^0)$.
This is, in fact, the only place where the origin $(x_1^0, \ldots, x_d^0)$ enters into the calculation.
6. It is assumed that the computational program contains a procedure *SHIFT(i,j)* which becomes equal to $a[i,I(n_1, \ldots, n_j+1, \ldots, n_d)]\times(n_j+1)$.
7. The case that the type t of the formula f is equal to *functional variable* corresponds to the treatment of functions; one should compare table 3 in section 5.3.3 and section 5.7.6.

```
integer procedure Taylor index(f,free); value f; integer f; Boolean free;
begin integer Ta,Tb,T,x,y,t,a,b; Boolean free a,free b; t:= TYPE(f,a,b);

  if t = sum then
  begin free:= true; if TYPE(a,x,y) = constant then
    begin Tb:= Taylor index(b,free b);
      if free b then store index into free set(Tb);
      Taylor index:= T:= take index from free set;
      if KNOWN(Tb) then PR ass st for arr el(T,(if first time then     *)
      OUTPUT(a) × PR(‘+’) else 1) × PR array element(Tb))
      else formula for [T]:= formula for [Tb]
    end else
    begin Ta:= Taylor index(a,free a); Tb:= Taylor index(b,free b);
      if free a then store index into free set(Ta);
      if free b then store index into free set(Tb);
      Taylor index:= T:= take index from free set;
      DEF ALGOL ST FOR DYADIC OP(T,Ta,sum,Tb)
  end end else

  if t = product then
  begin free:= true; if TYPE(a,x,y) = constant then
    begin Tb:= Taylor index(b,free b);
      if free b then store index into free set(Tb);
      Taylor index:= T:= take index from free set;
      if KNOWN(Tb) then PR ass st for arr el(T,
        PR(‘(’) × OUTPUT(a) × PR(‘) × ’) × PR array element(Tb))     *)
      else formula for [T]:= P(COEFF(OUTPUT(a)),formula for [Tb])
    end else
    begin Ta:= Taylor index(a,free a); Tb:= Taylor index(b,free b);
      Taylor index:= T:= take index from free set;
      DEF ALGOL ST FOR DYADIC OP(T,Ta,product,Tb)
  end end else
```

*) See the remarks on pp. 59-60.

```
if t = quotient then
begin free:= false; Ta:= Taylor index(a,free a);
  Tb:= Taylor index(b,free b);
  if free a then store index into free set(Ta);
  Taylor index:= T:= take index from free set;
  DEF ALGOL ST FOR DYADIC OP(T,Ta,quotient,Tb)
end else

if t = function then
begin free:= false; Tb:= Taylor index(b,free b);
  DEF ALGOL ST FOR FUNCTIONS(T,Tb,a);
  Taylor index:= T
end else

if t = integral power then
begin if TYPE(b,x,y) = algebraic variable ∧ x = independent then
  begin Taylor index:= T:= take index from free set; free:= true;
    PR ass st for arr el(T,PR(‹BINOMIAL COEFF(›) ×                      *)
    PR integral number(y) × PR(‹,›) × PR integral number(a) ×
    PR(‹)›))
  end else
  begin integer procedure aux arr(n); value n; integer n;
    if n = 1 then aux arr:= Tb else
    begin integer x,T; x:= aux arr(n : 2);
      aux arr:= T:= take index from free set;
      DEF ALGOL ST FOR DYADIC OP(T,x,product,x);
      if n : 2 × 2 ≠ n then
      begin aux arr:= x:= take index from free set;
        DEF ALGOL ST FOR DYADIC OP(x,T,product,Tb)
    end end aux arr;
    Tb:= Taylor index(b, free b); free:= true;
    Taylor index:= aux arr(a)
end end else
```

*) See the remarks on pp. 59-60.

```
 if t = constant then
 begin free:= true; Taylor index:= T:= take index from free set;
   PR ass st for arr el(T,if first time then OUTPUT(f) else PR(⊰0⊱))
 end else

 if t = algebraic variable ∧ a = independent then
 begin free:= true; Taylor index:= T:= take index from free set;
   PR ass st for arr el(T,PR(⊰BINOMIAL COEFF(⊱) ×          *)
   PR integral number(b) × PR(⊰, 1)⊱))
 end else

 if t = algebraic variable ∧ a = dependent then
 begin integer index of u, index of derivative;
   index of u:= b : 32; index of derivative:= b - index of u × 32;
   if index of derivative = 0 then
   begin Taylor index:= T:= index of u; free:= false end else
   begin Taylor index:= T:= take index from free set;
     free:= true
   end;
   if first time ∨ index of derivative ≠ type of u[index of u] then
   begin if index of derivative = 0 then formula for [T]:= -1 else
     PR ass st for arr el(T,OUTPUT(STORE(Unknown,
     algebraic variable,b)))
   end else formula for [T]:= unknown[index of u]
 end else

 if t = functional variable then
 begin free:= false; Taylor index:= b end
end Taylor index;
```

*) See the remarks on pp. 59-60.

5.7.5. The output for the dyadic operators

In order to calculate, in the computational program, the convolution product of two Taylor series without the leading term, e.g. $a[i,n] \times a[j,0]$, it is necessary that the procedure *CONV PRODUCT* has a parameter which indicates this situation. In the following procedure and the procedure of section 5.7.6 we, therefore, introduced the variable *check for zero* which is used as actual parameter of the procedure *PR conv product* in order to output a statement in which *CONV PRODUCT* occurs with such a parameter.

```
procedure DEF ALGOL ST FOR DYADIC OP(T,a,oper,b);
  value T,a,oper,b; integer T,a,oper,b;
begin integer x; switch SW:= SUM1,SUM2,SUM3,PROD1,PROD2,PROD3,
  QUOT1,QUOT2a,QUOT3;
  integer procedure case; case:= (if oper = sum then 0 else
    if oper = product then 1 else 2) × 3;

  procedure END; goto out;

  if KNOWN(a) ∧ KNOWN(b) then goto SW[case + 1] else
  if KNOWN(a) then goto SW[case + 2] else
  if KNOWN(b) then
  begin if oper = quotient then goto QUOT2b;
    x:= a; a:= b; b:= x; goto SW[case + 2]
  end else goto SW[case + 3];

SUM1: PR ass st for arr el(T,PR array element(a) × PR(⊀ + ⊁) ×   *)
  PR array element(b)); END;
SUM2: formula for [T]:= S(COEFF(PR array element(a)),formula for [b]);
  END;
SUM3: formula for[T]:= S(formula for [a],formula for [b]); END;
```

*) See the remarks on pp. 59-60.

```
PROD1: if first time then
  begin PR ass st for arr el(T, PR zeroth array element(a) ×           *)
    PR(‹ × ›) × PR zeroth array element(b))
  end else  PR ass st for arr el(T,PR conv product(a,b,0,0)); END;
PROD2: formula for [T]:= S(COEFF(PR conv product(a,b,0,
  check for zero)),P(COEFF(PR zeroth array element(a)),
  formula for [b])); END;
PROD3: formula for [T]:= S(P(formula for [a],
  COEFF(PR zeroth array element(b))),
    S(P(COEFF(PR zeroth array element(a)),formula for [b]),
    COEFF(PR conv product(a,b,check for zero,check for zero)))); END;

QUOT1: if first time then
  begin PR ass st for arr el(T,PR zeroth array element(a) ×
    PR(‹ / ›) × PR zeroth array element(b)); END                       *)
  end;
  PR ass st for arr el(T,PR(‹(›) × PR array element(a) ×               *)
  PR(‹ - ›) × PR conv product(T,b,check for zero,0) × PR(‹ )/ ›) ×
  PR zeroth array element(b)); END;
QUOT2a: formula for [T]:= S(COEFF(PR(‹(›) × PR array element(a) ×
  PR(‹ - ›) × PR conv product(T,b,check for zero,check for zero) ×     *)
  PR(‹) / ›) × PR zeroth array element(b)),
  P(COEFF(PR(‹ - ›) × PR zeroth array element(T) × PR(‹ / ›) ×
  PR zeroth array element(b)),formula for [b])); END;
QUOT2b: formula for [T]:= P(S(formula for [a],COEFF(
  PR(‹ - ›) × PR conv product(T,b,check for zero,0))),                 *)
  COEFF(PR(‹1/ ›) × PR zeroth array element (b))); END;
QUOT3: formula for [T]:= P(S(formula for [a],S(P(COEFF(
  PR(‹ - ›) × PR zeroth array element (T)),formula for [b]),           *)
  COEFF(PR(‹ - ›) × PR conv product(T,b,check for zero,
  check for zero)))),COEFF(PR(‹1/ ›) × PR zeroth array
  element (b))); END;
out:
end DEF ALGOL ST FOR DYADIC OP;
```

*) See the remarks on pp. 59-60.

5.7.6. The output for functions

Let the subformula $A_i = \Phi(A_j)$, then the procedure of this subsection produces output defining the a[i,n] in terms of the a[j,n].
From section 5.3.3 we have, by introducing the functions Φ_ν and Ψ_ν, $\nu = 1, \ldots, \mu$, with $\mu = 1$ or 2,

$$d\Phi_\nu(z)/dz = \Psi_\nu(\Phi_1, \ldots, \Phi_\mu, z). \qquad (5.7.1)$$

For the subformulae $\Phi_\nu^j = \Phi_\nu(A_j)$ and $\Psi_\nu^j = \Psi_\nu(\Phi_1(A_j), \ldots, \Phi_\mu(A_j), A_j)$ we have

$$\phi^j_{\nu,n_1,\ldots,n_d} = \frac{1}{n_q} \sum_{p_1=0}^{n_1} \ldots \sum_{p_d=0}^{n_d} \psi^j_{\nu,p_1,\ldots,p_d}(n_q-p_q)a_{j,n_1-p_1,\ldots,n_d-p_d},$$

where $n_q \neq 0$. (5.7.2)

The a[i,n] are equal to $\phi^j_{1,n_1,\ldots,n_d}$.

Let us now see how the procedure *DEF ALGOL ST FOR FUNCTIONS*, with parameters *T*, *Ta* and *name of function* performs the necessary operations. The actual value of *Ta* gives the Taylor index j; the resulting Taylor index i should be assigned to *T*. The procedure declaration consists of 3 parts:

1. The declaration of the variables *X*, *i* and *psi is free*, the arrays *TPHI*, *PHI* and *PSI* and the procedure *FACTS ABOUT FTN*.
2. Some preparatory operations: *Z* and *PHI*[*1*] become *functional variables*, with a lhs quantity equal to the Taylor index of A_j and A_i, respectively; when the procedure *Taylor index* comes across these functional variables it then knows their Taylor indexes.
   Moreover, *n th time function occurred* is augmented by 1 for the book-keeping of the functions.
3. For the different functions output is produced by means of a call of *FACTS ABOUT FTN* which first actual parameter defines the *order* of the function (for sine and cosine 2 otherwise 1), which second actual parameter defines the string *s1* to be outputted for the zeroth Taylor coefficients of Φ_1^j, and which third actual parameter defines the string *s2* to be outputted for the zeroth Taylor

coefficient of Φ_2^j (if *order* = 2).
Before *FACTS ABOUT FTN* is called, the array element *PSI*[*1*] and possibly *PHI*[*2*] and *PSI*[*2*] become equal to formulae as given in table 2 of section 5.3.3.

The procedure *FACTS ABOUT FTN* does the following:
if *first time*, it outputs the statements defining the Taylor coefficients for Φ_ν^j; it then calculates the Taylor indexes of Ψ_ν^j and stores them into the array *TPSI*
if $\neg$*first time*, it first outputs the statements for the Taylor coefficients of Φ_ν^j, where the previously calculated values of *TPSI*[*n th time function occurred*,ν] are used; it then outputs the statements for the Taylor coefficients of Ψ_ν^j.
We thus see that it is indeed necessary to introduce the non-local array *TPSI* and the variable *n th time function occurred*.
Note, that the variable *diff* is introduced in order that the procedure *PR conv product* can output a statement in which *CONV PRODUCT* occurs with such a parameter that *CONV PRODUCT* calculates in the computational program the special "differentiated" convolution product as given by equation (5.7.2).

```
procedure DEF ALGOL ST FOR FUNCTIONS(T,Ta,name of function);
  value Ta,name of function; integer T,Ta,name of function;
begin integer Z,i; Boolean psi is free; integer array TPHI,PHI,PSI[1:2];

  procedure FACTS ABOUT FTN(order,st1,st2); value order;
    integer order; string st1,st2;
  begin if first time then
    begin PR ass st for arr el(TPHI[1],PR(st1) × PR(‹(›) ×          *)
      PR zeroth array element(Ta) × PR(‹)›));
      if order = 2 then PR ass st for arr el(TPHI[2],PR(st2) × PR(‹(›) ×    *)
      PR zeroth array element(Ta) × PR(‹)›))
    end else
```

*) See the remarks on pp. 59-60.

```
    begin for i:= 1 step 1 until order do
      begin if KNOWN(Ta) then PR ass st for arr el(TPHI[i],
        PR conv product(TPSI[nth time function occurred,i],Ta,0,diff))
        else formula for [TPHI[i]]:= S(P(COEFF(PR zeroth
          array element(TPSI[nth time function occurred,i])),
          formula for [Ta]),COEFF(PR conv product(TPSI[nth time
          function occurred,i],Ta,0,diff + check for zero)))
    end end;
    for i:= 1 step 1 until order do
    TPSI[nth time function occurred,i]:= Taylor index(PSI[i],psi is free);
    comment note that if first time, the value of TPSI[...] is not
      needed. If ¬ first time this value is known;
end FACTS ABOUT FTN;

Z:= STORE(0,functional variable,Ta);
T:= TPHI[1]:= take index from free set;
PHI[1]:= STORE(0,functional variable,TPHI[1]);
nth time function occurred:= nth time function occurred + 1;

if name of function = sinf then
begin TPHI[2]:= take index from free set;
  PHI[2]:= STORE(0,functional variable,TPHI[2]);
  PSI[1]:= PHI[2]; PSI[2]:= P(CONSTANT FORMULA(minone),PHI[1]);
  FACTS ABOUT FTN(2,⊀sin⊁,⊀cos⊁)
end else

if name of function = cosf then
begin TPHI[2]:= take index from free set;
  PHI[2]:= STORE(0,functional variable,TPHI[2]);
  PSI[2]:= PHI[1]; PSI[1]:= P(CONSTANT FORMULA(minone),PHI[2]);
  FACTS ABOUT FTN(2,⊀cos⊁,⊀sin⊁)
end else
```

```
  if name of function = expf then
  begin PSI[1]:= PHI[1]; FACTS ABOUT FTN(1,⟨exp⟩,⟨⟩) end else

  if name of function = lnf then
  begin PSI[1]:= Q(CONSTANT FORMULA(one),Z);
    FACTS ABOUT FTN(1,⟨ln⟩,⟨⟩)
  end else

  if name of function = sqrtf then
  begin PSI[1]:= Q(CONSTANT FORMULA(RN(.5)),PHI[1]);
    FACTS ABOUT FTN(1,⟨sqrt⟩,⟨⟩)
  end else

  if name of function = arctanf then
  begin PSI[1]:= Q(CONSTANT FORMULA(one),S(CONSTANT FORMULA(one),
    P(Z,Z))); FACTS ABOUT FTN(1,⟨arctan⟩,⟨⟩)
end end DEF ALGOL ST FOR FUNCTIONS;
```

5.7.7. Final part of *DEFINE TAYLOR COEFFICIENTS*

Before discussing the final part of the procedure, it is necessary to make some comments on the computational program. This program is provided with two library procedures of the Mathematical Centre; the first procedure *DET* transforms a given matrix A into such a matrix that the other procedure *SOL* can easily solve the equation AX = B, where B is a known vector, for the unknown vector X. In the calculation process for the Taylor coefficients we have to solve repeatedly, for a large number of values of the index vector $(n_1, \dots, n_d)$, an equation for the unknown vector X, consisting of Taylor coefficients for the functions U_k. However, the coefficient matrix A is constant, i.e. does not depend on the index vector $(n_1, \dots, n_d)$, whereas the right-hand side vector B is not constant. This means that the computational program may transform the coefficient matrix once and for all by means of *DET*; moreover, it is not necessary, it would even be erroneous, for the computational program to compute repeatedly the coefficient matrix. Therefore, *DEFINE TAYLOR COEFFICIENTS* should provide, besides the statements already discussed, some procedure calls and some labels for the necessary jumps.

Let us illustrate this by means of the example:

$$D_1 = U_1 + \partial U_2/\partial x = 0,$$

$$D_2 = U_2 \times \partial U_1/\partial x = 0,$$

which gives rise to the following output:

```
a[3,n]:= SHIFT(2,1)              }   These are the statements for
a[3,n]:= a[3,n] + a[1,n];        }   a[k,n], with n = 0.
a[3,n]:= SHIFT(1,1)              }   (The procedures SHIFT, AUGMENT
a[4,n]:= a[2,0] × a[3,0];        }   INDEX and GOTO will be explained below.)
AGAIN: AUGMENT INDEX;
coeff[1]:= a[1,n];
RHS[1]:= -(coeff[1]);
GOTO(LABEL 2);
COEFF[1,1]:= 0;
```

```
COEFF[1,2]:= 1;
LABEL 2:
coeff[2]:= CONV PRODUCT(2,3,0,1);
coeff[3]:= a[2,0];
RHS[2]:= -(coeff[2]);
GOTO(LABEL 3);
COEFF[2,1]:= coeff[3];
COEFF[2,2]:= 0;
LABEL 3:
JUMP TO CALC OF UNKNOWNS;
FILL IN OPEN PLACES:
a[3,n]:= SHIFT(1,1)
goto  AGAIN;
OUT: end end end end end
4     3   2   1   1
```

The above output is the second part of the computational program. The first part is provided with, among others, the procedures *AUGMENT INDEX* for changing the index vector, *GOTO* for jumping to the label given as parameter only if the matrix *COEFF* has been transformed already, and *JUMP TO CALC OF UNKNOWNS* for calculating the unknown Taylor coefficients of the functions U_k and storing them.

The procedure *SHIFT(i,j)* becomes equal to

$a[i,I(n_1, \ldots, n_d)]$, if $j = 0$, and

$a[i,I(n_1, \ldots, n_{j-1}, n_j+1, n_{j+1}, \ldots, n_d)] \times (n_j+1)$, if $j > 0$.

The numbers which are finally output give the values of:

1. the number of Taylor series used,
2. the number of array elements of *coeff* used,
3. the number of differential equations, i.e. *order*,
4. the value of *type of u[k]*, k = 1, *order*.

```
 comment second part of procedure body of
    DEFINE TAYLOR COEFFICIENTS;
 first time:= true;
 for i:= 1 step 1 until order do unknown[i]:=
    STORE(Unknown,algebraic variable,type of u[i] + i × 32);
again: nth time function occurred:= pointer of set of free indexes:=
 pointer of coeff:= 0;
 next Taylor index:= order;
 for i:= 1 step 1 until order do
 begin T:= Taylor index(differential equations[new index[i]],free);
   if free then store index into free set(T);
   if ¬ first time then
   begin FIX; PR nlcr; PR(‹RHS[›); PR integral number(new index[i]);
     PR(‹]:= - (›); OUTPUT(SUBSTITUTE(formula for [T],
     k,1,order,unknown[k],zero)); PR(‹);›); ERASE;
     PR nlcr; PR(‹GOTO(LABEL›); PR integral number(i+1);
     PR(‹);›); for j:= 1 step 1 until order do
     begin FIX; PR nlcr; PR(‹COEFF[›); PR integral number(
       new index[i]); PR(‹,›); PR integral number(j); PR(‹]:= ›);
       OUTPUT(DER(formula for [T],unknown[j])); PR(‹;›); ERASE
     end; PR nlcr; PR(‹LABEL›); PR integral number(i+1); PR(‹:›)
end end;

if first time then
begin first time:= false; PR nlcr; PR(‹AGAIN: AUGMENT INDEX;›);
  goto again
end else
begin PR nlcr; PR(‹JUMP TO CALC OF UNKNOWNS;›); PR nlcr;
  PR(‹FILL IN OPEN PLACES: ›);
  for i:= order + 1 step 1 until next Taylor index do
  begin T:= set of indexes[i];
    for j:= 1 step 1 until pointer of set of free indexes do
    begin if T = set of free indexes[j] then goto out end;
```

```
          if ¬ KNOWN(T) then
          begin PR nlcr; PR array element(T); PR(⌈:= ⌉);
            OUTPUT(formula for [T]); PR(⌈;⌉);
            REPLACE(formula for [T],STORE(Taylor coefficient,
            algebraic variable,T))
          end;
        out: end
      end; PR nlcr; for i:= 126,16,126,24,126,29,126,24 do PR sym(i);
      comment the symbols goto are output;
      PR(⌈ AGAIN;⌉); PR nlcr; PR(⌈OUT: ⌉);
      for i:= 1,2,3,4,5 do for j:= 126,14,126,23,126,13,93 do PR sym(j);
      comment the symbols end end end end end are output;
      PR nlcr; RUNOUT; comment The effect of RUNOUT is
        the punching of a piece of blank tape;
      PR nlcr; PR integral number(next Taylor index); PR space(2);
      PR integral number(pointer of coeff); PR space(2);
      PR integral number(order); PR space(2);
      for i:= 1 step 1 until order do
      begin PR integral number(type of u[i]); PR space(2) end
end end DEFINE TAYLOR COEFFICIENTS;
```

5.7.8. The contents of the algebraic program

In this section we shall discuss the ALGOL 60 program in which the procedure *DEFINE TAYLOR COEFFICIENTS*, of the preceding sections, is imbedded. The program consists of:

1. The first part of the processor (section 3.3.1).
2. The general formula-manipulation system (chapter 2). In order to keep the program within the limits imposed by the available X8 computer system, some changes have been made which concern redundancies, with respect to the purposes of this chapter, only. For example, the procedures *CONVERT* and *SIMPL 2 REPR* were deleted, while, e.g., *QUOTIENT* and *COMMON DIVISOR* were changed into trivial procedures.
3. The second part of the processor of section 3.3.2, i.e. the initializing statements and declarations of section 3.3.2 and the procedures of section 3.3.2.1.
4. The declarations:

```
integer constant,dependent,independent,coefficient,check for zero,
diff,functional variable,Taylor coefficient,Unknown;
integer procedure D(a,b); value a,b; integer a,b;
D:= S(a,P(minone,b));
integer procedure POWER(a,b); value a,b; integer a,b;
POWER:= EXP(P(b,LN(a)));
```

5. The procedure *DEFINE TAYLOR COEFFICIENTS*.
6. The final part (section 5.7.8.1).

5.7.8.1. The final part of the algebraic program

As already suggested above, we assume that the differential equations are given on input tape as formulae; therefore,that part of the processor of chapter 3,which reads formulae from input tape, is contained

in the algebraic program. We assume, moreover, that the input tape has the form of a formula program, in which the "formula statement identifiers" and "special formula identifiers": REAL, SPEC DER, OUTPUT C, OUTPUT R, FIX, ERASE, ER B RET, NLCR, PR STRING, EXPAND, NOT EXP, COEFF, SOL LIN EQ, TPS, DER, SIMPL, CC, SUBST, QUOT and COMM DIV do not occur.
One new formula statement is added:
<Define Taylor Coefficients statement>::= DTC(<formula list>)
and two new primaries are added:
<independent variable>::= X(<integral arithmetic expression>)
<derivative of dependent variable>::=
    dUdX(<integral arithmetic expression>,<integral arithmetic expression>).
These primaries may be used to write

x_i as X(i),

$\frac{dU_i}{dx_j}$ as dUdX(i,j)

and U_i as dUdX(i,0).

