HALDOR TOPSØE

PROGRAMMING OF CHEMICAL CALCULATIONS

VOLUME 3: NUMERICAL METHODS FOR INTEGRATION AND SOLUTION OF DIFFERENTIAL EQUATIONS

Jørgen Kjær

Haldor Topsøe, Vedbæk, Denmark

1965

Copy No. 1

PREFACE

This third volume in the series of books on programming of chemical calculations contains the numerical methods which could not be contained in volume 2. They are the methods for calculation of definite integrals and for solution of differential equations. For ordinary differential equations we discuss the methods of Euler and of Runge-Kutta. We do not have any general procedures for partial differential equations, but we explain two special procedures for the solution of a certain type of partial differential equations which are of interest for the calculation of temperature and conversion profiles in cylindric catalytic reactors.

Vedbæk, July 1965

Jørgen Kjær

CONTENTS

1. INTRODUCTION

In volume 2 we discussed various examples of function analysis. As an example we saw how to find the maximum of a function by expressing it as a polynomial in the independent variables, calculating the derivatives, and putting these equal to zero. Here, we begin with the original function, find the derivatives, and continue the analysis (here a root determination) on the derivatives.

In calculation of definite integrals and solution of differential equations we have the reverse situation: We start with one or more derivatives of an otherwise unknown function and must now calculate backwards to the original function.

When we pass from a derivative to the function itself, we say that we solve a differential equation. The most important classification of differential equations is obtained by considering the number of independent variables. If there is only one independent variable, we have an ordinary differential equation, and if there are several independent variables, we have a set of partial differential equations. Both types are discussed in chapter 3.

An ordinary differential equation of the first order may be written as:

$$dy/dx = f(x, y)$$

The derivative, dy/dx, is a function of the independent variable, x, and of the unknown function, y. In simpler cases y is not contained in the right-hand side:

$$dy/dx = f(x)$$

and the increase in the y-function may then be calculated as a definite integral. This is described in chapter 2.

2. CALCULATION OF DEFINITE INTEGRALS

Calculation of definite integrals is a special case of the solution of differential equations. We have a function of a single independent variable, and we know an expression for the derivative of the function:

$$dy/dx = f(x)$$

The function expression on the right-hand side contains x only, not y. We must solve the differential equation to find the function y:

$$y = F(x)$$

so that it has the correct derivative. In calculation of the definite integral of f(x) from a to b:

$$INT = \int_a^b f(x)dx$$

we must calculate:

$$INT = F(b) - F(a)$$

i.e. the increase in the F-function from a to b.

In the Haldor Topsøe GIER installation we very often must solve complicated differential equations, whereas definite integrals are calculated only now and then. We, therefore, have several different procedures for the solution of differential equations, and only a single procedure for the calculation of definite integrals.

2.1. The Procedure SIMPS2.