Of course, one may use other, more appropriate, identifiers, as e.g. in the example of section 5.7.7:

```
(8192,100,0,0,0,₁₀-10,₁₀-10,7,0)
x:= X(1); u1:= dUdX(1,0); u2:= dUdX(2,0); du1dx:= dUdX(1,1);
du2dx:= dUdX(2,1);
D1:= u1 + du2dx;
D2:= u2 × du1dx;
DTC(D1,D2);
END;
```

In constructing the program such that it recognises the independent variables and the derivatives of the independent variables, we use a little trick, in that the independent variables are treated as simplified formulae and the derivatives of dependent variables as complex conjugates. We only have to change the special identifier (section 3.3) SIMPLIFY into X and the special identifier CC into dUdX. Moreover, we have to redefine

the procedures *SIMPLIFY* and *CC*, bearing in mind that *SIMPLIFY* and *CC* are called within the procedure *primary* with as actual parameter: *primary*; we, therefore, need the value of the primary read, which is an integral number. Furthermore,*CC* should read two integers, although it has only one parameter; this can be done by an extra call, within the body of *CC*, of *real number*.
Finally, in order to read a Define Taylor Coefficients statement the special identifier REAL is changed into DTC.
Besides the procedures for inputting the differential equations, we need procedures for printing and punching numbers, strings and formulae. These procedures should all be of integral type. For the output of formulae, we use the procedure *OUTPUT* as defined in section 2.13, in which the statements of lines 2 and 39, producing superfluous (with respect to the purposes of this chapter) brackets, have been removed and in which the spaces around the +, ×, and / signs have been removed also.
The procedure still produces superfluous brackets around e.g. numbers, partial sums and partial products; these brackets are, however, expressly maintained for the output of the differential equations, in order to check the transformation.
Since the procedure *OUTPUT* of section 2.13 is not of integral type, we imbed it in another procedure *output*, which itself is imbedded in a procedure *OUTPUT* of integral type. *OUTPUT*, of section 2.13, uses *ABSFIXT* and *FLOT*, which, therefore, have to be redefined in order that they print and punch.
Since the procedures *PR* and *PR int num* of section 3.3.1 are not of type integer, we declare *PR string* and *PR integral number*; moreover, *PR space* is introduced for outputting spaces. The procedure *OUTPUT VARIABLE* is declared in such a way that it outputs formulae of type *constant*, the independent and dependent variables, the derivatives of dependent variables and the "pure" algebraic variables which are used as parameters in the differential equations; ~~these~~ are algebraic variables of type *complex*. (N.B. the processor interprets them as being complex.)
Finally, we give the last part of the program in which some initializing statements occur and in which the input tape is being read.

```
integer procedure SIMPLIFY(i); integer i;
begin integer index;
   FIX; VAL OF INT NUM(i,index); ERASE;
   comment Note that within the procedure primary, the actual
     parameter call is: primary. Hence the index i is read as
     a formula, whence we need the value of this formula.;
   SIMPLIFY:= STORE(independent,algebraic variable,index)
end SIMPLIFY;

integer procedure CC(i); integer i;
begin integer index1,index2;
   FIX; NS;
   comment Note that within the procedure primary, the actual
     parameter call is: primary. Without the statement NS,
     which skips the left bracket, primary would require a
     right bracket after the first index read.;
   VAL OF INT NUM(i,index1); NS; VAL OF INT NUM(real number,
   index2); ERROR(next symbol ≠ right br,⊀) missing⊁); NS; ERASE;
   CC:= STORE(dependent,algebraic variable,index2 + index1 × 32)
end CC;

integer procedure OUTPUT(f); value f; integer f;
begin procedure flot(n,m,x); value n,m,x; integer n,m; real x;
   begin FLOT(n,m,x); FLOP(n,m,x) end flot;
   procedure output(f); value f; integer f;
   begin procedure FLOT(n,m,x); flot(n,m,x);
     comment at this place the declaration of the procedure OUTPUT
       of section 2.13 should be inserted;
     OUTPUT(f);
   end output; output(f); OUTPUT:= 1
end OUTPUT;

integer procedure PR(s); string s; begin PR string(s); PR:= 1 end;
integer procedure PR integral number(i);
begin PR int num(i); PR integral number:= 1 end;
integer procedure PR nlcr; PR nlcr:= PR(⊀
⊁);
```

```
procedure ABSFIXT(n,m,i); PR int num(i);
procedure FIXT(n,m,i); PR int num(i);
procedure PR space(n); value n; integer n;
begin SPACE(n); PUSPACE(n) end;

procedure OUTPUT VARIABLE(f); value f; integer f;
begin integer t,a,b;  t:= TYPE(f,a,b);
  if t = constant then begin OUTPUT(b); goto out end;
  ERROR(t ≠ algebraic variable,⌈error in output of formula⌉);
  if a = independent then
  begin PR(⌈X(⌉); PR int num(b); PR(⌈)⌉) end else
  if a = dependent then
  begin integer index of u,index of der; index of u:= b : 32;
    index of der:= b - index of u × 32; PR(⌈dUdX(⌉);
    PR int num(index of u); PR(⌈,⌉); PR int num(index of der);
    PR(⌈)⌉)
  end else
  if a = Taylor coefficient then
  begin PR(⌈a[⌉); PR int num(b); PR(⌈,n]⌉) end else
  if a = coefficient then
  begin PR(⌈coeff[⌉); PR int num(b); PR(⌈]⌉) end else
  if a = Unknown then
  begin integer u,d; u:= b : 32; d:= b - u × 32; if d = 0 then
    begin PR(⌈a[⌉); PR int num(u); PR(⌈,n]⌉) end else
    begin PR(⌈SHIFT(⌉); PR int num(u); PR(⌈,⌉);
      PR int num(d); PR(⌈)⌉)
  end end else
  if a = complex then
  begin integer p,sym,j; a:= 0; for j:= 0,1,2,3,4,5,6,7 do
    begin p:= 64 ↑ (3 - j + j : 4 × 4);
      sym:= (identifier list[b,j : 4] - a) : p;
      a:= if j = 3 then 0 else a + sym × p;
      if sym = 0 then goto out else PR sym(sym - 1)
  end end;
out: end OUTPUT VARIABLE;
```

```
  comment The body of the algebraic program;
  expand:= false;
  functional variable:= 11; constant:= 12; dependent:= 4; independent:= 5;
  coefficient:= 6; Taylor coefficient:= 7; Unknown:= 8; check for zero:= 1;
  diff:= 2;
  s:= RE sym; if s = underlining then s:= RE sym;
  ERROR(s ≠ right br,⊀heading not closed with ) ⊁);
  next symbol:= semi colon; line counter:= 0;
NEXT: ERROR(next symbol ≠ semi colon,⊀statement not closed with ;⊁);
  NS; s:= IDENTIFIER; if s > numb of spec id then goto Assign
  else goto CASE[s];
Assign: ERROR(next symbol ≠ colon,⊀wrong assignment st⊁); NS;
  ERROR(next symbol ≠ equal,⊀wrong assignment st⊁); NS;
  identifier list[s, -1]:= formula; goto NEXT;
Real: NS; comment The Real operator is used as the DTC operator;
  begin integer i; integer array diff eq[1:50];
    i:= 0; A: i:= i + 1; diff eq[i]:= formula;
    if next symbol = comma then begin NS; goto A end
    else ERROR(next symbol ≠ right br,⊀) missing⊁);
    NS; DEFINE TAYLOR COEFFICIENTS(i,diff eq); goto NEXT
  end;
  Spec der: Output c: Output r: Fix: Erase: Erase but retain: Nlcr:
  Pr string: Expand: Not expand: Tps coeff: Solve linear equations:
End: ERROR(true,⊀'ready⊁); comment The presence of an accent symbol
  on an input tape has as effect that the numbers following this symbol,
  i.c. line number and execution time, upto the next accent symbol are
  being skipped by the tape reader.;
end; comment the next two ends correspond to the two begins of the
general system;
end end
end end    +-×/=∧,:; ()      i.₁₀?
  DTC,SPEC DER,OUTPUT C,OUTPUT R,FIX,ERASE,ER B RET,NLCR,
  PR STRING,EXPAND,NOT EXP,COEFF,SOL LIN EQ,END+
  TPS,exp,ln,sin,cos,arctan,sqrt,DER,X,dUdX,SUBST,QUOT,COMM DIV;
```

5.7.9. An example

Consider the differential equations $f_i = 0$, $i = 1, 2$, and 3, where f_1, f_2 and f_3 are defined in the following formula program:

```
Formula program RPR 200668/03
(8192, 100, 0, 0, 0, ⏨-10, ⏨-10, 12, 0)
u1:= dUdX(1,0); u2:= dUdX(2,0); u3:= dUdX(3,0);
du2dx:= dUdX(2,1); du3dx:= dUdX(3,1); du3dy:= dUdX(3,2);
x:= X(1); y:= X(2);
f1:= (du2dx - y × u3) ↑ 11 + sin(u1 - exp(x × y));
f2:= ln(u1) - sqrt(x↑2 × y↑2) + arctan(du3dy + alpha × x × (u2 + du3dx/y));
f3:= x ↑ 2 × du2dx ↑ 2 - ( y × du3dy) ↑ 2 ;
DTC(f1,f2,f3);
END;
```

These differential equations have for $\alpha = -.5$ the simple solution:
$u_1 = u_2 = u_3 = \exp(xy)$.
Note that the types of u_1, u_2 and u_3 equal 0, 1 and 2, respectively.

The output produced is:

```
The differential equations are:
diff eq[1] = (((dUdX(2,1)+((-1)×(X(2)×dUdX(3,0)))))↑11+sin((dUdX(1,0)+((-1)×
          exp((X(1)×X(2)))))))
diff eq[2] = ((ln(dUdX(1,0))+((-1)×sqrt(((X(1))↑2×(X(2))↑2))))+arctan((dUdX(3,2)+
          ((alpha×X(1))×(dUdX(2,0)+(dUdX(3,1)/X(2)))))))
diff eq[3] = (((X(1))↑2×(dUdX(2,1))↑2)+((-1)×((X(2)×dUdX(3,2)))↑2))
The transformed differential equations are:
diff eq[3] = (((X(1))↑2×(dUdX(2,1))↑2)+((-1)×((X(2)×dUdX(3,2)))↑2))
diff eq[1] = (((dUdX(2,1)+((-1)×(dUdX(3,0)×X(2)))))↑11+sin((((-1)×exp((X(1)×X(2))))+
          dUdX(1,0))))
diff eq[2] = ((((-1)×sqrt(((X(1))↑2×(X(2))↑2)))+ln(dUdX(1,0)))+arctan((dUdX(3,2)+
          (alpha×(X(1)×(dUdX(2,0)+(dUdX(3,1)/X(2))))))));
a[4,n]:= BINOMIAL COEFF(1,2);
a[5,n]:= SHIFT(2,1);
a[6,n]:= a[5,0] × a[5,0];
a[7,n]:= a[4,0] × a[6,0];
```

```
a[8,n]:= BINOMIAL COEFF(2, 1);
a[9,n]:= SHIFT(3,2);
a[10,n]:= a[8,0] × a[9,0];
a[11,n]:= a[10,0] × a[10,0];
a[11,n]:= ((-1)) × a[11,n];
a[11,n]:= a[7,n] + a[11,n];
a[11,n]:= SHIFT(2,1);
a[7,n]:= BINOMIAL COEFF(2, 1);
a[12,n]:= a[3,0] × a[7,0];
a[12,n]:= ((-1)) × a[12,n];
a[12,n]:= a[11,n] + a[12,n];
a[11,n]:= a[12,0] × a[12,0];
a[13,n]:= a[11,0] × a[11,0];
a[14,n]:= a[13,0] × a[12,0];
a[15,n]:= a[14,0] × a[14,0];
a[16,n]:= a[15,0] × a[12,0];
a[17,n]:= BINOMIAL COEFF(1, 1);
a[18,n]:= BINOMIAL COEFF(2, 1);
a[19,n]:= a[17,0] × a[18,0];
a[20,n]:= exp(a[19,0]);
a[21,n]:= ((-1)) × a[20,n];
a[21,n]:= a[21,n] + a[1,n];
a[22,n]:= sin(a[21,0]);
a[23,n]:= cos(a[21,0]);
a[24,n]:= ((-1)) × a[22,n];
a[16,n]:= a[16,n] + a[22,n];
a[16,n]:= BINOMIAL COEFF(1,2);
a[25,n]:= BINOMIAL COEFF(2,2);
a[26,n]:= a[16,0] × a[25,0];
a[27,n]:= sqrt(a[26,0]);
a[28,n]:= (+.500000000000₁₀-  0 );
a[28,n]:= a[28,0] / a[27,0];
a[29,n]:= ((-1)) × a[27,n];
a[30,n]:= ln(a[1,0]);
a[31,n]:= 1;
a[31,n]:= a[31,0] / a[1,0];
```

```
a[29,n]:= a[29,n] + a[30,n];
a[32,n]:= SHIFT(3,2);
a[33,n]:= BINOMIAL COEFF(1, 1);
a[34,n]:= SHIFT(3,1);
a[35,n]:= BINOMIAL COEFF(2, 1);
a[34,n]:= a[34,0] / a[35,0];
a[36,n]:= a[2,n] + a[34,n];
a[37,n]:= a[33,0] × a[36,0];
a[37,n]:= (alpha) × a[37,n];
a[37,n]:= a[32,n] + a[37,n];
a[32,n]:= arctan(a[37,0]);
a[38,n]:= 1;
a[39,n]:= a[37,0] × a[37,0];
a[39,n]:= 1 + a[39,n];
a[38,n]:= a[38,0] / a[39,0];
a[29,n]:= a[29,n] + a[32,n];
AGAIN: AUGMENT INDEX;
a[4,n]:= BINOMIAL COEFF(1,2);
coeff[1]:= a[5,0];
coeff[2]:= a[5,0];
coeff[3]:= CONV PRODUCT(5,5,1,1);
coeff[4]:= CONV PRODUCT(4,6,0,1);
coeff[5]:= a[4,0];
a[8,n]:= BINOMIAL COEFF(2, 1);
coeff[6]:= CONV PRODUCT(8,9,0,1);
coeff[7]:= a[8,0];
coeff[8]:= a[10,0];
coeff[9]:= a[10,0];
coeff[10]:= CONV PRODUCT(10,10,1,1);
coeff[11]:= (-1);
RHS[3]:= - (((coeff[4]+(coeff[5]×coeff[3]))+(coeff[11]×((coeff[6]×coeff[8])+((coeff[9]×
         coeff[6])+coeff[10])))));
GOTO(LABEL2);
COEFF[3,1]:= 0;
COEFF[3,2]:= (coeff[5]×(coeff[1]+coeff[2]));
COEFF[3,3]:= (coeff[11]×((coeff[7]×coeff[8])+(coeff[9]×coeff[7])));
```

```
LABEL2:
a[7,n]:= BINOMIAL COEFF(2, 1);
a[12,n]:= CONV PRODUCT(3,7,0,0);
a[12,n]:= ((-1)) × a[12,n];
coeff[12]:= a[12,n];
coeff[13]:= a[12,0];
coeff[14]:= a[12,0];
coeff[15]:= CONV PRODUCT(12,12,1,1);
coeff[16]:= a[11,0];
coeff[17]:= a[11,0];
coeff[18]:= CONV PRODUCT(11,11,1,1);
coeff[19]:= a[12,0];
coeff[20]:= a[13,0];
coeff[21]:= CONV PRODUCT(13,12,1,1);
coeff[22]:= a[14,0];
coeff[23]:= a[14,0];
coeff[24]:= CONV PRODUCT(14,14,1,1);
coeff[25]:= a[12,0];
coeff[26]:= a[15,0];
coeff[27]:= CONV PRODUCT(15,12,1,1);
a[17,n]:= BINOMIAL COEFF(1, 1);
a[18,n]:= BINOMIAL COEFF(2, 1);
a[19,n]:= CONV PRODUCT(17,18,0,0);
a[20,n]:= CONV PRODUCT(20,19,0,-1);
a[21,n]:= ((-1)) × a[20,n];
coeff[28]:= a[21,n];
coeff[29]:= a[23,0];
coeff[30]:= CONV PRODUCT(23,21,0,-2);
coeff[31]:= a[24,0];
coeff[32]:= CONV PRODUCT(24,21,0,-2);
coeff[33]:= (-1);
RHS[1]:= - ((((((((((((coeff[12]×coeff[13])+((coeff[14]×coeff[12])+coeff[15]))×coeff[16])+
   ((coeff[17]×((coeff[12]×coeff[13])+((coeff[14]×coeff[12])+coeff[15])))+coeff[18]))×
   coeff[19])+((coeff[20]×coeff[12])+coeff[21]))×coeff[22])+((coeff[23]×((((((coeff[12]×
   coeff[13])+((coeff[14]×coeff[12])+coeff[15]))×coeff[16])+((coeff[17]×((coeff[12]×coeff[13])+
   ((coeff[14]×coeff[12])+coeff[15])))+coeff[18]))×coeff[19])+((coeff[20]×coeff[12])+
  coeff[21]))+coeff[24]))×coeff[25])+((coeff[26]×coeff[12])+coeff[27]))+((coeff[29]×
  coeff[28])+coeff[30])));
GOTO(LABEL3);
COEFF[1,1]:= coeff[29];
COEFF[1,2]:= ((((((((coeff[13]+coeff[14])×coeff[16])+(coeff[17]×(coeff[13]+coeff[14])))×
  coeff[19])+coeff[20])×coeff[22])+(coeff[23]×(((((coeff[13]+coeff[14])×coeff[16])+
  (coeff[17]×(coeff[13]+coeff[14])))×coeff[19])+coeff[20])))×coeff[25])+coeff[26]);
COEFF[1,3]:= 0;
LABEL3:
a[16,n]:= BINOMIAL COEFF(1,2);
a[25,n]:= BINOMIAL COEFF(2,2);
a[26,n]:= CONV PRODUCT(16,25,0,0);
a[27,n]:= CONV PRODUCT(28,26,0,-1);
a[28,n]:= 0;
a[28,n]:= (a[28,n] - CONV PRODUCT(28,27,1,0) )/ a[27,0];
a[29,n]:= ((-1)) × a[27,n];
coeff[34]:= a[31,0];
coeff[35]:= CONV PRODUCT(31,1,0,-2);
a[31,n]:= 0;
coeff[36]:= (a[31,n] - CONV PRODUCT(31,1,1,1)) / a[1,0];
coeff[37]:=  - a[31,0] / a[1,0];
coeff[38]:= a[29,n];
a[33,n]:= BINOMIAL COEFF(1, 1);
a[34,n]:= SHIFT(3,1);
a[35,n]:= BINOMIAL COEFF(2, 1);
a[34,n]:= (a[34,n] - CONV PRODUCT(34,35,1,0) )/ a[35,0];
a[36,n]:= a[2,n] + a[34,n];
a[37,n]:= CONV PRODUCT(33,36,0,0);
a[37,n]:= (alpha) × a[37,n];
coeff[39]:= a[37,n];
coeff[40]:= a[38,0];
coeff[41]:= CONV PRODUCT(38,37,0,-2);
a[38,n]:= 0;
coeff[42]:= a[37,0];
coeff[43]:= a[37,0];
coeff[44]:= CONV PRODUCT(37,37,1,1);
```

```
coeff[45]:= (a[38,n] - CONV PRODUCT(38,39,1,1)) / a[39,0];
coeff[46]:=  - a[38,0] / a[39,0];
RHS[2]:= - (((coeff[38]+coeff[35])+((coeff[40]×coeff[39])+coeff[41])));
GOTO(LABEL4);
COEFF[2,1]:= coeff[34];
COEFF[2,2]:= 0;
COEFF[2,3]:= coeff[40];
LABEL4:
JUMP TO CALC OF UNKNOWNS;
FILL IN OPEN PLACES:
a[5,n]:= SHIFT(2,1);
a[6,n]:= ((a[5,n]×coeff[1])+((coeff[2]×a[5,n])+coeff[3]));
a[9,n]:= SHIFT(3,2);
a[10,n]:= (coeff[6]+(coeff[7]×a[9,n]));
a[12,n]:= (coeff[12]+a[5,n]);
a[11,n]:= ((a[12,n]×coeff[13])+((coeff[14]×a[12,n])+coeff[15]));
a[13,n]:= ((a[11,n]×coeff[16])+((coeff[17]×a[11,n])+coeff[18]));
a[14,n]:= ((a[13,n]×coeff[19])+((coeff[20]×a[12,n])+coeff[21]));
a[15,n]:= ((a[14,n]×coeff[22])+((coeff[23]×a[14,n])+coeff[24]));
a[21,n]:= (coeff[28]+a[1,n]);
a[22,n]:= ((coeff[29]×a[21,n])+coeff[30]);
a[23,n]:= ((coeff[31]×a[21,n])+coeff[32]);
a[24,n]:= (coeff[33]×a[22,n]);
a[30,n]:= ((coeff[34]×a[1,n])+coeff[35]);
a[31,n]:= (coeff[36]+(coeff[37]×a[1,n]));
a[37,n]:= (coeff[39]+a[9,n]);
a[32,n]:= ((coeff[40]×a[37,n])+coeff[41]);
a[39,n]:= ((a[37,n]×coeff[42])+((coeff[43]×a[37,n])+coeff[44]));
a[38,n]:= (coeff[45]+(coeff[46]×a[39,n]));
goto AGAIN;
OUT: end end end end end

39  46  3  0  1  2
'ready
line number = 8 execution time = 21 sec
```

More examples are shown on pp. 157-160, 161-168 and 182-187.

5.8. The computational program

In this section we describe the first part of the computational program consisting of all the declarations of the necessary variables and procedures. The second part of the computational program is produced by the algebraic program of the preceding sections. The first part of the computational program should be general enough to cope with the situation that the differential equations are of arbitrary order and of arbitrary dimension (d). In particular, the arbitrary dimension causes problems: we have to determine d-dimensional index vectors; (d+1)-dimensional Taylor coefficients $a_{p,i_1, \ldots, i_d}$; d-dimensional sums (*CONV PRODUCT*).
These problems will be studied in the following sections.

Section 5.8.1 is devoted to the index function $I(d,(i_1, \ldots, i_d))$, which maps the index vectors one-to-one on the set of non-negative integers.

Section 5.8.2 is devoted to the problem of determining a new index vector j from an index vector i, assuming that the Taylor coefficients $a_{p,i}$ have just been calculated and that the $a_{p,j}$ should be calculated. The condition which must be satisfied is that all the Taylor coefficients needed for calculating the $a_{p,j}$ have been calculated already.

Section 5.8.3 is devoted to a special device which saves storage space.

Section 5.8.4 is devoted to the calculation of the convolution product, which is a d-dimensional sum.

Section 5.8.5 is devoted to the reading of the initial data.

Section 5.8.6 is devoted to the calculation of the unknown Taylor coefficients and to the general outline of the program.

Section 5.8.7 contains the reproduction of the computational program.

Section 5.8.8 is a continuation of section 5.7.9 and gives the results of the computations.

5.8.1. The index function

Consider the grid points $(\xi_1, \ldots, \xi_d)$, denoted by ξ, in a d-dimensional Euclidian space, E^d, each coordinate taking non-negative integral values only.
We shall use the following semi-norms:

$$||\xi||_k = \sum_{i=k}^{d} \xi_i \ , \quad k = 1, \ldots, d. \tag{5.8.1}$$

Consider the set $S(d,n)$, defined by:

$$S(d,n) = \{\xi \ : \ ||\xi||_1 \leq n\}. \tag{5.8.2}$$

Let the number of points in $S(d,n)$ be $M(d,n)$.
$M(d,n)$ can be determined easily as follows:
The number of points in $S(d,n)$ is equal to the number of points in $S(d,n-1)$ plus the number of points $M_1(d,n)$ in $\{\xi \ : \ ||\xi||_1 = n\}$.
However, the number of points in $\{\xi \ : \ ||\xi||_1 = n\}$ is equal to the number of points in

$$\{\overline{\xi} \ : \ \overline{\xi} = (\xi_2, \ldots, \xi_d), \ ||\xi||_2 \leq n\} = S(d-1,n).$$

Hence,

$$M(d,n) = M(d,n-1) + M(d-1,n). \tag{5.8.3}$$

This holds, of course, only if $d > 0$ and $n > 0$; however,

$$M(0,n) = M(d,0) = 1. \tag{5.8.4}$$

From (5.8.3) and (5.8.4) it follows that

$$M(d,n) = \binom{d+n}{d}. \tag{5.8.5}$$

We now determine a one-to-one mapping of all the points ξ in $S(d,n)$ onto the set of integers: $\{i \ : \ 0 \leq i < M(d,n)\}$.
The mapping will be denoted by $I(d,\xi)$, hence:

$$S(d,n) \xleftrightarrow{I} \{i \ : \ 0 \leq i \leq M(d,n) - 1\}. \tag{5.8.6}$$

The integer $I(d,\xi)$ will be called the <u>index</u> of ξ. As basic principle for I we use:
first, count the numbers in $S(d,n-1)$, where $n = ||\xi||_1$;
second, count the numbers in $T = \{\eta : ||\eta||_1 = ||\xi||_1\}$;
Since there are as much as numbers in T as there are in $S(d-1,n)$, the counting in T can be performed by $I(d-1,(\eta_2, \dots, \eta_d))$.
We now find immediately:

$$I(d,\xi) = \begin{cases} 0, \text{ if } d = 0, \text{ or if } ||\xi||_1 = 0, \\ \text{otherwise:} \\ M(d,||\xi||_1 - 1) + I(d-1,(\xi_2, \dots, \xi_d)), \end{cases} \tag{5.8.7}$$

or,

$$I(d,\xi) = M(d,||\xi||_1-1)+M(d-1,||\xi||_2-1)+\dots+M(1,||\xi||_d-1)$$

$$= \sum_{i=1}^{d} M(d-i+1,||\xi||_i - 1) \tag{5.8.8}$$

$$= \sum_{i=1}^{d} \binom{d - i + ||\xi||_i}{d - i + 1}. \tag{5.8.9}$$

5.8.2. <u>The new index vector</u>

Suppose the index vector ξ has been treated; i.e. the Taylor coefficients $a_{p,\xi}$ have been calculated, for $p = 1, \dots,$ *numb of Taylor series*.
Which index vector η should be chosen next?
Before answering this question, we make the following observations:

Let the index vector $\mu(\eta,j)$ be defined by:

$$\mu(\eta,j) = \begin{cases} (\eta_1, \eta_2, \dots, \eta_d), \text{ if } j = 0, \\ (\eta_1,\dots,\eta_{j-1},\eta_j+1,\eta_{j+1},\dots,\eta_d), \text{ if } d \geq j > 0; \end{cases} \tag{5.8.10}$$

let the index vector $\nu(\eta,j,k)$ be defined by:

$$\nu(\eta,j,k) = \begin{cases} \mu(\eta,j), \text{ if } k = 0, \\ (\eta_1, \ldots, \eta_{k-1}, \eta_k-1, \eta_{k+1}, \ldots, \eta_d), \\ \qquad\qquad \text{if } j = 0 \text{ and } d \geq k > 0, \\ \nu(\mu(\eta,j),0,k) \text{ if } d \geq j > 0 \text{ and } d \geq k > 0. \end{cases} \qquad (5.8.11)$$

Let the type of U_k be t_k; which means: $\frac{\partial U_k}{\partial x_{t_k}}$ occurs in the differential equations, but $\frac{\partial U_k}{\partial x_i}$, $i > t_k$, do not occur; if $t_k = 0$, then U_k occurs but no derivative occurs. Hence, if the index vector ξ is treated, then the Taylor coefficients

$$u_{k,\mu(\xi,t_k)}, \; k = 1, \ldots, \textit{order}, \qquad (5.8.12)$$

are calculated; moreover, the Taylor coefficients

$$u_{k,\mu(\xi,j)}, \; k = 1, \ldots, \textit{order}, \; j = 0,\ldots,t_k-1, \quad (5.8.13)$$

and the $u_{k,\eta}$, $\eta \in \{\zeta : \text{for } i = 1, \ldots, d : \zeta_i \leq \xi_i \wedge ||\zeta||_1 < ||\xi||_1\}$,

are needed for calculating the $a_{p,\xi}$ and the $u_{k,\mu(\xi,t_k)}$.

The Taylor coefficients (5.8.13) have been calculated during the treatment of the index vectors:

$$\nu(\xi,j,t_k), \; k = 1,\ldots,\textit{order}, \; j = 0,\ldots,t_k-1.$$

We thus obtain the following necessary condition for the ordering of the index vectors:

Condition: The treatment of $\nu(\xi,j,k)$, $k = 1, \ldots, d$, $0 \leq j < k$, should precede the treatment of ξ.

This condition is sufficient for a satisfactory course of the calculation process. We have derived this condition for the case: $t_k > 0$. If $t_k = 0$, however, then the $u_{k,\eta}$, with $\eta \in \{\zeta : \text{for } i = 1, \ldots, d : \zeta_i \leq \xi_i \wedge ||\zeta||_1 < ||\xi||_1\}$, are needed to calculate the $a_{p,\xi}$, and if the condition is satisfied, then

$$\nu(\xi,0,k) = (\xi_1, \ldots, \xi_{k-1}, \xi_k-1, \xi_{k+1}, \ldots, \xi_d), \; k = 1, \ldots, d,$$

is treated before ξ is treated; hence, the $u_{k,\eta}$ are indeed calculated already.

We shall now prove:

Theorem: If the ordering of the index vectors ξ is chosen such that the index $I(d,\xi)$ increases monotonically, then the above condition is satisfied at any moment during the course of the calculations.

Proof: The theorem follows immediately from the following assertion:

$$\text{If } \xi_k > 0, \text{ then } I(d,\xi) > I(d,\nu(\xi,j,k)), \quad k = 1, \ldots, d,\ 0 \le j < k. \tag{5.8.14}$$

The truth of this assertion follows from:

$$\begin{aligned} & I(d,\xi) - I(d,\nu(\xi,j,k)) = \\ & \sum_{i=1}^{d} M(d-i+1,\ ||\xi||_i - 1) - \\ & - \{ \sum_{i=k+1}^{d} M(d-i+1,\ ||\xi||_i - 1) + \\ & + \sum_{i=j+1}^{k} M(d-i+1,\ ||\xi||_i - 2) + \\ & + \sum_{i=1}^{j} M(d-i+1,\ ||\xi||_i - 1)\} = \\ & = \sum_{i=j+1}^{k} M(d-i,\ ||\xi||_i - 1); \end{aligned}$$

Since $\xi_k > 0$, it follows that $||\xi||_i - 1 \ge 0$ and

$$I(d,\xi) - I(d,\nu(\xi,j,k)) \ge k - j > 0.$$

q.e.d.

Remark: The above considerations do not pertain if $\xi_k = 0$; then, however, the initial conditions should provide the unknown $u_{p,\xi}$.

Using the theorem above, we now have to determine a process for producing a new index vector η from a given index vector ξ such that $I(d,\eta) = I(d,\xi)+1$.

Let this process be denoted by $P(d,\xi,\eta)$.
If $d = 1$, then $P(d,\xi,\eta)$ produces $\eta_d = \xi_d + 1$, which satisfies, obviously, the requirement.
Assume that $P(d-1,\overline{\xi},\overline{\eta})$ produces an $\overline{\eta}$ with $I(d-1,\overline{\eta}) = I(d-1,\overline{\xi}) + 1$; we now construct P by recursion:

1. determine $\overline{\eta} = (\eta_2, \dots, \eta_d)$ by means of $P(d-1,\overline{\xi},\overline{\eta})$, with $\overline{\xi} = (\xi_2, \dots, \xi_d)$.

2.1. Let $||\xi||_1 - ||\eta||_2 \geq 0$, then η_1 is determined from:

$$\eta_1 = ||\xi||_1 - ||\eta||_2.$$

For the value of $I(d,\eta)$ we obtain:

$$I(d,\eta) = M(d,||\eta||_1 - 1) + I(d-1,\overline{\eta});$$

from $||\eta||_1 = ||\xi||_1$, it follows that

$$I(d,\eta) = M(d,||\xi||_1 - 1) + I(d-1,\overline{\xi}) + 1$$

$$= I(d,\xi) + 1.$$

Hence, in this case η is determined.

2.2. Let now $||\xi||_1 - ||\eta||_2 < 0$, from which it follows that $||\xi||_2 < ||\eta||_2$. Since $I(d-1,\overline{\xi}) = I(d-1,\overline{\eta}) - 1$, it follows that $||\xi||_2 = ||\eta||_2 - 1$; it follows, moreover, that $I(d-1,\overline{\xi}) =$ $= M(d-1,||\xi||_2) - 1$, or, in other words, $\overline{\xi}$ is the "last" grid point in $S(d-1,||\xi||_2)$, or, of all indexes of vectors in $S(d-1,||\xi||_2)$, the vector $\overline{\xi}$ has the greatest index. This is, however, only possible if:

$$\xi_2 = \dots = \xi_{d-1} = 0 \quad \text{and} \quad \xi_d = ||\xi||_2,$$

which easily follows from (5.8.9).
From $0 > ||\xi||_1 - ||\eta||_2 = \xi_1 + ||\xi||_2 - ||\eta||_2 = \xi_1 - 1$, it follows that also $\xi_1 = 0$.
Hence, the problem is transformed into:

determine from the index vector $\xi = (0, \dots, 0, \xi_d)$ a new vector η with $I(d,\eta) = I(d,\xi) + 1$;

this problem has the trivial solution:

$$\eta = (\xi_d + 1, 0, \dots, 0).$$

For,

$$I(d,\eta) = M(d,\xi_d+1) = M(d,\xi_d) + M(d-1,\xi_d+1) =$$

$$= M(d,\xi_d) + M(d-1,\xi_d) + \dots + M(1,\xi_d) + M(0,\xi_d+1)$$

$$= I(d,\xi) + 1.$$

An ALGOL 60 procedure for the process P is now easily determined. We use the above proven fact that

$$\text{if } ||\xi||_1 - ||\eta||_2 < 0 \text{ then } \xi_1 = \dots = \xi_{d-1} = 0,$$

which enables us to test on:

$$\sum_{i=1}^{d-1} \xi_i = 0.$$

This has, by the way, the advantage that a test on d = 1 becomes superfluous.

In the above investigations, the vector ξ was equal to $(\xi_1, \dots, \xi_d)$; since d equals the fixed *dimension* on the one hand, and is also used as the "recursion parameter", on the other hand, it is necessary to refine the above arguments: the vector ξ, to be considered in the procedure below, will be equal to:

$$(\xi_j, \xi_{j+1}, \dots, \xi_{dimension}) \text{ with } j = dimension - d + 1.$$

This only means a shift of ξ.

We now give the procedure *P*:

```
procedure P(d); value d; integer d;
begin integer i,j,s; j:= dimension - d + 1;
      s:= SUM(i,j,dimension - 1,ksi[i]);
      if s = 0 then
      begin for i:= j + 1 step 1 until dimension do etha[i]:= 0;
            etha[j]:= ksi[dimension] + 1
      end else
      begin P(d-1); etha[j]:= s + ksi[dimension] -
                          SUM(i,j+1,dimension,etha[i])
      end
end P
```

which determines from a known index vector *ksi* a new index vector *etha* with

$$I(\mathit{dimension},\mathit{etha}) = I(\mathit{dimension},\mathit{ksi}) + 1.$$

The procedure *P* is used within the procedure *AUGMENT INDEX*, declared in the computational program.

5.8.3. Saving storage space

By means of the integer *highest degree* the user can specify how much Taylor coefficients he wants to be calculated by the computational program. These Taylor coefficients are:

if the type of $U_k = 0$: $u_{k,\xi}$

with $\xi \epsilon S(\mathit{dimension},\mathit{highest\ degree} - 1)$

and

if the type of $U_k > 0$: $u_{k,\xi}$

with $\xi \epsilon S(\mathit{dimension},\mathit{highest\ degree})$.

This means that we need, in principle, array elements *a*[p,n] declared by:

```
array a[1 : numb of Taylor series,
        0 : M(dimension,highest degree) - 1].
```

Note: from now on we shall write $I(\xi)$ for $I(\mathit{dimension},\xi)$.

In order to save storage space, we may use the fact that

$a[p,I(\xi)]$, $\xi \in \{\zeta : ||\zeta||_1 = \mathit{highest\ degree}\}$,
$p = 1, \dots,$ *numb of Taylor series*,

will not be needed during the calculation process for the calculation of convolution products since these use only the array elements $a[p,I(\eta)]$, with

$\eta \in \{\zeta :$ for $i = 1, \dots, \mathit{dimension} : \zeta_i \le \xi_i\}$.

The above $a[p,I(\xi)]$ are needed only to calculate the Taylor coefficients of derivatives of the unknown functions U_p, $p = 1, \dots,$ *order*. Therefore, we may declare a special array u in which these Taylor coefficients, i.e. $u_{p,\xi}$ with $||\xi||_1 =$ *highest degree*, are stored.
We thus declare:

```
array a[1 : numb of Taylor series,
        0 : M(dimension,highest degree - 1) - 1],
      u[1 : order,
        M(dimension,highest degree - 1):
        M(dimension,highest degree) - 1].
```

The saving is:

(*numb of Taylor series* − *order*) × *M*(*dimension* − 1, *highest degree*)

array elements, which, for the example of section 5.7.9 and section 5.8.9, amounts to 36% of the total storage needed. For the example of section 5.9 it amounts to 18%. In both cases this is considerable; whence this storage saving capability has been build in at the price of some extra programming (the Boolean variable *special storage* has been introduced for this purpose), which concerns the storing and extracting of the array elements in the procedures: *SHIFT*, *JUMP TO CALC OF UN-KNOWNS* and *READ INITIAL DATA*. Note that within *AUGMENT INDEX* the Boolean *special storage* gets the value true just prior to the treatment of the "first" index vector $\xi \in \{\zeta : ||\zeta||_1 = \mathit{highest\ degree} - 1\}$.

5.8.4. The convolution product

Before describing the procedure *CONV PRODUCT*, for calculating the convolution product of two sets of Taylor coefficients, we want to construct a procedure for calculating a d-dimensional sum:

$$\mathit{Sum} := \sum_{i_1=l_1}^{u_1} \cdots \sum_{i_d=l_d}^{u_d} f_{i_1,\ldots,i_d}.$$

From

$$\mathit{Sum} = \sum_{i_d=l_d}^{u_d} \left(\sum_{i_1=l_1}^{u_1} \cdots \sum_{i_{d-1}=l_{d-1}}^{u_{d-1}} f_{i_1,\ldots,i_d} \right)$$

we may construct the following procedure:

```
real procedure Sum(d,i,lb,ub,fi); value d; integer d;
   integer array i,lb,ub; real fi;
Sum:= SUM(i[d],lb[d], ub[d],
          if d = 1 then fi else
          Sum(d-1,i,lb,ub,fi));
```

We now come to the procedure *CONV PRODUCT*, with parameters *p*, *q*, *case p* and *case q*. Referring to sections 5.5 and 5.7.6, we remark that if *case p* = *case q* = 0, then

$$\mathit{CONV\ PRODUCT} := \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} a_{p,i_1,\ldots,i_d} \times a_{q,n_1-i_1,\ldots,n_d-i_d} \qquad (5.8.15)$$

with $(n_1, \ldots, n_d)$ the current index vector.

If *case p* = 1, or *case q* = 0, then the above sum does not contain the term:

$$a_{p,n_1,\ldots,n_d} \times a_{q,0,\ldots,0},$$

or, $\quad a_{p,0,\ldots,0} \times a_{q,n_1,\ldots,n_d}$, respectively.

Instead of defining a very complicated sum we just put $a_{p,n_1,\dots,n_d}$, or $a_{q,n_1,\dots,n_d}$ equal to zero. That this does not damage the calculation follows from the fact that if *case* $p = 1$, then $a_{p,n_1,\dots,n_d}$ could not be calculated already; e.g. it depends on the unknown Taylor coefficients of the U_k's and it will be calculated as soon as these unknown Taylor coefficients have been calculated (or it will never be calculated if the corresponding formula A_p is free).
The same holds for *case* $q = 1$.

If *case* $p = -1$, then

$$CONV\ PRODUCT := \left(\sum_{i_1=0}^{n_1} \dots \sum_{i_d=0}^{n_d} a_{p,i_1,\dots,i_d} \times i_k \times a_{q,n_1-i_1,\dots,n_d-i_d} \right) \frac{1}{n_k}, \tag{5.8.16}$$

and a similar formula for *case* $q = -1$.
Here k is such that $n_k \neq 0$.
These types of convolution products are used for the special functions (section 5.7.6).

If *case* $p = -2$, then the above sum (5.8.16) does not contain the term $a_{p,n_1,\dots,n_d} \times a_{q,0,\dots,0}$.
In the same way as above, $a_{p,n_1,\dots,n_d}$ is made equal to zero, beforehand, and the sum (5.8.16) is calculated.

The same situations are pertinent for *case* $q = -1$, or *case* $q = -2$.

5.8.5. <u>The initial data</u>

The input tape for the computational program should start with the numbers output by the algebraic program; i.e.

numb of Taylor series,
numb of coeff,
order,
type of u[1], ..., *type of u[order]*.

The output tape of the algebraic program contains also the "line number" and the "execution time" of the formula program. These numbers are made "invisible" for the tape reader through the occurrence of the accent symbol "'" before the word "ready".
In order that the rest of the input tape for the computational program is made visible for the tape reader, another accent symbol should precede this rest.
The next numbers on the input tape should be:

dimension,

highest degree,

$x0_1, \ldots, x0_{dimension}$,

the initial values of the unknown functions.

Note that $(x0_1, \ldots, x0_{dimension})$ determines the point in which the Taylor series should be expanded.

The situation concerning the initial values of the unknown functions, is rather complicated. Let the type of U_k be t_k, then we need, in principle:

$$u_{k,\xi}, \ \xi \in \{\zeta : \zeta_{t_k} = \ldots = \zeta_{dimension} = 0 \ \wedge \ ||\zeta||_1 \le highest\ degree\}, \quad k = 1, \ldots, order, \tag{5.8.17}$$

and

$$u_{k,\bar{\xi}}, \text{ with } \bar{\xi}_i = 0, \text{ for all } i \ne t_k \text{ and } \bar{\xi}_{t_k} = 1. \tag{5.8.18}$$

The latter Taylor coefficients are necessary to start the calculations (the equations are, in general, non-linear in these coefficients).

Instead of requiring precisely these Taylor coefficients to be put on input tape, we require the following Taylor coefficients to be put on input tape:

$$u_{k,\xi}, \ \xi \in \{\zeta : \zeta_{t_k} = 0 \wedge ||\zeta||_1 \le highest\ degree\}, \tag{5.8.19}$$

for $k = 1, \ldots, order$,

and the Taylor coefficients (5.8.18).
The Taylor coefficients contained in (5.8.19) but not contained in (5.8.17) may be given as arbitrary numbers; they are read by the program but not used.
The format of "the initial values of the unknown functions" on the input tape is the following:

$$\begin{array}{ll}
(\text{index vector } \xi^0)(\text{T coeff}_1^0)(\text{T coeff}_2^0) \ldots (\text{T coeff}_{\mathit{order}}^0) & \\
(\text{index vector } \xi^1)(\text{T coeff}_1^1)(\text{T coeff}_2^1) \ldots (\text{T coeff}_{\mathit{order}}^1) & \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & (5.8.20) \\
(\text{index vector } \xi^n)(\text{T coeff}_1^n)(\text{T coeff}_2^n) \ldots (\text{T coeff}_{\mathit{order}}^n) & \\
(\text{index vector } \xi^s)(\text{T coeff}_1^s)(\text{T coeff}_2^s) \ldots (\text{T coeff}_{\mathit{order}}^s), &
\end{array}$$

in which the index vectors ξ^i, $i = 0, \ldots, n$ are (*dimension* - 1) - dimensional index vectors with:

$$I(\mathit{dimension} - 1, \xi^{i+1}) = I(\mathit{dimension} - 1, \xi^i) + 1,$$
$$I(\mathit{dimension} - 1, \xi^0) = 0,$$

and

$$I(\mathit{dimension} - 1, \xi^n) = M(\mathit{dimension} - 1, \mathit{highest\ degree}) - 1.$$

The index vector ξ^s has components $\xi_i^s = 0$, $i = 1, \ldots,$ *dimension* - 1.
The numbers T coeff_k^i, $i = 0, \ldots, n$, $k = 1, \ldots,$ *order*, should equal

$$u_{k,\eta^i}, \text{ with } \eta_p^i = \xi_p^i,\ p = 1, \ldots, t_k - 1,$$
$$\eta_{t_k}^i = 0,$$
$$\eta_q^i = \xi_{q-1}^i,\ q = t_k + 1, \ldots, \mathit{dimension}.$$

The numbers T coeff_k^s, $k = 1, \ldots,$ *order*, should equal

$$u_{k,\eta^s}, \text{ with } \eta_p^s = 0, \text{ for all } p \neq t_k;\ \eta_{t_k}^s = 1.$$

The program tests whether the index vectors ξ^i and ξ^s are erroneous. For *dimension* = 4, *highest degree* = 3 the ξ^i's are:

i = 0	(0, 0, 0)	i=10	(3, 0, 0)
1	(1, 0, 0)	11	(2, 1, 0)
2	(0, 1, 0)	12	(2, 0, 1)
3	(0, 0, 1)	13	(1, 2, 0)
4	(2, 0, 0)	14	(1, 1, 1)
5	(1, 1, 0)	15	(1, 0, 2)
6	(1, 0, 1)	16	(0, 3, 0)
7	(0, 2, 0)	17	(0, 2, 1)
8	(0, 1, 1)	18	(0, 1, 2)
9	(0, 0, 2)	19	(0, 0, 3).

5.8.6. The computation of the unknowns

The procedure *JUMP TO CALC OF UNKNOWNS* is used to calculate the unknown Taylor coefficients, to store them and to output them. This procedure is invoked as soon as the arrays *COEFF* and *RHS* are filled. The unknowns X_i are calculated from the equation

$$COEFF \times X = RHS$$

and are stored, after a slight modification, into *u[i,shift(t)]*, or, *a[i,shift(t)]*, dependent on whether *special storage* is <u>true</u> or <u>*false*</u>. The newly calculated Taylor coefficients are then output in a format similar to (5.8.20).

Now the index vector ξ^i is a *dimension*-dimensional index vector, and

$$\text{T coeff}_k^i \text{ stands for } u_{k,\eta^i}, \text{ with } \eta^i = \mu(\xi^i, t_k).$$

The final index vector ξ^n being output is: (0, ..., 0, *highest degree* - 1).

The solution process for *COEFF* × X = *RHS*, is:
First, a call of the MC library procedure *DET* transforms *COEFF* into two matrices of lower and upper-triangular form. Second, by means of the transformed *COEFF*, the unknown X is calculated after a call of

SOL, also an MC library procedure. The calculation process has the property that *COEFF* remains constant; therefore, a call of *DET* is only necessary after the first time *COEFF* has been calculated. In fact, *COEFF* is the matrix (evaluated in the origin *x0*)

$$\left\{\frac{\partial G_l}{\partial \overline{Q}_k}\right\}, \text{ with } \overline{Q}_k = \frac{\partial U_k}{\partial x_{t_k}} \text{ (see formulae (5.1.1), (5.1.5a)),}$$

$$l = 1, \ldots, \mathit{order},\ k = 1, \ldots, \mathit{order}.$$

The linear equations in the unknowns

$$u_{k,\mu(\xi,t_k)} = u_{k,\eta}, \text{ with } \eta = (\xi_1, \ldots, \xi_{t_k-1}, \xi_{t_k}+1, \xi_{t_k+1}, \ldots, \xi_d),$$

(we write d for *dimension*)

read:

$$\sum_{k=1}^{\mathit{order}} A_{i,k}\, u_{k,\eta} = \mathit{RHS}[i],\ i = 1, \ldots, \mathit{order}, \qquad (5.8.21)$$

with

$$A_{i,k} = \frac{\partial g_{i,\xi}}{\partial u_{k,\eta}},\ i = 1, \ldots, \mathit{order},\ k = 1, \ldots, \mathit{order},$$

and with $g_{i,\xi}$ the "ξ-th" Taylor coefficient of G_i.

From

$$g_{i,\xi} = \frac{d^\xi G_i}{dx^\xi}\ \frac{1}{\xi_1! \ldots \xi_d!},\ i = 1, \ldots, \mathit{order},$$

where

$$\frac{d^\xi G_i}{dx^\xi} \overset{d}{=} \left[\sum_{k=1}^{\mathit{order}} \frac{\partial G_i}{\partial \overline{Q}_k} \cdot \frac{\partial^\eta U_k}{\partial x^\eta} + \ldots\right]_{x=x0},$$

the dots representing the other derivatives of G_i (see formula (5.2.1)), and with

$$\frac{\partial^{\eta} U_k}{\partial x^{\eta}} \stackrel{d}{=} \frac{\partial^{||\eta||_1} U_k}{\partial x_1^{\eta_1} \ldots \partial x_i^{\eta_d}},$$

and from

$$u_{k,\eta} = \left[\frac{\partial^{\eta} U_k}{\partial x^{\eta}}\right]_{x=x0} \cdot \frac{1}{\eta_1! \ldots \eta_d!}, \; k = 1, \ldots, \mathit{order},$$

it follows that

$$\mathit{COEFF}[1,k] = \frac{\partial G_1}{\partial \bar{Q}_k} = \begin{cases} A_{i,k}, \text{ if } t_k = 0, \\ (\xi_k+1)A_{i,k}, \text{ if } t_k > 0. \end{cases}$$

Hence, if the X_k's, k = 1, ..., *order*, are the components of the solution of

$$\mathit{COEFF} \times X = \mathit{RHS},$$

then

$$u_{k,\eta} = X_k/(\underline{\mathit{if}}\ t_k = 0\ \underline{\mathit{then}}\ 1\ \underline{\mathit{else}}\ \xi_{t_k}+1) \qquad (5.8.22)$$

are the components of the solution of (5.8.21).

5.8.7. The first part of the computational program

```
begin comment Computational program for the Cauchy-problem.
  RPR 080668/01. T 8190.
  At this place one may declare some extra parameters
  and give them values, as, for example;
  real alpha; alpha:= - .5;

  begin integer dimension, order, numb of Taylor series,
    numb of coeff, highest degree, n, i, m, kf;
    Boolean matrix is inverted, special storage;
    matrix is inverted:= special storage:= false;
    numb of Taylor series:= read;
    numb of coeff:= read;
    order:= read;

    begin integer array type of u[1:order]; array fac[0:100];
      for i:= 1 step 1 until order do type of u[i]:= read;
      dimension:= read; highest degree:= read; fac[0]:= 1; kf:= 0;
      begin integer array M[0:dimension, -1:highest degree],
        pivot[1:order],n sub[1:dimension];
        for i:= -1 step 1 until highest degree do M[0,i]:= 1;
        for n:= 1 step 1 until dimension do
        begin m:= M[n,-1]:= 0; n sub[n]:= 0; for i:= 0 step 1 until
          highest degree do m:= M[n, i]:= m + M[n - 1, i]
        end; n:= 0;

        begin array a[1:numb of Taylor series,
                      0:M[dimension, highest degree -1] -1],
          u[1:order, M[dimension, highest degree -1]:
                      M[dimension, highest degree] -1],
          coeff[1:numb of coeff], COEFF[1:order, 1:order],
          RHS[1:order], x0[1:dimension];
```

```
integer procedure I(ksi); integer array ksi;
if dimension = 1 then I:= ksi[1] else
if dimension = 2 then
begin integer k1, k2; k2:= ksi[2]; k1:= ksi[1] + k2;
  I:= k2 + ((k1 + 1) × k1) : 2
end else
begin integer i, j, s; s:= 0; j:= -1;
  for i:= 1 step 1 until dimension do
  begin j:= j + ksi[dimension - i + 1]; s:= s + M[i, j] end;
  I:= s
end I;

procedure AUGMENT INDEX;
begin integer i; integer array etha[1:dimension];
  procedure P(d); value d; integer d;
  begin integer i, j, s; j:= dimension - d + 1;
    s:= SUM(i, j, dimension - 1, n sub[i]);
    if s = 0 then
    begin for i:= j + 1 step 1 until dimension do
      etha[i]:= 0; etha[j]:= n sub[dimension] + 1
    end else
    begin P(d - 1); etha[j]:= s + n sub[dimension]
        - SUM(i, j + 1, dimension, etha[i])
  end end P;
  if n sub[dimension] = highest degree -2 then
  special storage:= true; P(dimension);
  for i:= 1 step 1 until dimension do n sub[i]:= etha[i];
  n:= I(n sub)
end AUGMENT INDEX;

real procedure SHIFT(i, j); value i, j; integer i, j;
SHIFT:= (if j = 0 ∨ ¬ special storage then a[i, shift(j)]
        else u[i, shift(j)]) × (n sub[j] + 1);
```

```
integer procedure shift(j); value j; integer j;
begin integer n; integer array nj[0:dimension]; nj[0]:= 0;
  for n:= 1 step 1 until dimension do nj[n]:= n sub[n];
  nj[j]:= nj[j] + 1; shift:= I(nj)
end shift;

real procedure sum(i, ub, fi); integer array i, ub; real fi;
begin real procedure loc sum(d); value d; integer d;
  begin integer ii, ubd; real s; s:= 0; ubd:= ub[d];
    for ii:= 0 step 1 until ubd do
    begin i[d]:= ii; s:= s + (if d = 1 then fi else loc sum(d - 1))
    end; loc sum:= s
  end loc sum; sum:= loc sum(dimension)
end sum;

real procedure CONV PRODUCT(p, q, case p, case q);
  value p, q, case p, case q; integer p, q, case p, case q;
begin integer k, k1; integer array i, j[1:dimension];
  integer procedure n min(i); integer array i;
  begin for k1:= 1 step 1 until dimension do
    j[k1]:= n sub[k1] - i[k1]; n min:= I(j)
  end n min i;
  if case p = +1 ∨ case p = -2 then a[p, n]:= 0;
  if case q = +1 ∨ case q = -2 then a[q, n]:= 0;
  k:= 0; if case p < 0 ∨ case q < 0 then
  begin for k1:= k + 1 while n sub[k1] = 0 do k:= k1; k:= k + 1 end;
  CONV PRODUCT:=
    if k = 0 then sum(i, n sub, a[p, I(i)] × a[q, n min(i)]) else
    if case p < 0 then
      sum(i, n sub, a[p, I(i)] × a[q, n min(i)] × i[k])/n sub[k] else
      sum(i, n sub, a[q, I(i)] × a[p, n min(i)] × i[k])/n sub[k]
end CONV PRODUCT;
```

```
real procedure BINOMIAL COEFF(p, j); value p, j; integer p, j;
begin integer i, np;
   for i:= 1 step 1 until p - 1, p + 1 step 1 until dimension do
   begin if n sub[i] ≠ 0 then
      begin BINOMIAL COEFF:= 0; goto END end
   end; np:= n sub[p]; BINOMIAL COEFF:=
   if np > j then 0 else if np = j then 1 else
   FAC(j)/(FAC(np) × FAC(j - np)) × x0[p] ↑ (j - np);
END: end BINOMIAL COEFF;
real procedure FAC(n); value n; integer n;
begin integer i; A: if n ≤ kf then FAC:= fac[n] else
   begin for i:= 1 step 1 until 5 do fac[kf + i]:=
      fac[kf + i - 1] × (kf + i); kf:= kf + 5; goto A
end end FAC;

procedure GOTO(label); label label;
if matrix is inverted then goto label;

procedure JUMP TO CALC OF UNKNOWNS;
begin integer i,t; real aux;
   if ¬ matrix is inverted then
   begin DET(COEFF, order, pivot); matrix is inverted:= true end;
   SOL(COEFF, RHS, order, pivot);
   PR nlcr; PR string(‹(›); for i:= 1 step 1 until dimension do
   begin PR int num(n sub[i]);
      if i < dimension then PR string(‹,›)
   end; PR string(‹)   ›); for i:= 1 step 1 until order do
   begin t:= type of u[i]; aux:= RHS[i]/(1 +(if t = 0 then 0
      else n sub[t])); if t ≠ 0 ∧ special storage then
      u[i, shift(t)]:= aux else a[i, shift(t)]:= aux; PR real num(aux)
   end; if n sub[dimension] = highest degree -1 then EXIT
end JUMP TO CALC OF UNKNOWNS;
```

```
procedure READ INITIAL DATA;
begin integer i, j, k, t; Boolean last;
   integer array n read, n write[0:dimension];
   for i:= 1 step 1 until dimension do
   begin x0[i]:= read; n sub[i]:= 0 end; last:= false;
AGAIN: for i:= 1 step 1 until dimension -1 do
   begin n read[i]:= read;
      if (¬last ∧ n read[i] ≠ n sub[i]) ∨ (last ∧ n read[i] ≠ 0) then
      begin PR nlcr; PR string(⊰error in initial data⊱);
         PR int num(i); PR int num(n read[i]); EXIT
   end end;
   for i:= 1 step 1 until order do
   begin t:= type of u[i]; n write[t]:= if last then 1 else 0;
      n read[0]:= 0; for k:= 1 step 1 until t - 1 do n write[k]:=
      n read[k]; for k:= t + 1 step 1 until dimension do
      n write[k]:= n read[k - 1]; k:= I(n write);
      if SUM(j, 1, dimension, n write[j]) = highest degree then
         u[i, k]:= read else a[i, k]:= read
   end;
   if last then goto END; dimension:= dimension - 1;
   AUGMENT INDEX; dimension:= dimension + 1;
   if n sub[1] ≤ highest degree then goto AGAIN;
   last:= true; goto AGAIN;
END: special storage:= false; for i:= 1 step 1 until dimension
  do n sub[i]:= 0; n:= 0
end READ INITIAL DATA;


real procedure DET(A,n,p); value n; integer n; array A;
  integer array p;
comment DET is an MC library procedure. The effect of a call
  is that A is transformed and that p is filled. A call of DET
  is necessary before SOL can be called.
  The MC procedure INPROD calculates the inner product of
  two vectors.;
begin integer i,j,k; real d,r,s; array v[1:n]; for i:= 1 step 1 until n do
  v[i]:= sqrt (INPROD (j,1,n,A[i,j],A[i,j])); d:= 1; for k:= 1 step 1
```

```
    until n do begin r:= - 1; for i:= k step 1 until n do begin A[i,k]:=
    A[i,k] - INPROD (j,1,k - 1,A[i,j],A[j,k]); s:= abs (A[i,k]) / v[i];
    if s > r then begin r:= s; p[k]:= i end end LOWER; v[p[k]]:= v[k];
    for j:= 1 step 1 until n do begin r:= A[k,j]; A[k,j]:= if j ≤ k then
    A[p[k],j] else (A[p[k],j] - INPROD (i,1,k - 1,A[k,i],A[i,j])) / A[k,k];
    if p[k] ≠ k then A[p[k],j]:= - r end UPPER; d:= A[k,k] × d
    end LU; DET:= d
  end DET;
  procedure SOL(LU,b,n,p); value n; integer n; array LU,b;
    integer array p;
  comment SOL solves the equation LU × X = b and stores the
    solution X int b.;
  begin integer i,k; real r; for k:= 1 step 1 until n do begin r:= b[k]; b[k]:=
    (b[p[k]] - INPROD (i,1,k - 1,LU[k,i],b[i])) / LU[k,k]; if p[k] ≠ k
    then b[p[k]]:= - r end; for k:= n step - 1 until 1 do b[k]:= b[k] -
    INPROD (i,k + 1,n,LU[k,i],b[i])
  end SOL;

  procedure PR string(s); string s;
  begin PUTEXT(s); PRINTTEXT(s) end;
  procedure PR int num(i); value i; integer i;
  begin ABSFIXP(2,0,i); ABSFIXT(2,0,i) end;
  procedure PR real num(r); value r; real r;
  begin FLOP(12,3,r); FLOT(12,3,r) end;
  procedure PR nlcr; PR string(⊀
⊁);
  procedure PR space(i); value i; integer i;
  begin PUSPACE(i); SPACE(i) end;

  PR nlcr; PR string(⊀results computational program RPR 080668/01⊁);
  PR nlcr; m:= (5 × dimension + 1) : 2 - 1;
  PR space(m); PR string(⊀n =⊁); PR space(5 × dimension + 1 - m);
  for i:= 1 step 1 until order do
  begin PR string(⊀u[⊁); PR int num(i); PR string(⊀,m(⊁);
    PR int num(i); PR string(⊀,n)]    ⊁)
  end; READ INITIAL DATA; comment
```

5.8.8. Example: numerical results

The following is the continuation of the example described in section 5.7.9.

First, the remainder of the input tape is reproduced; next, the results are reproduced.

Note, that m(i,n) stands for the index vector n, if $t_i = 0$, and if $t_i > 0$, for:

$$(n_1, \ldots, n_{t_i-1}, n_{t_i}+1, n_{t_i+1}, \ldots, n_{\mathit{dimension}}),$$

that $e \simeq 2.71828\ 18284\ 590$,

and that

$$U_1 = U_2 = U_3 = e^{xy} = e^{(1+\xi)(1+\eta)} =$$

$$e[1 + \xi + \eta + \frac{1}{2}\xi^2 + 2\xi\eta + \frac{1}{2}\eta^2 + \frac{1}{6}\xi^3 + \frac{3}{2}\xi^2\eta + \frac{3}{2}\xi\eta^2 + \frac{1}{6}\eta^3$$

$$+ \frac{1}{24}\xi^4 + \frac{2}{3}\xi^3\eta + \frac{7}{4}\xi^2\eta^2 + \frac{2}{3}\xi\eta^2 + \frac{1}{24}\xi^4 +$$

$$+ \frac{1}{120}\xi^5 + \frac{5}{24}\xi^4\eta + \frac{13}{12}\xi^3\eta^2 + \frac{13}{12}\xi^2\eta^3 + \frac{5}{24}\xi\eta^4 + \frac{1}{120}\xi^5 + \ldots],$$

with $\xi = 1 - x$, $\eta = 1 - y$.

```
dimension = 2 highest degree = 5
x = 1 y = 1
0       1            2.71828 18284 590      2.71828 18284 590
1       1            2.71828 18284 590      2.71828 18284 590
2       1            1.35914 09142 952      1.35914 09142 952
3       1             .45304 69714 098       .45304 69714 098
4       1             .11326 17428 525       .11326 17428 525
5       1            1                       .02265 23485 705
0 2.71828 18284 590  2.71828 18284 590      2.71828 18284 590
```

```
results computational program RPR 080668/01
  n =        u[ 1 ,m( 1 ,n)]        u[ 2 ,m( 2 ,n)]        u[ 3 ,m( 3 ,n)]
( 1 , 0 )  +.271828182846⏨+  1  +.135914091426⏨+  1  +.543656365699⏨+  1
( 0 , 1 )  +.271828182846⏨+  1  +.543656365696⏨+  1  +.135914091425⏨+  1
( 2 , 0 )  +.135914091423⏨+  1  +.453046971417⏨-  0  +.407742274278⏨+  1
( 1 , 1 )  +.543656365691⏨+  1  +.407742274275⏨+  1  +.407742274272⏨+  1
( 0 , 2 )  +.135914091423⏨+  1  +.407742274275⏨+  1  +.453046971415⏨-  0
( 3 , 0 )  +.453046971410⏨-  0  +.113261742852⏨-  0  +.181218788566⏨+  1
( 2 , 1 )  +.407742274269⏨+  1  +.181218788565⏨+  1  +.475699319985⏨+  1
( 1 , 2 )  +.407742274269⏨+  1  +.475699319985⏨+  1  +.181218788566⏨+  1
( 0 , 3 )  +.453046971410⏨-  0  +.181218788567⏨+  1  +.113261742859⏨-  0
( 4 , 0 )  +.113261742852⏨-  0  +.226523485704⏨-  1  +.566308714266⏨-  0
( 3 , 1 )  +.181218788564⏨+  1  +.566308714316⏨-  0  +.294480531420⏨+  1
( 2 , 2 )  +.475699319980⏨+  1  +.294480531420⏨+  1  +.294480531420⏨+  1
( 1 , 3 )  +.181218788564⏨+  1  +.294480531425⏨+  1  +.566308714275⏨-  0
( 0 , 4 )  +.113261742852⏨-  0  +.566308714288⏨-  0  +.226523485663⏨-  1
```

5.9. The blunt-body problem

5.9.1. Statement of the problem

The techniques, as developed in this chapter, may be applied to the important blunt-body problem in aerodynamics. Consider the flow of air around an axially symmetric body with a blunt nose; e.g. a supersonically moving projectile. We are interested in the shape of the shockfront ahead of the nose. Figure 3 illustrates the phenomenon.

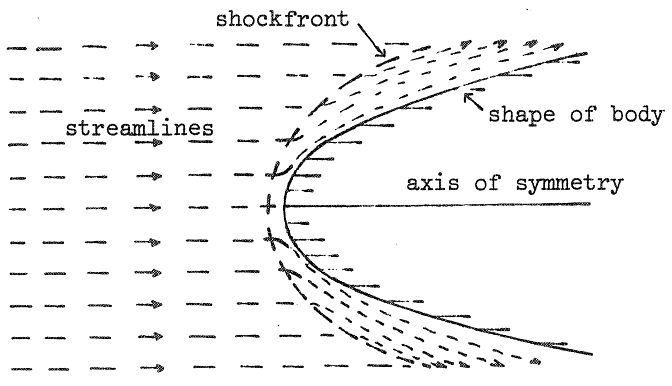

fig. 3. The blunt-body problem.

The crux in making the problem an ordinary Cauchy problem has been given by Richtmyer [13]; he inverts the problem:by assuming a certain shape of the shockfront, the shape of the body can be calculated.
By means of an iteration process, it is then possible to modify the shape of the shockfront such that the shape of the body becomes the actual shape.

We shall not be concerned with the iteration process, but rather with the first part of the problem, viz. determining the shape of the body from a given shape of the shockfront.

The equations of this section are due to C.R. Traas, of the "Nationaal Lucht- en Ruimtevaart Laboratorium" (National Aerospace Laboratory) at Amsterdam, who performed the calculations by means of finite-difference methods (see [25]).
His results could thus be compared with our results. Using three independent tests, namely:
1. Exact values are known on the axis of symmetry,
2. The entropy should remain constant,

3. An internal test (see section 5.9.4),
it turned out that our results were at least one decimal more accurate than the finite-difference results (see section 5.9.4).

The equations governing the fluid flow are the Navier-Stokes equations without viscosity, the continuity equation and the Bernoulli relation.
u and v are the horizontal, and the vertical velocity, respectively
p is the pressure and ρ is the density.

The Bernoulli relation states:

$$u^2 + v^2 + \frac{2\gamma}{\gamma-1}\frac{p}{\rho} = 1 + \frac{2}{(\gamma-1)M_\infty^2} \qquad (5.9.1)$$

where M_∞ is the Mach number of the free stream and γ the well-known adiabatic coefficient $\gamma = 1.4$. For M_∞, henceforth denoted by M, we take 4.

The coordinates are:the horizontal distance x from the shockfront and the axial distance r (see figure 4).

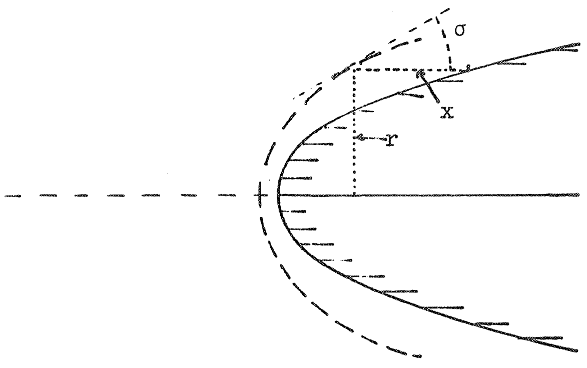

fig. 4. The coordinates x and r, and the angle σ.

The streamfunction Ψ is introduced by:

$$\frac{\partial\Psi}{\partial r} = \rho u r \ , \quad \frac{\partial\Psi}{\partial x} = -\rho v r . \qquad (5.9.2)$$

By means of (the unknown) Ψ, a new coordinate τ is introduced by:

$$\tau = \frac{2\Psi}{r^2} . \qquad (5.9.3)$$

Along the body contour $\Psi = 0$, thus $\tau = 0$ determines this contour. In the free stream $\Psi = \frac{1}{2}\rho u r^2$; taking $u = \rho = 1$, we thus have $\Psi = \frac{1}{2} r^2$ in the free stream and on the shock front. Hence, the shock front is given by $\tau = 1$.

The τ coordinate is, therefore, a very convenient coordinate; its relation with the x coordinate follows:

$$\frac{\partial x}{\partial \tau} = - r/(2\rho v). \tag{5.9.4}$$

The Navier-Stokes equations and the continuity equation, transformed into the new coordinates r and τ, now read:

$$A \times \partial U/\partial\tau = B, \tag{5.9.5}$$

where

$$A = \begin{pmatrix} \tau & 0 & 1 \\ 0 & \tau v & \tau/\rho - u \\ \rho v(1 + \frac{\gamma-1}{\gamma}\frac{\tau u}{p}) & \tau - \rho u + \frac{\gamma-1}{\gamma}\frac{\tau\rho v^2}{p} & \frac{\tau v}{p} \end{pmatrix}, \tag{5.9.6}$$

$$U = \begin{pmatrix} u \\ v \\ p \end{pmatrix} \quad \text{and} \quad B = \frac{1}{2}\begin{pmatrix} r\ \partial u/\partial r \\ r(v\,\frac{\partial v}{\partial r} + \frac{\partial p}{\partial r}/\rho) \\ \{r(\frac{\partial v}{\partial r} + \frac{v}{\rho}\frac{\partial \rho}{\partial r}) + v\} \end{pmatrix}. \tag{5.9.7}$$

The conditions on the shock front involve the angle σ as defined in figure 4; they are the Rankine-Hugoniot jump conditions:

$$\begin{aligned} u &= 1 + \frac{2}{\gamma + 1}(\frac{1}{M^2} - \sin^2\sigma) = \\ &= 1 + \frac{2}{\gamma + 1}(\frac{1}{M^2} - \frac{1}{1 + \alpha^2 r^2}), \end{aligned} \tag{5.9.8}$$

$$v = (1 - u)\text{cotan}\,\sigma = (1 - u)\alpha r, \tag{5.9.9}$$

$$p = \frac{2}{\gamma + 1} \sin^2 \sigma - \frac{\gamma - 1}{\gamma + 1} \frac{1}{\gamma M^2} =$$
$$= \frac{2}{(\gamma + 1)(1 + \alpha^2 r^2)} - \frac{\gamma - 1}{\gamma + 1} \frac{1}{\gamma M^2}, \qquad (5.9.10)$$

where α is defined as cotan $\sigma = \alpha r$; in fact, the value of α determines the shape of the shock front.

Remarks

1. Instead of relation (5.9.1) we use the, with respect to τ, differentiated relation:

$$\frac{\partial \rho}{\partial \tau} + \{\frac{1 - \gamma}{\gamma} \rho (u \frac{\partial u}{\partial \tau} + v \frac{\partial v}{\partial \tau}) - \frac{\partial p}{\partial \tau}\} \frac{\rho}{p} = 0 \qquad (5.9.11)$$

and the new dependent variable ρ. The reason is that the quantity ρ occurs several times in (5.9.5); since the algebraic program does not encounter common subformulae it would, without introduction of (5.9.11), treat each ρ anew.

2. The problem (5.9.5) is linear; we might, therefore, solve directly for the unknown $\partial U/\partial \tau$ and apply the theory of section 5.3 to it. That this is very cumbersome follows, for example, from the expression for $\partial u/\partial \tau$:

$$\frac{\partial u}{\partial \tau} = \{a_{32}(b_2 - a_{23}b_1) - a_{22}(b_3 - a_{33}b_1)\} \times$$
$$\{a_{22}(a_{33}a_{11} - a_{31}) - a_{32}a_{23}a_{11}\}^{-1}$$

in which the a_{ij} and b_i are the entries of the array A and the vector B.

5.9.2. The formula programs and their results

Formula program for the blunt-body problem.

```
Initial values RPR 150768/01
(8192, 100, 0, 0, 0, ⏨-10, ⏨-10, 13, 0)
u:= dUdX(1,0); v:= dUdX(2,0); p:= dUdX(3,0); rho:= dUdX(4,0);
r:= X(1);
aux:= 2/(ga+1)/(1 + alpha↑2 × r↑2);
f1:= u - (1 + 2/((ga+1) × Minf↑2) - aux);
f2:= v - (1 - u) × alpha × r;
f3:= p - aux + (ga-1)/(ga+1)/(ga × Minf↑2);
f4:= u × u + v × v + 2 × ga/(ga-1) × p/rho - 1 - 2/((ga-1) × Minf↑2);
DTC(f1,f2,f3,f4);
END;
```

The differential equations are:

```
diff eq[1] = (dUdX(1,0)+((-1)×((1+((2)/((ga+1)×(Minf)↑2)))+((-1)×(((2)/(ga+1))/(1+
   ((alpha)↑2×(X(1))↑2)))))))
diff eq[2] = (dUdX(2,0)+((-1)×(((1+((-1)×dUdX(1,0)))×alpha)×X(1))))
diff eq[3] = ((dUdX(3,0)+((-1)×(((2)/(ga+1))/(1+((alpha)↑2×(X(1))↑2)))))+(((ga+(-1))/
   (ga+1))/(ga×(Minf)↑2)))
diff eq[4] = (((((dUdX(1,0)×dUdX(1,0))+(dUdX(2,0)×dUdX(2,0)))+(((((2)×ga)/(ga+(-1)))×
   dUdX(3,0))/dUdX(4,0)))+(-1))+((-1)×((2)/((ga+(-1))×(Minf)↑2))))
```

The transformed differential equations are:

```
diff eq[4] = (((-1)+((-1)×((2)/((ga+(-1))×(Minf)↑2))))+(((dUdX(1,0)×dUdX(1,0))+(dUdX(2,0)
   ×dUdX(2,0)))+(((((2)×ga)/(ga+(-1)))×dUdX(3,0))/dUdX(4,0))))
diff eq[2] = ((((-1)×alpha)×((1+((-1)×dUdX(1,0)))×X(1)))+dUdX(2,0))
diff eq[3] = ((((ga+(-1))/(ga+1))/(ga×(Minf)↑2))+(((-1)×(((2)/(ga+1))/(1+((alpha)↑2×
   (X(1))↑2))))+dUdX(3,0)))
diff eq[1] = (((-1)×((1+((2)/((ga+1)×(Minf)↑2)))+((-1)×(((2)/(ga+1))/(1+((alpha)↑2×
   (X(1))↑2))))))+dUdX(1,0));
a[5,n]:= a[1,0] × a[1,0];
a[6,n]:= a[2,0] × a[2,0];
a[6,n]:= a[5,n] + a[6,n];
```

```
a[5,n]:= ((((2)×ga)/(ga+(-1)))) × a[3,n];
a[5,n]:= a[5,0] / a[4,0];
a[6,n]:= a[6,n] + a[5,n];
a[6,n]:= ((-1)+((-1)×((2)/((ga+(-1))×(Minf)↑2)))) + a[6,n];
a[6,n]:= ((-1)) × a[1,n];
a[6,n]:= 1 + a[6,n];
a[7,n]:= BINOMIAL COEFF(1, 1);
a[8,n]:= a[6,0] × a[7,0];
a[8,n]:= (((-1)×alpha)) × a[8,n];
a[8,n]:= a[8,n] + a[2,n];
a[8,n]:= ((2)/(ga+1));
a[9,n]:= BINOMIAL COEFF(1,2);
a[9,n]:= ((alpha)↑2) × a[9,n];
a[9,n]:= 1 + a[9,n];
a[8,n]:= a[8,0] / a[9,0];
a[10,n]:= ((-1)) × a[8,n];
a[10,n]:= a[10,n] + a[3,n];
a[10,n]:= (((ga+(-1))/(ga+1))/(ga×(Minf)↑2)) + a[10,n];
a[10,n]:= ((2)/(ga+1));
a[11,n]:= BINOMIAL COEFF(1,2);
a[11,n]:= ((alpha)↑2) × a[11,n];
a[11,n]:= 1 + a[11,n];
a[10,n]:= a[10,0] / a[11,0];
a[12,n]:= ((-1)) × a[10,n];
a[12,n]:= (1+((2)/((ga+1)×(Minf)↑2))) + a[12,n];
a[12,n]:= ((-1)) × a[12,n];
a[12,n]:= a[12,n] + a[1,n];
AGAIN: AUGMENT INDEX;
coeff[1]:= a[1,0];
coeff[2]:= a[1,0];
coeff[3]:= CONV PRODUCT(1,1,1,1);
coeff[4]:= a[2,0];
coeff[5]:= a[2,0];
coeff[6]:= CONV PRODUCT(2,2,1,1);
coeff[7]:= (((2)×ga)/(ga+(-1)));
```

```
coeff[8]:=  - a[5,0];
coeff[9]:=  - CONV PRODUCT(5,4,1,1);
coeff[10]:= 1/ a[4,0];
RHS[4]:= - (((coeff[3]+coeff[6])+(coeff[9]×coeff[10])));
GOTO(LABEL2);
COEFF[4,1]:= (coeff[1]+coeff[2]);
COEFF[4,2]:= (coeff[4]+coeff[5]);
COEFF[4,3]:= (coeff[7]×coeff[10]);
COEFF[4,4]:= (coeff[8]×coeff[10]);
LABEL2:
coeff[11]:= (-1);
a[7,n]:= BINOMIAL COEFF(1, 1);
coeff[12]:= CONV PRODUCT(7,6,0,1);
coeff[13]:= a[7,0];
coeff[14]:= ((-1)×alpha);
RHS[2]:= - ((coeff[14]×coeff[12]));
GOTO(LABEL3);
COEFF[2,1]:= (coeff[14]×(coeff[13]×coeff[11]));
COEFF[2,2]:= 1;
COEFF[2,3]:= 0;
COEFF[2,4]:= 0;
LABEL3:
a[8,n]:= 0;
a[9,n]:= BINOMIAL COEFF(1,2);
a[9,n]:= ((alpha)↑2) × a[9,n];
a[9,n]:= a[9,n];
a[8,n]:= (a[8,n] - CONV PRODUCT(8,9,1,0) )/ a[9,0];
a[10,n]:= ((-1)) × a[8,n];
coeff[15]:= a[10,n];
RHS[3]:= - (coeff[15]);
GOTO(LABEL4);
COEFF[3,1]:= 0;
COEFF[3,2]:= 0;
COEFF[3,3]:= 1;
COEFF[3,4]:= 0;
```

```
LABEL4:
a[10,n]:= 0;
a[11,n]:= BINOMIAL COEFF(1,2);
a[11,n]:= ((alpha)↑2) × a[11,n];
a[11,n]:= a[11,n];
a[10,n]:= (a[10,n] - CONV PRODUCT(10,11,1,0) )/ a[11,0];
a[12,n]:= ((-1)) × a[10,n];
a[12,n]:= a[12,n];
a[12,n]:= ((-1)) × a[12,n];
coeff[16]:= a[12,n];
RHS[1]:= - (coeff[16]);
GOTO(LABEL5);
COEFF[1,1]:= 1;
COEFF[1,2]:= 0;
COEFF[1,3]:= 0;
COEFF[1,4]:= 0;
LABEL5:
JUMP TO CALC OF UNKNOWNS;
FILL IN OPEN PLACES:
a[5,n]:= (((coeff[7]×a[3,n])+((coeff[8]×a[4,n])+coeff[9]))×coeff[10]);
a[6,n]:= (coeff[11]×a[1,n]);
goto AGAIN;
OUT: end end end end end

12  16  4  0  0  0  0
'ready
line number = 9 execution time = 15 sec
```

Formula program for the blunt-body problem.
Main values RPR 150768/02

```
(8192, 100, 0, 0, 0, ⏨-10, ⏨-10, 28, 0)
u:= dUdX(1,0); v:= dUdX(2,0); p:= dUdX(3,0); rho:= dUdX(4,0);
r:= X(1); tau:= X(2);
dudr:= dUdX(1,1); dvdr:= dUdX(2,1); dpdr:= dUdX(3,1); drhodr:= dUdX(4,1);
dudt:= dUdX(1,2); dvdt:= dUdX(2,2); dpdt:= dUdX(3,2); drhodt:= dUdX(4,2);
A11:= tau;        A12:= 0;          A13:= 1;
A21:= 0;          A22:= tau × v;    A23:= tau/rho - u;
A31:= rho × v × (1 + (ga-1)/ga × (tau × u/p));
A32:= tau - rho × u + (ga-1)/ga × (tau × rho × v × v/p);
A33:= tau × v/p;
f1:= A11 × dudt + A12 × dvdt + A13 × dpdt - .5 ×
        r × dudr;
f2:= A21 × dudt + A22 × dvdt + A23 × dpdt - .5 ×
        r × (v × dvdr + dpdr/rho);
f3:= A31 × dudt + A32 × dvdt + A33 × dpdt - .5 ×
        (r × (dvdr + v × drhodr/rho) + v);
f4:= drhodt + ((1-ga)/ga × rho × (u × dudt + v × dvdt) - dpdt) × rho/p;
DTC(f1,f2,f3,f4);
END;
```

The differential equations are:

```
diff eq[1] = (((X(2)×dUdX(1,2))+dUdX(3,2))+((-1)×(((+.500000000000⏨-  0 )×X(1))×
  dUdX(1,1))))
diff eq[2] = ((((X(2)×dUdX(2,0))×dUdX(2,2))+(((X(2)/dUdX(4,0))+((-1)×dUdX(1,0)))×
  dUdX(3,2)))+((-1)×(((+.500000000000⏨-  0 )×X(1))×((dUdX(2,0)×dUdX(2,1))+(dUdX(3,1)
  /dUdX(4,0))))))
diff eq[3] = ((((((dUdX(4,0)×dUdX(2,0))×(1+(((ga+(-1))/ga)×((X(2)×dUdX(1,0))/dUdX(3,0)))
  ))×dUdX(1,2))+(((X(2)+((-1)×(dUdX(4,0)×dUdX(1,0))))+(((ga+(-1))/ga)×((((X(2)×dUdX(4,0)
  )×dUdX(2,0))×dUdX(2,0))/dUdX(3,0))))×dUdX(2,2)))+(((X(2)×dUdX(2,0))/dUdX(3,0))×
  dUdX(3,2)))+((-1)×((+.500000000000⏨-  0 )×((X(1)×(dUdX(2,1)+((dUdX(2,0)×dUdX(4,1)
  )/dUdX(4,0))))+dUdX(2,0)))))
diff eq[4] = (dUdX(4,2)+((((((1+((-1)×ga))/ga)×dUdX(4,0))×((dUdX(1,0)×dUdX(1,2))+
  (dUdX(2,0)×dUdX(2,2))))+((-1)×dUdX(3,2)))×dUdX(4,0))/dUdX(3,0)))
```

The transformed differential equations are:

```
diff eq[1] = ((dUdX(3,2)+(X(2)×dUdX(1,2)))+(((-1)×(+.500000000000_{10}-  0 ))×(X(1)×
   dUdX(1,1))))
diff eq[2] = (((dUdX(2,0)×(X(2)×dUdX(2,2)))+(dUdX(3,2)×(((-1)×dUdX(1,0))+(X(2)/dUdX
   (4,0)))))+(((-1)×(+.500000000000_{10}-  0 ))×(X(1)×((dUdX(2,0)×dUdX(2,1))+(dUdX(3,1)/
   dUdX(4,0))))))
diff eq[3] = ((((dUdX(3,2)×((dUdX(2,0)×X(2))/dUdX(3,0)))+(((-1)×(+.500000000000_{10}-  0 )
   )×((X(1)×(dUdX(2,1)+((dUdX(2,0)×dUdX(4,1))/dUdX(4,0))))+dUdX(2,0))))+(dUdX(4,0)×
   (dUdX(2,0)×(dUdX(1,2)×(1+(((ga+(-1))/ga)×((dUdX(1,0)×X(2))/dUdX(3,0))))))))+
   (((X(2)+((-1)×(dUdX(4,0)×dUdX(1,0))))+(((ga+(-1))/ga)×((dUdX(4,0)×(dUdX(2,0)×(dUdX(
   2,0)×X(2))))/dUdX(3,0))))×dUdX(2,2)))
diff eq[4] = (dUdX(4,2)+(((((-1)×dUdX(3,2))+(((1+((-1)×ga))/ga)×(((dUdX(1,0)×dUdX(1,2)
   )+(dUdX(2,0)×dUdX(2,2)))×dUdX(4,0))))×dUdX(4,0))/dUdX(3,0)));
a[5,n]:= SHIFT(3,2);
a[6,n]:= BINOMIAL COEFF(2, 1);
a[7,n]:= SHIFT(1,2);
a[8,n]:= a[6,0] × a[7,0];
a[8,n]:= a[5,n] + a[8,n];
a[5,n]:= BINOMIAL COEFF(1, 1);
a[9,n]:= SHIFT(1,1);
a[10,n]:= a[5,0] × a[9,0];
a[10,n]:= (((-1)×(+.500000000000_{10}-  0 ))) × a[10,n];
a[10,n]:= a[8,n] + a[10,n];
a[10,n]:= BINOMIAL COEFF(2, 1);
a[8,n]:= SHIFT(2,2);
a[11,n]:= a[10,0] × a[8,0];
a[12,n]:= a[2,0] × a[11,0];
a[13,n]:= SHIFT(3,2);
a[14,n]:= ((-1)) × a[1,n];
a[15,n]:= BINOMIAL COEFF(2, 1);
a[15,n]:= a[15,0] / a[4,0];
a[14,n]:= a[14,n] + a[15,n];
a[16,n]:= a[13,0] × a[14,0];
a[16,n]:= a[12,n] + a[16,n];
a[12,n]:= BINOMIAL COEFF(1, 1);
```

```
a[17,n]:= SHIFT(2,1);
a[18,n]:= a[2,0] × a[17,0];
a[19,n]:= SHIFT(3,1);
a[19,n]:= a[19,0] / a[4,0];
a[18,n]:= a[18,n] + a[19,n];
a[20,n]:= a[12,0] × a[18,0];
a[20,n]:= (((-1)×(+.500000000000₁₀-  0 ))) × a[20,n];
a[20,n]:= a[16,n] + a[20,n];
a[20,n]:= SHIFT(3,2);
a[16,n]:= BINOMIAL COEFF(2, 1);
a[21,n]:= a[2,0] × a[16,0];
a[21,n]:= a[21,0] / a[3,0];
a[22,n]:= a[20,0] × a[21,0];
a[23,n]:= BINOMIAL COEFF(1, 1);
a[24,n]:= SHIFT(2,1);
a[25,n]:= SHIFT(4,1);
a[26,n]:= a[2,0] × a[25,0];
a[26,n]:= a[26,0] / a[4,0];
a[24,n]:= a[24,n] + a[26,n];
a[27,n]:= a[23,0] × a[24,0];
a[27,n]:= a[27,n] + a[2,n];
a[27,n]:= (((-1)×(+.500000000000₁₀-  0 ))) × a[27,n];
a[27,n]:= a[22,n] + a[27,n];
a[22,n]:= SHIFT(1,2);
a[28,n]:= BINOMIAL COEFF(2, 1);
a[29,n]:= a[1,0] × a[28,0];
a[29,n]:= a[29,0] / a[3,0];
a[30,n]:= (((ga+(-1))/ga)) × a[29,n];
a[30,n]:= 1 + a[30,n];
a[31,n]:= a[22,0] × a[30,0];
a[32,n]:= a[2,0] × a[31,0];
a[33,n]:= a[4,0] × a[32,0];
a[33,n]:= a[27,n] + a[33,n];
a[27,n]:= BINOMIAL COEFF(2, 1);
a[34,n]:= a[4,0] × a[1,0];
```

```
a[34,n]:= ((-1)) × a[34,n];
a[34,n]:= a[27,n] + a[34,n];
a[27,n]:= BINOMIAL COEFF(2, 1);
a[35,n]:= a[2,0] × a[27,0];
a[36,n]:= a[2,0] × a[35,0];
a[37,n]:= a[4,0] × a[36,0];
a[37,n]:= a[37,0] / a[3,0];
a[38,n]:= (((ga+(-1))/ga)) × a[37,n];
a[38,n]:= a[34,n] + a[38,n];
a[34,n]:= SHIFT(2,2);
a[39,n]:= a[38,0] × a[34,0];
a[39,n]:= a[33,n] + a[39,n];
a[39,n]:= SHIFT(4,2);
a[33,n]:= SHIFT(3,2);
a[33,n]:= ((-1)) × a[33,n];
a[40,n]:= SHIFT(1,2);
a[41,n]:= a[1,0] × a[40,0];
a[42,n]:= SHIFT(2,2);
a[43,n]:= a[2,0] × a[42,0];
a[43,n]:= a[41,n] + a[43,n];
a[41,n]:= a[43,0] × a[4,0];
a[41,n]:= (((1+((-1)×ga))/ga)) × a[41,n];
a[41,n]:= a[33,n] + a[41,n];
a[33,n]:= a[41,0] × a[4,0];
a[33,n]:= a[33,0] / a[3,0];
a[39,n]:= a[39,n] + a[33,n];
AGAIN: AUGMENT INDEX;
comment The following label has been inserted by hand;
AGAIN2:
a[6,n]:= BINOMIAL COEFF(2, 1);
coeff[1]:= CONV PRODUCT(6,7,0,1);
coeff[2]:= a[6,0];
a[5,n]:= BINOMIAL COEFF(1, 1);
a[9,n]:= SHIFT(1,1);
a[10,n]:= CONV PRODUCT(5,9,0,0);
```

```
a[10,n]:= (((-1)×(+.500000000000₁₀-  0 ))) × a[10,n];
coeff[3]:= a[10,n];
RHS[1]:= - ((coeff[3]+coeff[1]));
GOTO(LABEL2);
COEFF[1,1]:= coeff[2];
COEFF[1,2]:= 0;
COEFF[1,3]:= 1;
COEFF[1,4]:= 0;
LABEL2:
a[10,n]:= BINOMIAL COEFF(2, 1);
coeff[4]:= CONV PRODUCT(10,8,0,1);
coeff[5]:= a[10,0];
coeff[6]:= CONV PRODUCT(2,11,0,1);
coeff[7]:= a[2,0];
a[14,n]:= ((-1)) × a[1,n];
a[15,n]:= BINOMIAL COEFF(2, 1);
a[15,n]:= (a[15,n] - CONV PRODUCT(15,4,1,0) )/ a[4,0];
a[14,n]:= a[14,n] + a[15,n];
coeff[8]:= CONV PRODUCT(14,13,0,1);
coeff[9]:= a[14,0];
a[12,n]:= BINOMIAL COEFF(1, 1);
a[17,n]:= SHIFT(2,1);
a[18,n]:= CONV PRODUCT(2,17,0,0);
a[19,n]:= SHIFT(3,1);
a[19,n]:= (a[19,n] - CONV PRODUCT(19,4,1,0) )/ a[4,0];
a[18,n]:= a[18,n] + a[19,n];
a[20,n]:= CONV PRODUCT(12,18,0,0);
a[20,n]:= (((-1)×(+.500000000000₁₀-  0 ))) × a[20,n];
coeff[10]:= a[20,n];
RHS[2]:= - ((coeff[10]+((coeff[6]+(coeff[7]×coeff[4]))+coeff[8])));
GOTO(LABEL3);
COEFF[2,1]:= 0;
COEFF[2,2]:= (coeff[7]×coeff[5]);
COEFF[2,3]:= coeff[9];
COEFF[2,4]:= 0;
```

```
LABEL3:
a[16,n]:= BINOMIAL COEFF(2, 1);
a[21,n]:= CONV PRODUCT(2,16,0,0);
a[21,n]:= (a[21,n] - CONV PRODUCT(21,3,1,0) )/ a[3,0];
coeff[11]:= CONV PRODUCT(21,20,0,1);
coeff[12]:= a[21,0];
a[23,n]:= BINOMIAL COEFF(1, 1);
a[24,n]:= SHIFT(2,1);
a[25,n]:= SHIFT(4,1);
a[26,n]:= CONV PRODUCT(2,25,0,0);
a[26,n]:= (a[26,n] - CONV PRODUCT(26,4,1,0) )/ a[4,0];
a[24,n]:= a[24,n] + a[26,n];
a[27,n]:= CONV PRODUCT(23,24,0,0);
a[27,n]:= a[27,n] + a[2,n];
a[27,n]:= (((-1)×(+.500000000000₁₀-  0 ))) × a[27,n];
coeff[13]:= a[27,n];
a[28,n]:= BINOMIAL COEFF(2, 1);
a[29,n]:= CONV PRODUCT(1,28,0,0);
a[29,n]:= (a[29,n] - CONV PRODUCT(29,3,1,0) )/ a[3,0];
a[30,n]:= (((ga+(-1))/ga)) × a[29,n];
comment The following statement has been changed by hand from:
a[30,n]:= a[30,n], into:;
a[30,n]:= (if n = 0 then 1 else 0) + a[30,n];
coeff[14]:= CONV PRODUCT(30,22,0,1);
coeff[15]:= a[30,0];
coeff[16]:= CONV PRODUCT(2,31,0,1);
coeff[17]:= a[2,0];
coeff[18]:= CONV PRODUCT(4,32,0,1);
coeff[19]:= a[4,0];
a[27,n]:= BINOMIAL COEFF(2, 1);
a[34,n]:= CONV PRODUCT(4,1,0,0);
a[34,n]:= ((-1)) × a[34,n];
a[34,n]:= a[27,n] + a[34,n];
a[27,n]:= BINOMIAL COEFF(2, 1);
```

```
a[35,n]:= CONV PRODUCT(2,27,0,0);
a[36,n]:= CONV PRODUCT(2,35,0,0);
a[37,n]:= CONV PRODUCT(4,36,0,0);
a[37,n]:= (a[37,n] - CONV PRODUCT(37,3,1,0) )/ a[3,0];
a[38,n]:= (((ga+(-1))/ga)) × a[37,n];
a[38,n]:= a[34,n] + a[38,n];
coeff[20]:= CONV PRODUCT(38,34,0,1);
coeff[21]:= a[38,0];
RHS[3]:= - ((((coeff[13]+coeff[11])+(coeff[18]+(coeff[19]×(coeff[16]+(coeff[17]×
  coeff[14])))))+coeff[20]));
GOTO(LABEL4);
COEFF[3,1]:= (coeff[19]×(coeff[17]×coeff[15]));
COEFF[3,2]:= coeff[21];
COEFF[3,3]:= coeff[12];
COEFF[3,4]:= 0;
LABEL4:
coeff[22]:= (-1);
coeff[23]:= CONV PRODUCT(1,40,0,1);
coeff[24]:= a[1,0];
coeff[25]:= CONV PRODUCT(2,42,0,1);
coeff[26]:= a[2,0];
coeff[27]:= CONV PRODUCT(4,43,0,1);
coeff[28]:= a[4,0];
coeff[29]:= ((1+((-1)×ga))/ga);
coeff[30]:= CONV PRODUCT(4,41,0,1);
coeff[31]:= a[4,0];
coeff[32]:=  - CONV PRODUCT(33,3,1,0);
coeff[33]:= 1/ a[3,0];
RHS[4]:= - ((((coeff[30]+(coeff[31]×(coeff[29]×(coeff[27]+(coeff[28]×(coeff[23]+
  coeff[25]))))))+coeff[32])×coeff[33]));
GOTO(LABEL5);
COEFF[4,1]:= ((coeff[31]×(coeff[29]×(coeff[28]×coeff[24])))×coeff[33]);
COEFF[4,2]:= ((coeff[31]×(coeff[29]×(coeff[28]×coeff[26])))×coeff[33]);
COEFF[4,3]:= ((coeff[31]×coeff[22])×coeff[33]);
COEFF[4,4]:= 1;
```

```
LABEL5:
JUMP TO CALC OF UNKNOWNS;
FILL IN OPEN PLACES:
a[7,n]:= SHIFT(1,2);
a[8,n]:= SHIFT(2,2);
a[11,n]:= (coeff[4]+(coeff[5]×a[8,n]));
a[13,n]:= SHIFT(3,2);
a[20,n]:= a[13,n];
a[22,n]:= a[7,n];
a[31,n]:= (coeff[14]+(coeff[15]×a[22,n]));
a[32,n]:= (coeff[16]+(coeff[17]×a[31,n]));
a[34,n]:= a[8,n];
a[40,n]:= a[22,n];
a[42,n]:= a[34,n];
a[43,n]:= ((coeff[23]+(coeff[24]×a[40,n]))+(coeff[25]+(coeff[26]×a[42,n])));
a[41,n]:= ((coeff[22]×a[20,n])+(coeff[29]×(coeff[27]+(coeff[28]×a[43,n]))));
a[33,n]:= (((coeff[30]+(coeff[31]×a[41,n]))+coeff[32])×coeff[33]);
goto AGAIN;
OUT: end end end end end

43  33  4  2  2  2  2
'ready
line number = 18 execution time = 27 sec
```

5.9.3. The computations

Computational programs were constructed for the initial values (RPR 170768/01) and for the main values (RPR 180768/01). The output of the former served as the input of the latter program.

By means of RPR 170768/01, the Taylor series of $u(r,1)$, $v(r,1)$, $p(r,1)$ and $\rho(r,1)$ were calculated upto ten's order in the points:
$r = .03(.12).87$, on the line $\tau = 1$.

By means of RPR 180768/01, the Taylor series of $u(r,1)$, $v(r,1)$, $p(r,1)$ $p(r,\tau)$ and $\rho(r,\tau)$ were calculated upto ten's order in the points:

a_1: $\tau = 1$, $r = .03(.12).87$,
a_2: $\tau = .8$, $r = .03(.12).87$,
a_3: $\tau = .6$, $r = .03(.12).87$,
a_4: $\tau = .4$, $r = .03(.12).87$,
a_5: $\tau = .2$, $r = .03(.12).87$.

The initial values for the computation of the Taylor series in the points under a_1 followed from the results of RPR 170768/01.
The initial values for the computation of the Taylor series in the points under a_i, $i = 2, 3, 4, 5$, followed from the results of the computations in the points under a_{i-1}; the following formula has been used for that purpose:

$$\overline{f}_{i,0} = \sum_{j=0}^{h-i} f_{i,j}(-.2)^j, \text{ (h = \textit{highest degree})}$$

where $f_{i,j}$ is the (i,j)-th Taylor coefficient of $F(r,\tau)$ in r_0, τ_0 and $\overline{f}_{i,0}$ is the $(i,0)$-th Taylor coefficient of $F(r,\tau)$ in r_0, $\tau_0-.2$.

The process, which is a sort of analytic continuation is illustrated in figure 5.

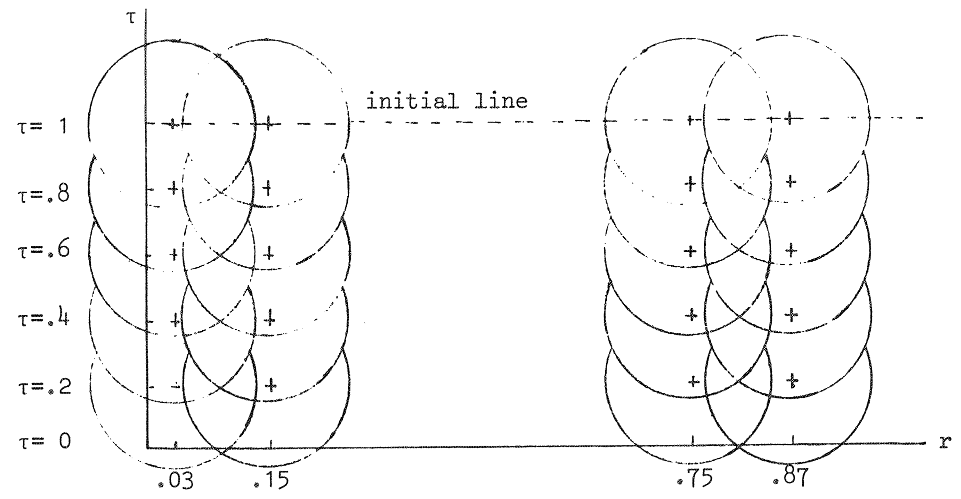

fig. 5. The "analytic" continuation process.

Instead of the Taylor coefficients themselves, the values of u, v, p, ρ and p/ρ^{γ}, calculated in the points:

$r = 0,\ \tau = 1(-.2)0$

and $\tau = 0,\ r = .03(.12).87$, will be reproduced (in section 5.9.4). These calculations were performed using the Taylor series expanded in points near the above points.

A final, rather important, remark should be made. As described in section 5.8.5, and as required by condition 3 in section 5.1, the values of the (0,1)-st Taylor coefficients of u, v, p and ρ are explicitly needed before the calculations can be started.These values should, therefore, be calculated beforehand. As has been observed already, the equations (5.9.5) and (5.9.11) are linear in $\partial u/\partial\tau$, $\partial v/\partial\tau$, $\partial p/\partial\tau$ and $\partial\rho/\partial\tau$; and are,hence,linear in these (0,1)-st Taylor coefficients. The coefficients of these equations should be calculated. This may be done, of course, by 20 times repeated execution of the statements for the initial Taylor coefficients (see section 5.5 p. 62); however, it may also be done by execution of the statements for the non-initial Taylor coefficients, which are altered a little bit.

The only point we have to take care of concerns a formula of the form "constant + f", where 'f" is a non-constant formula. The algebraic program outputs the 'constant" only in the statements for the initial Taylor coefficients. We, therefore, have to change the corresponding statement for the non-initial Taylor coefficient.
For the problem considered, this is the statement

a[30,n]:= a[30,n] on line 22 of page 166

which has been changed into:

a[30,n]:= a[30,n] + (if n = 0 then 1 else 0).

In order to skip the, now superfluous, statements for the initial values, a new label *AGAIN2* has been inserted right after the label *AGAIN*; and the computations start by a direct jump to *AGAIN2*.

It is obvious that these tricks may be used in any linear problem.

Those parts of the computational programs which differ from the corresponding parts of the computational program of section 5.8.7 will now be reproduced.
These parts concern the heading, and the body of the program, i.e. the last 7 lines on p. 150, and a new procedure *EXIT*.

```
begin comment Computational program for the blunt-body problem.
  Initial values RPR 170768/01;
  real Minf,ga,alpha,aux;
  Minf:= 4; ga:= 1.4; alpha:= .70533;

  procedure EXIT; goto ANEW;

  comment Next follows the new body of the program.;
  PR nlcr; PR string(⟨results computational program for the blunt-
  body problem. Initial values.⟩); goto anew;
```

```
ANEW: PR nlcr; PR string(⌈(0), 1, 1, 1, 1⌉);
anew: x0[1]:= read; if x0[1] < 0 then begin PR nlcr; goto OUT end;
  PR nlcr; PR string(⌈r = ⌉); PR real num(x0[1]);
  PR string(⌈tau = 1⌉); PR nlcr;
  PR string(⌈the values of the initial Taylor coefficients of
  u, v, p and rho are:⌉);
  aux:= 2/((ga+1) × (1 + alpha↑2 × x0[1]↑2));
  a[1,0]:= 1 + 2/((ga+1) × Minf↑2) - aux;
  a[2,0]:= (1 - a[1,0]) × alpha × x0[1];
  a[3,0]:= aux - (ga-1)/((ga+1) × ga × Minf↑2);
  a[4,0]:= 2 × ga × a[3,0]/((ga-1) × (1 + 2/((ga-1) × Minf↑2)
    - a[1,0]↑2 - a[2,0]↑2));
  PR nlcr; PR string(⌈(  0 )  ⌉); for i:= 1,2,3,4 do PR real num(a[i,0]);
  PR nlcr; n sub[1]:= 0; n:= 0;
  matrix is inverted:= special storage:= false;
  comment
```

The input tape reads:

```
'dimension = 1
highest degree = 11
x = .03, .15, .27, .39, .51, .63, .75, .87, -1
```

```
begin comment Computational program for the blunt-body problem.
   Main values RPR 180768/01;
   real ga; ga:= 1.4;

   procedure EXIT; goto ANEW;

   comment Next follows the new body of the program.;
   PR nlcr; PR string(⊀results computational program for the blunt-body
   problem. Main values.⊁);
BEGIN: READ INITIAL DATA; goto START;
ANEW:
   begin real x,y; integer j,k; array U[1:4];
     integer procedure I1(j,k); integer j,k;
     begin n sub[1]:= j; n sub[2]:= k; I1:= I(n sub) end;
     procedure print;
     begin PR nlcr; FIXT(0,2,x); FIXP(0,2,x); FIXT(1,1,y); FIXP(1,1,y);
       for i:= 1,2,3,4 do
       begin U[i]:= SUM(j,0,highest degree, SUM(k,0,highest degree - j,
         (if k + j = highest degree then u[i,I1(j,k)] else a[i,I1(j,k)]) ×
           (x - x0[1]) ↑ j × (y - x0[2]) ↑ k));
         FLOT(8,2,U[i]); FLOP(8,2,U[i])
       end; FLOT(8,2,U[3]/(U[4] ↑ ga)); FLOP(8,2,U[3]/(U[4] ↑ ga))
     end print;
     if x0[1] < .16 ∨ x0[2] < .3 then
     begin RUNOUT; PR nlcr; PR string(⊀r0 = ⊁); PR real num(x0[1]);
       PR string(⊀tau0 = ⊁); PR real num(x0[2]); PR nlcr;
       if x0[1] < .04 ∧ x0[2] > .9 then PR string(
⊀r,   tau,  u,     v,        p,        rho,      p/(rho↑ga)⊁)
     end;
     if x0[1] < .16 then
     begin x:= 0; for y:= x0[2] + .2, x0[2], x0[2] - .2 do
       begin if y < 1.1 then print end
     end;
     if x0[2] < .3 then
     begin y:= 0; for x:= x0[1] - .12, x0[1], x0[1] + .12 do print;
       if x0[1] < .86 then goto BEGIN else begin PR nlcr; goto OUT end
     end;
```

```
    for i:= 1,2,3,4 do for j:= 0 step 1 until highest degree do
    begin x:= SUM(k,0,highest degree - j,
      (if k = highest degree - j then u[i,I1(j,k)] else a[i,I1(j,k)]) × (-.2) ↑ k);
      if j = highest degree then u[i,I1(j,0)]:= x else a[i,I1(j,0)]:= x
    end; x0[2]:= x0[2] - .2
  end;

START: NEW PAGE; RUNOUT; PR nlcr; PR string(⟨r = ⟩);
  PR real num(x0[1]); PR string(⟨tau = ⟩); PR real num(x0[2]);
  PR nlcr; m:= (5 × dimension + 1) : 2 - 1;
  PR space(m); PR string(⟨n =⟩); PR space(5 × dimension + 1 - m);
  for i:= 1 step 1 until order do
  begin PR string(⟨u[⟩); PR int num(i); PR string(⟨,m(⟩);
    PR int num(i); PR string(⟨,n)]      ⟩)
  end;
  for i:= 0 step 1 until highest degree do
  begin PR nlcr; PR string(⟨(⟩); PR int num(i); PR string(⟨,- 1 )⟩);
    for m:= 1,2,3,4 do
    begin n sub[1]:= i; n sub[2]:= 0; PR real num(
      if i = highest degree then u[m,I(n sub)] else a[m,I(n sub)])
    end
  end;
  n sub[1]:= n sub[2]:= n:= 0;
  special storage:= matrix is inverted:= false;
  goto AGAIN2;
  comment
```

5.9.4. The results of the calculations

The results concerning the values of

$$u,\ v,\ p,\ \rho \text{ and } p/\rho^{\gamma}$$

are reproduced in this section.

The point, in which a Taylor series has been calculated, is given by *r0* and *tau0*, denoting the values of r_0 and τ_0, respectively. The points in which the values of the calculated series for u, v, p and ρ are computed and printed, are given by *r* and *tau*, the first two columns in the table; while the values of u, v, p, ρ and p/ρ^{γ} themselves are given in the next five columns.

```
r0 = +.300000000000⏨- 1 tau0 = +.100000000000⏨+ 1
r,   tau, u,                v,                p,                rho,              p/(rho↑ga)
+.00 +1.0 +.21875000⏨- 0 -.15900720⏨-13 +.82589286⏨- 0 +.45714286⏨+ 1 +.98367356⏨- 1
+.00  +.8 +.16843859⏨- 0 -.44820643⏨- 8 +.87128246⏨- 0 +.47495072⏨+ 1 +.98367358⏨- 1
r0 = +.300000000000⏨- 1 tau0 = +.800000000000⏨- 0
+.00 +1.0 +.21876266⏨- 0 -.19276610⏨- 5 +.82588068⏨- 0 +.45713805⏨+ 1 +.98367352⏨- 1
+.00  +.8 +.16843859⏨- 0 -.44820358⏨- 8 +.87128246⏨- 0 +.47495072⏨+ 1 +.98367358⏨- 1
+.00  +.6 +.12314710⏨- 0 +.15852056⏨- 5 +.90304915⏨- 0 +.48725624⏨+ 1 +.98367359⏨- 1
r0 = +.300000000000⏨- 1 tau0 = +.600000000000⏨- 0
+.00  +.8 +.16858043⏨- 0 -.23277007⏨- 3 +.87120583⏨- 0 +.47492118⏨+ 1 +.98367270⏨- 1
+.00  +.6 +.12314710⏨- 0 +.15852056⏨- 5 +.90304915⏨- 0 +.48725624⏨+ 1 +.98367359⏨- 1
+.00  +.4 +.80884192⏨- 1 +.18363339⏨- 3 +.92420806⏨- 0 +.49538388⏨+ 1 +.98367386⏨- 1
r0 = +.300000000000⏨- 1 tau0 = +.400000000000⏨- 0
+.00  +.6 +.98166912⏨- 1 +.20099914⏨- 1 +.91533062⏨- 0 +.49197590⏨+ 1 +.98368631⏨- 1
+.00  +.4 +.80884192⏨- 1 +.18363339⏨- 3 +.92420806⏨- 0 +.49538388⏨+ 1 +.98367386⏨- 1
+.00  +.2 -.79058354⏨- 1 -.81171113⏨- 1 +.96955092⏨- 0 +.51269828⏨+ 1 +.98347658⏨- 1
r0 = +.300000000000⏨- 1 tau0 = +.200000000000⏨- 0
+.00  +.4 +.12253269⏨+ 4 -.17857510⏨+ 4 -.37377233⏨+ 3 -.14260710⏨+ 4 -.60396013⏨+619
+.00  +.2 -.79058354⏨- 1 -.81171113⏨- 1 +.96955092⏨- 0 +.51269828⏨+ 1 +.98347658⏨- 1
+.00  -.0 +.10300616⏨+ 2 +.81676880⏨+ 1 -.34431356⏨+ 1 -.10964676⏨+ 2 -.55635916⏨+617
-.09  +.0 -.26089683⏨+ 7 -.16793473⏨+ 7 +.36668186⏨+ 6 +.14222786⏨+ 7 +.89147851⏨- 3
+.03  +.0 +.78797326⏨- 0 -.30448974⏨- 1 +.73679259⏨- 0 +.42475287⏨+ 1 +.97264509⏨- 1
+.15  +.0 +.10656364⏨+ 8 -.16712334⏨+ 8 -.26987080⏨+ 7 -.10306779⏨+ 8 -.43607081⏨+623
r0 = +.150000000000⏨- 0 tau0 = +.100000000000⏨+ 1
+.00 +1.0 +.21875000⏨- 0 -.59308191⏨-11 +.82589286⏨- 0 +.45714286⏨+ 1 +.98367356⏨- 1
+.00  +.8 +.16843853⏨- 0 -.61613292⏨- 8 +.87128254⏨- 0 +.47495075⏨+ 1 +.98367357⏨- 1
r0 = +.150000000000⏨- 0 tau0 = +.800000000000⏨- 0
+.00 +1.0 +.21875022⏨- 0 -.15891891⏨- 6 +.82589263⏨- 0 +.45714276⏨+ 1 +.98367357⏨- 1
+.00  +.8 +.16843853⏨- 0 -.61614447⏨- 8 +.87128254⏨- 0 +.47495075⏨+ 1 +.98367357⏨- 1
+.00  +.6 +.12313805⏨- 0 +.11425161⏨- 6 +.90305499⏨- 0 +.48725850⏨+ 1 +.98367357⏨- 1
r0 = +.150000000000⏨- 0 tau0 = +.600000000000⏨- 0
+.00  +.8 +.16843858⏨- 0 -.35382905⏨- 7 +.87128250⏨- 0 +.47495074⏨+ 1 +.98367357⏨- 1
+.00  +.6 +.12313805⏨- 0 +.11425148⏨- 6 +.90305499⏨- 0 +.48725850⏨+ 1 +.98367357⏨- 1
+.00  +.4 +.80740983⏨- 1 +.34110732⏨- 6 +.92428991⏨- 0 +.49541532⏨+ 1 +.98367357⏨- 1
```

```
r0 = +.150000000000₁₀- 0 tau0 = +.400000000000₁₀- 0
+.00  +.6 +.12313813₁₀- 0 +.14825358₁₀- 7 +.90305496₁₀- 0 +.48725849₁₀+ 1 +.98367356₁₀- 1
+.00  +.4 +.80740983₁₀- 1 +.34110720₁₀- 6 +.92428991₁₀- 0 +.49541532₁₀+ 1 +.98367357₁₀- 1
+.00  +.2 +.39994029₁₀- 1 +.31455005₁₀- 6 +.93653343₁₀- 0 +.50009397₁₀+ 1 +.98367361₁₀- 1
r0 = +.150000000000₁₀- 0 tau0 = +.200000000000₁₀- 0
+.00  +.4 +.80740943₁₀- 1 +.11084884₁₀- 6 +.92429001₁₀- 0 +.49541537₁₀+ 1 +.98367354₁₀- 1
+.00  +.2 +.39994029₁₀- 1 +.31454992₁₀- 6 +.93653343₁₀- 0 +.50009397₁₀+ 1 +.98367361₁₀- 1
+.00  -.0 -.15513982₁₀- 5 -.72662958₁₀- 5 +.94053892₁₀- 0 +.50162049₁₀+ 1 +.98367447₁₀- 1
+.03  +.0 +.65490327₁₀- 3 +.20179074₁₀- 1 +.93951705₁₀- 0 +.50123136₁₀+ 1 +.98367386₁₀- 1
+.15  +.0 +.16207865₁₀- 1 +.99298518₁₀- 1 +.91539386₁₀- 0 +.49200473₁₀+ 1 +.98367357₁₀- 1
+.27  +.0 +.51289288₁₀- 1 +.17223454₁₀- 0 +.86200002₁₀- 0 +.47133090₁₀+ 1 +.98367357₁₀- 1
r0 = +.270000000000₁₀- 0 tau0 = +.200000000000₁₀- 0
+.15  +.0 +.16207510₁₀- 1 +.99298307₁₀- 1 +.91539396₁₀- 0 +.49200477₁₀+ 1 +.98367358₁₀- 1
+.27  +.0 +.51289436₁₀- 1 +.17223493₁₀- 0 +.86199969₁₀- 0 +.47133077₁₀+ 1 +.98367357₁₀- 1
+.39  +.0 +.10330450₁₀- 0 +.23492808₁₀- 0 +.78545151₁₀- 0 +.44103943₁₀+ 1 +.98367356₁₀- 1
r0 = +.390000000000₁₀- 0 tau0 = +.200000000000₁₀- 0
+.27  +.0 +.51288329₁₀- 1 +.17223271₁₀- 0 +.86199978₁₀- 0 +.47133049₁₀+ 1 +.98367448₁₀- 1
+.39  +.0 +.10330416₁₀- 0 +.23492749₁₀- 0 +.78545132₁₀- 0 +.44103920₁₀+ 1 +.98367404₁₀- 1
+.51  +.0 +.16895296₁₀- 0 +.28453660₁₀- 0 +.69334759₁₀- 0 +.40344622₁₀+ 1 +.98367377₁₀- 1
r0 = +.510000000000₁₀- 0 tau0 = +.200000000000₁₀- 0
+.39  +.0 +.10330100₁₀- 0 +.23492174₁₀- 0 +.78545139₁₀- 0 +.44103808₁₀+ 1 +.98367762₁₀- 1
+.51  +.0 +.16895128₁₀- 0 +.28453415₁₀- 0 +.69334732₁₀- 0 +.40344541₁₀+ 1 +.98367619₁₀- 1
+.63  +.0 +.24509282₁₀- 0 +.31943096₁₀- 0 +.59291156₁₀- 0 +.36077929₁₀+ 1 +.98367510₁₀- 1
r0 = +.630000000000₁₀- 0 tau0 = +.200000000000₁₀- 0
+.51  +.0 +.16896925₁₀- 0 +.28456132₁₀- 0 +.69334757₁₀- 0 +.40345325₁₀+ 1 +.98364978₁₀- 1
+.63  +.0 +.24510388₁₀- 0 +.31944451₁₀- 0 +.59291052₁₀- 0 +.36078311₁₀+ 1 +.98365880₁₀- 1
+.75  +.0 +.32991918₁₀- 0 +.33870833₁₀- 0 +.48925658₁₀- 0 +.31451030₁₀+ 1 +.98366646₁₀- 1
r0 = +.750000000000₁₀- 0 tau0 = +.200000000000₁₀- 0
+.63  +.0 +.24508672₁₀- 0 +.31945822₁₀- 0 +.59291154₁₀- 0 +.36077656₁₀+ 1 +.98368550₁₀- 1
+.75  +.0 +.33000280₁₀- 0 +.33878531₁₀- 0 +.48924547₁₀- 0 +.31453442₁₀+ 1 +.98353848₁₀- 1
+.87  +.0 +.42578876₁₀- 0 +.34059603₁₀- 0 +.38285803₁₀- 0 +.26399035₁₀+ 1 +.98359086₁₀- 1
r0 = +.870000000000₁₀- 0 tau0 = +.200000000000₁₀- 0
+.75  +.0 +.31374634₁₀- 0 +.32579148₁₀- 0 +.48928245₁₀- 0 +.30766508₁₀+ 1 +.10144955₁₀- 0
+.87  +.0 +.42449828₁₀- 0 +.33939421₁₀- 0 +.38265076₁₀- 0 +.26330490₁₀+ 1 +.98664304₁₀- 1
+.99  +.0 +.55693038₁₀- 0 +.31115755₁₀- 0 +.25560168₁₀- 0 +.19771404₁₀+ 1 +.98426302₁₀- 1
```

Conclusions

1. The results obtained with the Taylor series expanded in $r_0 = .03$ and $\tau_0 = 1(-.2).2$ become more and more worthless the smaller τ_0. It can be seen from the not reproduced table of the Taylor coefficients, that these coefficients show a very unstable character. Note that the very large values $\simeq 10^{600}$ are due to the fact that the calculated value of ρ is negative.

2. The results obtained with the Taylor series expanded in $r_0 = .15$ and $\tau_0 = 1(-.2).2$ are extremely good. The entropy p/ρ^γ remains constant within six decimals; while $|v| < 10^{-5}$, that is an absolute error of the order 10^{-5} since v should be zero; moreover, u should be zero at $r = \tau = 0$, which results in an absolute error of the order 10^{-6}.

3. The results obtained with the Taylor series expanded in $r_0 = .15(.12).87$, $\tau_0 = .2$ seem to be very reliable; for $r_0 = .15$ about 5 significant decimals and for $r_0 = .87$ about 2 significant decimals seem to be good. This may be seen from:
   a) the "constancy" of the entropy;
   b) comparing the values of the functions calculated in the same point but with different Taylor series.
   It is emphasized that the Taylor series expanded in $(r_0, \tau_0 = .2)$ and in $(r_0 \pm .12, \tau_0 = .2)$ are calculated completely independent of each other (except that they have been calculated using the same differential equations and the same initial values).

5.10. Numerical, approximate, data; higher-order partial-differential equations; common subformulae

Suppose that the differential equations depend upon an approximately known function $F(x_1, \ldots, x_d)$; how should this function be handled. An easy way is to replace $F(x_1, \ldots, x_d)$ by a high power of x_1, say $x_1 \uparrow 1000$. This power manifests itself in the computational program through the occurrence of *BINOMIAL COEFF(1,1000)*. The procedure *BINOMIAL COEFF* can then be made such that for *j* = 1000, it delivers the n-th Taylor coefficient of $F(x_1, \ldots, x_d)$ using the approximate data.

The same provisions can be supplied if the approximately known function F depends also on $U_1, \ldots, U_{order}$, if, at least, *type of U*[i] $\neq 0$ for $i = 1, \ldots,$ *order*.

If, however, F depends also on some highest derivatives of the U_i's, i.e. $F = F(\ldots, \partial U_i/\partial x_j, \ldots)$ and *type of U*[i] = j, then the above device cannot be used anymore.

The only way to treat this problem seems to be to write F explicitly as a truncated power series in these highest derivatives of the U_i's; the approximate coefficients, possibly in a symbolic fashion, should then be used to construct the formula program for the algebraic program.

A "higher-order" system of partial-differential equations, "higher-order" meaning: involving higher-than-the-first-order partial derivatives, can be brought into a system of first-order partial-differential equations. Without sufficient care, however, this may lead to difficulties which will be illustrated by the following example:

Let $\phi(x_1, x_2, x_3, t)$ satisfy

$$\Delta\phi = \frac{\partial^2\phi}{\partial t^2} + F(\phi, x_1, x_2, x_3, t), \tag{5.10.1}$$

and $\phi(x_1, x_2, x_3, 0) = \phi_0(x_1, x_2, x_3)$,

$$\frac{\partial\phi}{\partial t}(x_1, x_2, x_3, 0) = \psi_0(x_1, x_2, x_3);$$

calculate the Taylor coefficients of ϕ in the point (0,0,0,0). The system does not admit second-order derivatives, we, therefore, introduce in a naive manner the functions U_i, $i = 1, \ldots, 5$, with $U_5 = \phi$, and the equations:

$$G_i = U_i - \frac{\partial U_5}{\partial x_i} = 0,\ i = 1, \ldots, 4,$$

$$G_5 = \sum_{i=1}^{3} \frac{\partial U_i}{\partial x_i} - \frac{\partial U_4}{\partial x_4} - F(U_5, x_1, x_2, x_3, x_4) = 0, \qquad (5.10.2)$$

with $x_4 = t$.

These equations are of the desired "first-order" form; they do not, however, satisfy condition 5 of section 5.1. This may be seen as follows:

the types of the U_i's are given by $t_i = i$, for $i = 1, 2, 3, 4$ and $t_5 = 4$;

hence (see equation 5.1.5a)

$$\det \left\{\frac{\partial G_1}{\partial \overline{Q}_k}\right\} = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 1 & -1 & 0 \end{vmatrix} = 0.$$

In fact, the equations G_1, G_2 and G_3 of (5.10.2) have a different character than G_4 and G_5. A possible calculation process could be:

Use G_4 and G_5 to calculate the new Taylor coefficient of U_4 and U_5; use G_1, G_2 and G_3 to "update" the coefficients of U_1, U_2 and U_3.

Of course, the algebraic program of section 5.7 does not produce this process.

The difficulty is not very serious; instead of equations (5.10.2), we use

$$\overline{G}_i = \frac{\partial U_i}{\partial x_4} + \frac{\partial U_4}{\partial x_i} = 0,\ i = 1, 2, 3,$$

$$\overline{G}_4 = \frac{\partial U_4}{\partial x_4} + \sum_{i=1}^{3} \frac{\partial U_i}{\partial x_i} - F(U_5, x_1, x_2, x_3, x_4) = 0, \qquad (5.10.3)$$

$$\overline{G}_5 = \frac{\partial U_5}{\partial x_4} + U_4 = 0,$$

which are derived from (5.10.2) by differentiating G_1, G_2 and G_3 with respect to t (= x_4) and by changing the sign of U_4.

For

$$F(\phi, x_1, x_2, x_3, t) =$$

$$\alpha \sin(\phi) + 4e^s f(t) - \alpha \sin(e^s f(t)),$$

with $s = x_1 + x_2 + x_3$ and $f(t) = \sin(t) + \cos(t)$,

and $\psi_0 = \phi_0 = e^s$, we have $\phi = e^s f(t)$.

The formula program is given below:

```
Example 2: Formula program RPR 250768/01
(8192, 100, 10, 0, 0, ₁₀-10, ₁₀-10, 9, 0)
t:= X(4); phi:= dUdX(5,0); s:= X(1) + X(2) + X(3);
f1:= dUdX(1,4) + dUdX(4,1);
f2:= dUdX(2,4) + dUdX(4,2);
f3:= dUdX(3,4) + dUdX(4,3);
f4:= dUdX(4,4) + dUdX(1,1) + dUdX(2,2) + dUdX(3,3)
   - alpha × sin(phi) + alpha × sin(exp(s) × (sin(t) + cos(t)))
   - 4 × exp(s) × (sin(t) + cos(t));
f5:= dUdX(5,4) + dUdX(4,0);
DTC(f1,f2,f3,f4,f5);
END;
```

The 4-dimensionality and the frequent occurrence of the sine and cosine function are reasons to treat this problem,as another example, comprehensively.
The results of the algebraic program are given now:

```
The differential equations are:
diff eq[1] = (dUdX(1,4)+dUdX(4,1))
diff eq[2] = (dUdX(2,4)+dUdX(4,2))
diff eq[3] = (dUdX(3,4)+dUdX(4,3))
diff eq[4] = (((((dUdX(4,4)+dUdX(1,1))+dUdX(2,2))+dUdX(3,3))+((-1)×(alpha×sin(
   dUdX(5,0)))))+(alpha×sin((exp(((X(1)+X(2))+X(3)))×(sin(X(4))+cos(X(4)))))))+((-1)×
   (((4)×exp(((X(1)+X(2))+X(3))))×(sin(X(4))+cos(X(4))))))
diff eq[5] = (dUdX(5,4)+dUdX(4,0))
The transformed differential equations are:
diff eq[1] = (dUdX(1,4)+dUdX(4,1))
diff eq[2] = (dUdX(2,4)+dUdX(4,2))
diff eq[3] = (dUdX(3,4)+dUdX(4,3))
diff eq[4] = (((((dUdX(4,4)+dUdX(1,1))+dUdX(2,2))+dUdX(3,3))+(((-1)×alpha)×
   sin(dUdX(5,0))))+(((-1)×(4))×(exp(((X(1)+X(2))+X(3)))×(sin(X(4))+cos(X(4))))))+
   (alpha×sin((exp(((X(1)+X(2))+X(3)))×(sin(X(4))+cos(X(4)))))))
diff eq[5] = (dUdX(5,4)+dUdX(4,0));
a[6,n]:= SHIFT(1,4);
a[7,n]:= SHIFT(4,1);
a[7,n]:= a[6,n] + a[7,n];
a[7,n]:= SHIFT(2,4);
a[6,n]:= SHIFT(4,2);
a[6,n]:= a[7,n] + a[6,n];
a[6,n]:= SHIFT(3,4);
a[7,n]:= SHIFT(4,3);
a[7,n]:= a[6,n] + a[7,n];
a[7,n]:= SHIFT(4,4);
a[6,n]:= SHIFT(1,1);
a[6,n]:= a[7,n] + a[6,n];
a[7,n]:= SHIFT(2,2);
```

```
a[7,n]:= a[6,n] + a[7,n];
a[6,n]:= SHIFT(3,3);
a[6,n]:= a[7,n] + a[6,n];
a[7,n]:= sin(a[5,0]);
a[8,n]:= cos(a[5,0]);
a[9,n]:= ((-1)) × a[7,n];
a[10,n]:= (((-1)×alpha)) × a[7,n];
a[10,n]:= a[6,n] + a[10,n];
a[6,n]:= BINOMIAL COEFF(1, 1);
a[11,n]:= BINOMIAL COEFF(2, 1);
a[11,n]:= a[6,n] + a[11,n];
a[6,n]:= BINOMIAL COEFF(3, 1);
a[6,n]:= a[11,n] + a[6,n];
a[11,n]:= exp(a[6,0]);
a[12,n]:= BINOMIAL COEFF(4, 1);
a[13,n]:= sin(a[12,0]);
a[14,n]:= cos(a[12,0]);
a[15,n]:= ((-1)) × a[13,n];
a[16,n]:= BINOMIAL COEFF(4, 1);
a[17,n]:= cos(a[16,0]);
a[18,n]:= sin(a[16,0]);
a[19,n]:= ((-1)) × a[18,n];
a[20,n]:= a[13,n] + a[17,n];
a[21,n]:= a[11,0] × a[20,0];
a[21,n]:= (((-1)×(4))) × a[21,n];
a[21,n]:= a[10,n] + a[21,n];
a[10,n]:= BINOMIAL COEFF(1, 1);
a[22,n]:= BINOMIAL COEFF(2, 1);
a[22,n]:= a[10,n] + a[22,n];
a[10,n]:= BINOMIAL COEFF(3, 1);
a[10,n]:= a[22,n] + a[10,n];
a[22,n]:= exp(a[10,0]);
a[23,n]:= BINOMIAL COEFF(4, 1);
a[24,n]:= sin(a[23,0]);
a[25,n]:= cos(a[23,0]);
```

```
a[26,n]:= ((-1)) × a[24,n];
a[27,n]:= BINOMIAL COEFF(4, 1);
a[28,n]:= cos(a[27,0]);
a[29,n]:= sin(a[27,0]);
a[30,n]:= ((-1)) × a[29,n];
a[31,n]:= a[24,n] + a[28,n];
a[32,n]:= a[22,0] × a[31,0];
a[33,n]:= sin(a[32,0]);
a[34,n]:= cos(a[32,0]);
a[35,n]:= ((-1)) × a[33,n];
a[36,n]:= (alpha) × a[33,n];
a[36,n]:= a[21,n] + a[36,n];
a[36,n]:= SHIFT(5,4);
a[36,n]:= a[36,n] + a[4,n];
AGAIN: AUGMENT INDEX;
a[7,n]:= SHIFT(4,1);
coeff[1]:= a[7,n];
RHS[1]:= - (coeff[1]);
GOTO(LABEL2);
COEFF[1,1]:= 1;
COEFF[1,2]:= 0;
COEFF[1,3]:= 0;
COEFF[1,4]:= 0;
COEFF[1,5]:= 0;
LABEL2:
a[6,n]:= SHIFT(4,2);
coeff[2]:= a[6,n];
RHS[2]:= - (coeff[2]);
GOTO(LABEL3);
COEFF[2,1]:= 0;
COEFF[2,2]:= 1;
COEFF[2,3]:= 0;
COEFF[2,4]:= 0;
COEFF[2,5]:= 0;
```

```
LABEL3:
a[7,n]:= SHIFT(4,3);
coeff[3]:= a[7,n];
RHS[3]:= - (coeff[3]);
GOTO(LABEL4);
COEFF[3,1]:= 0;
COEFF[3,2]:= 0;
COEFF[3,3]:= 1;
COEFF[3,4]:= 0;
COEFF[3,5]:= 0;
LABEL4:
a[6,n]:= SHIFT(1,1);
coeff[4]:= a[6,n];
a[7,n]:= SHIFT(2,2);
coeff[5]:= a[7,n];
a[6,n]:= SHIFT(3,3);
coeff[6]:= a[6,n];
a[7,n]:= CONV PRODUCT(8,5,0,-1);
a[8,n]:= CONV PRODUCT(9,5,0,-1);
a[9,n]:= ((-1)) × a[7,n];
a[10,n]:= (((-1)×alpha)) × a[7,n];
coeff[7]:= a[10,n];
a[6,n]:= BINOMIAL COEFF(1, 1);
a[11,n]:= BINOMIAL COEFF(2, 1);
a[11,n]:= a[6,n] + a[11,n];
a[6,n]:= BINOMIAL COEFF(3, 1);
a[6,n]:= a[11,n] + a[6,n];
a[11,n]:= CONV PRODUCT(11,6,0,-1);
a[12,n]:= BINOMIAL COEFF(4, 1);
a[13,n]:= CONV PRODUCT(14,12,0,-1);
a[14,n]:= CONV PRODUCT(15,12,0,-1);
a[15,n]:= ((-1)) × a[13,n];
a[16,n]:= BINOMIAL COEFF(4, 1);
a[17,n]:= CONV PRODUCT(19,16,0,-1);
a[18,n]:= CONV PRODUCT(17,16,0,-1);
```

```
a[19,n]:= ((-1)) × a[18,n];
a[20,n]:= a[13,n] + a[17,n];
a[21,n]:= CONV PRODUCT(11,20,0,0);
a[21,n]:= (((-1)×(4))) × a[21,n];
coeff[8]:= a[21,n];
a[10,n]:= BINOMIAL COEFF(1, 1);
a[22,n]:= BINOMIAL COEFF(2, 1);
a[22,n]:= a[10,n] + a[22,n];
a[10,n]:= BINOMIAL COEFF(3, 1);
a[10,n]:= a[22,n] + a[10,n];
a[22,n]:= CONV PRODUCT(22,10,0,-1);
a[23,n]:= BINOMIAL COEFF(4, 1);
a[24,n]:= CONV PRODUCT(25,23,0,-1);
a[25,n]:= CONV PRODUCT(26,23,0,-1);
a[26,n]:= ((-1)) × a[24,n];
a[27,n]:= BINOMIAL COEFF(4, 1);
a[28,n]:= CONV PRODUCT(30,27,0,-1);
a[29,n]:= CONV PRODUCT(28,27,0,-1);
a[30,n]:= ((-1)) × a[29,n];
a[31,n]:= a[24,n] + a[28,n];
a[32,n]:= CONV PRODUCT(22,31,0,0);
a[33,n]:= CONV PRODUCT(34,32,0,-1);
a[34,n]:= CONV PRODUCT(35,32,0,-1);
a[35,n]:= ((-1)) × a[33,n];
a[36,n]:= (alpha) × a[33,n];
coeff[9]:= a[36,n];
RHS[4]:= - ((coeff[9]+(coeff[8]+(coeff[7]+(coeff[6]+(coeff[5]+coeff[4]))))));
GOTO(LABEL5);
COEFF[4,1]:= 0;
COEFF[4,2]:= 0;
COEFF[4,3]:= 0;
COEFF[4,4]:= 1;
COEFF[4,5]:= 0;
LABEL5:
coeff[10]:= a[4,n];
```

```
RHS[5]:= - (coeff[10]);
GOTO(LABEL6);
COEFF[5,1]:= 0;
COEFF[5,2]:= 0;
COEFF[5,3]:= 0;
COEFF[5,4]:= 0;
COEFF[5,5]:= 1;
LABEL6:
JUMP TO CALC OF UNKNOWNS;
FILL IN OPEN PLACES:
goto AGAIN;
OUT: end end end end end

36  10  5  4  4  4  4  4
'ready
line number = 10 execution time = 14 sec
```

The input tape for the computational program consists of:

```
'dimension = 4, highest degree = 3,
x = 0, y = 0, z = 0, t = 0
(0,0,0)  1    1    1    -1    1
(1,0,0)  1    1    1    -1    1
(0,1,0)  1    1    1    -1    1
(0,0,1)  1    1    1    -1    1
(2,0,0)  .5   .5   .5   -.5   .5
(1,1,0)  1    1    1    -1    1
(1,0,1)  1    1    1    -1    1
(0,2,0)  .5   .5   .5   -.5   .5
(0,1,1)  1    1    1    -1    1
```

(0,0,2)	.5	.5	.5	−.5	.5
(3,0,0)	.1667	.1667	.1667	−.1667	.1667
(2,1,0)	.5	.5	.5	−.5	.5
(2,0,1)	.5	.5	.5	−.5	.5
(1,2,0)	.5	.5	.5	−.5	.5
(1,1,1)	1	1	1	−1	1
(1,0,2)	.5	.5	.5	−.5	.5
(0,3,0)	.1667	.1667	.1667	−.1667	.1667
(0,2,1)	.5	.5	.5	−.5	.5
(0,1,2)	.5	.5	.5	−.5	.5
(0,0,3)	.1667	.1667	.1667	−.1667	.1667
(0,0,0)	1	1	1	1	1

A slightly modified computational program produced the following results: first the index vector n, then the Taylor coefficients $u_{i,m(n)}$, with $i = 1, 2, 3, 4, 5$ and $m(n) = \mu(n,4)$; i.e. $m_j = n_j$, $j = 1, 2, 3$, $m_4 = n_4+1$. Note that $U_5 = \phi$.

(1 , 0 , 0 , 0)	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$
(0 , 1 , 0 , 0)	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$
(0 , 0 , 1 , 0)	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$
(0 , 0 , 0 , 1)	$-.5000_{10}-0$	$-.5000_{10}-0$	$-.5000_{10}-0$	$+.5000_{10}-0$	$-.5000_{10}-0$
(2 , 0 , 0 , 0)	$+.5001_{10}-0$	$+.5000_{10}-0$	$+.5000_{10}-0$	$+.4999_{10}-0$	$+.5000_{10}-0$
(1 , 1 , 0 , 0)	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$
(1 , 0 , 1 , 0)	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$
(1 , 0 , 0 , 1)	$-.4999_{10}-0$	$-.5000_{10}-0$	$-.5000_{10}-0$	$+.4999_{10}-0$	$-.5000_{10}-0$
(0 , 2 , 0 , 0)	$+.5000_{10}-0$	$+.5001_{10}-0$	$+.5000_{10}-0$	$+.4999_{10}-0$	$+.5000_{10}-0$
(0 , 1 , 1 , 0)	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$	$+.1000_{10}+1$
(0 , 1 , 0 , 1)	$-.5000_{10}-0$	$-.4999_{10}-0$	$-.5000_{10}-0$	$+.4999_{10}-0$	$-.5000_{10}-0$
(0 , 0 , 2 , 0)	$+.5000_{10}-0$	$+.5000_{10}-0$	$+.5001_{10}-0$	$+.4999_{10}-0$	$+.5000_{10}-0$
(0 , 0 , 1 , 1)	$-.5000_{10}-0$	$-.5000_{10}-0$	$-.4999_{10}-0$	$+.4999_{10}-0$	$-.5000_{10}-0$
(0 , 0 , 0 , 2)	$-.1666_{10}-0$	$-.1666_{10}-0$	$-.1666_{10}-0$	$-.1668_{10}-0$	$-.1667_{10}-0$

In the above example we have shown that one has to be careful in constructing a first-order system of equations from a second- (or highest-) order system.

Another point to which we want to draw the attention has been exemplified already in section 5.9.
There, we have equations in which a certain formula ρ occurs several times. It is natural, therefore, to introduce a new dependent variable r, which should be inserted for the formula ρ and which might be introduced in a naive manner by the equation

$$\rho - r = 0.$$

This would work quite well if the differential equations did not involve ρ as well as, for some i, the formula $\partial\rho/\partial x_i$, in which case it is necessary to introduce the variable r by means of the equation:

$$\partial\rho/\partial x_i - \partial r/\partial x_i = 0,$$

where $\partial\rho/\partial x_i$ is the explicitly written down derivative of ρ with respect to x_i.
The reason is obvious: without using the latter equation the determinant (5.1.5a) would be zero.
The necessary differentiation, which has to be carried out anyhow, may be done by means of the formula-manipulation system of chapters 2 and 3. Note, that the differentiation cannot be done directly by a call of DER in the formula program for the algebraic program, since the use of DER is forbidden.

Concluding the investigations of this chapter, we observe that the question: how much Taylor coefficients should be calculated and in which points, is completely open and should be subject of future investigations.

Appendix

The headings of the Mathematical Centre standard-procedures as used in the ALGOL 60 programs are given below along with a short description. More information is given in [8].

Input procedures:

real procedure read; reads a number from input tape and becomes equal to this number.
Text between two apostrophes is skipped.
integer procedure RESYM; reads a symbol from input tape and becomes equal to the internal representation of this symbol;

Output procedures for printing:

procedure PRSYM(n); value n; integer n;
If the actual value of *n* corresponds to the internal representation of a symbol which may occur on input tape, then this symbol is printed after a call of this procedure.

procedure NLCR; a call of this procedure has the effect that a new-line-carriage-return command is given to the printer.

procedure ABSFIXT(n,m,x); value n, m, x; integer n, m; real x;
a call of this procedure has as effect the printing of the absolute value of the actual parameter *x* with *n* decimals before and *m* decimals after the decimal point.

procedure FIXT(n,m,x); value n, m, x; integer n, m; real x;
a call of this procedure has as effect the printing of the value of the actual parameter *x* (including the sign symbol) with *n* decimals before and *m* decimals after the decimal point.

procedure FLOT(n,m,x); value n, m, x; integer n, m; real x;
a call of this procedure has as effect the printing of the value of the actual parameter *x* in floating-point notation (including the sign symbol) with *n* decimals for the mantissa and *m* decimals for the exponent of 10.

procedure *PRINTTEXT(s);* _string_ *s;*
a call of this procedure has as effect the printing of the string given by the actual parameter *s*, without the string quotes ⫞ and ⫟.

procedure *SPACE(n);* _value_ *n;* _integer_ *n;*
a call of this procedure has as effect the printing of *n* space symbols.

<u>Output procedures for punching</u>:

The headings of these procedures are almost the same as the headings of the procedures for printing; the procedure identifiers are changed into *PUSYM*, *PUNLCR*, *ABSFIXP*, *FIXP*, *FLOP*, *PUTEXT* and *PUSPACE*.

procedure *RUNOUT;* a call of this procedure has as effect the punching of a piece of blank tape.

procedure *STOPCODE;* a call of this procedure has as effect the punching of a stopcode symbol and a call of *RUNOUT*.

<u>Other procedures</u>

procedure *EXIT;* a call of this procedure has as effect that the execution of the program is discontinued.

integer _procedure_ *EVEN(n);* _value_ *n;* _integer_ *n;*
may be described as: $EVEN := (-1) \uparrow n$.

real _procedure_ *SUM(i,a,b,xi);* _value_ *b;* _integer_ *i, a, b;* _real_ *xi;*
may be described as:

$$SUM := \sum_{i=a}^{b} xi$$

The actual value of *xi* will in general depend on the current value of *i*.

procedure *TO DRUM(A,p);* _value_ *p;* _array_ *A;* _integer_ *p;*
A call of this procedure has as effect that the array elements of *A* are stored on the drum; the first array element is stored in a location defined by the actual value of *p*.

procedure FROM DRUM(A, p); *value* p; *array* A; *integer* p;
A call of this procedure has as effect that a set of data occurring on the drum and with begin address defined by *p* are stored into the array elements of *A*.

real *procedure* time;
This procedure delivers the time, measured in seconds with an accuracy of .01 sec, during which an ALGOL 60 program was under execution.

References

[1] M. Abramowitz, I.A. Stegun (editors), Handbook of Mathematical Functions, U.S. Department of Commerce, National Bureau of Standards, Applied Mathematics Series 55, Third printing, March 1965.

[2] S.J. Bijlsma, Algebraic Operations in ALGOL 60. The Saddlepoint method, Report TN 45, Mathematical Centre, Amsterdam, May 1966.

[3] E. Bond, M. Auslander, S. Grisoff, R. Kenney, M. Myszewski, J.E. Sammet, R.G. Tobey, S. Zilles, FORMAC - An experimental formula manipulation compiler, Proceedings of the ACM National Conference, August 1964.

[4] W.S. Brown, The ALTRAN language and the ALPAK system for symbolic algebra on a digital computer, Presented at the Princeton University Conference in Computer Sciences, August 8 - September 2, 1966.

[5] R. Courant, D. Hilbert, Methods of Mathematical Physics, volume II, Interscience 1962.

[6] A. Gibbons, A program for the automatic integration of differential equations, using the method of Taylor series, The Computer Journal, 3 (1960), pp. 108-111.

[7] F.E.J. Kruseman Aretz, ALGOL 60 translation for everybody, Electronische Datenverarbeitung Heft 6/1964, pp. 233-244.

[8] F.E.J. Kruseman Aretz, Het MC-ALGOL 60-systeem voor de X8. Voorlopige programmeursopleiding. Report MR 81, Mathematical Centre, Amsterdam, June 1966.

[9] J. McCarthy, P. Abrahams, D. Edwards, T. Hart, M. Levin, LISP 1.5 Programmer's Manual,
Computation Centre and Research Laboratory of Electronics,
MIT, Cambridge Mass.

[10] R.E. Moore, The automatic analysis and control of error in digital computation based on the use of interval numbers,
published in Error in Digital Computation, vol. 1, edited by Louis B. Rall,
John Wiley & Sons, Inc., New York 1964.

[11] P. Naur (editor), Revised report on the algorithmic language ALGOL 60,
Regnecentralen, Copenhagen, 1962.

[12] A.J. Perlis, R. Itturiaga, T.A. Standish, A Definition of Formula ALGOL,
Carnegie Institute of Technology, Pittsburgh, P.A.,
March 1966.

[13] R.D. Richtmyer, Power series solution, by machine, of a nonlinear problem in two-dimensional fluid flow,
Annals of the New York Academy of Sciences, <u>86</u> (1960),
pp. 828-843.

[14] R.P. van de Riet, Algebraic Operations in ALGOL 60. A second order problem.
Report TW 96, Mathematical Centre, Amsterdam, March 1965.

[15] R.P. van de Riet, Algebraic Operations in ALGOL 60. The Cauchy Problem I,
Report TW 97, Mathematical Centre, Amsterdam, December 1965.

[16] R.P. van de Riet, An application of a method for algebraic manipulation in ALGOL 60,
Report TW 99, Mathematical Centre, Amsterdam, January 1966.

[17] R.P. van de Riet, Formula manipulation in ALGOL 60 (preliminary report),
Report TW 101, Mathematical Centre, Amsterdam, August 1966.

[18] R.P. van de Riet, Complex Arithmetic in ALGOL 60,
Report TW 105, Mathematical Centre, Amsterdam, March 1967.

[19] R.P. van de Riet, Algorithm 186 Complex Arithmetic,
Communications of the ACM, 6, 7, July 1963.

[20] J.E. Sammet, Formula Manipulation by Computer,
Advances in Computers, Volume 8, McGraw Hill,
Preliminary version as Technical Report TR00.1363, IBM Systems Development Division,
Poughkeepsie, N.Y., November 1965.

[21] J.E. Sammet, Survey of the Use of Computers for Doing Non-Numerical Mathematics,
Technical Report TR00.1428,
IBM Systems Development Division,
Poughkeepsie, N.Y., March 1966.

[22] J.E. Sammet, An Annotated Descriptor Based Bibliography on the Use of Computers for Non-Numerical Mathematics,
Computing Reviews 7, 4, July-August 1966.

[23] J.C. Schoenfeld, Complex Arithmetics in ALGOL,
RIJSKWATERSTAAT, Directie Waterhuishouding en Waterbeweging,
Mathematisch-Fysische Afdeling,
Boorlaan 14, Den Haag, Netherlands, 1965.

[24] R.L. Smith, Algorithm 116 Complex Division,
Communications of the ACM, 5 (1962).

[25] C.R. Traas, Blunt body supersonic flow. Investigations about an inverse and a direct method,
report G 63, National Aerospace Laboratory, Amsterdam (1967).

[26] B.L. van der Waerden, Algebra, Vierte Auflage,
Springer-Verlag, Berlin-Göttingen-Heidelberg (1955).

[27] P. Wynn, An Arsenal of ALGOL.procedures for Complex Arithmetic,
BIT, 2 (1962), pp. 232-255.

[28] R.P. van de Riet, ABC ALGOL, a portable language for formula
manipulation systems, part 1: the language.
MC Tracts 46, Mathematical Centre, Amsterdam, 1973.

[29] R.P. van de Riet, ABC ALGOL, a portable language for formula
manipulation systems, part 2, the compiler.
MC Tracts 47, Mathematical Centre, Amsterdam, 1973.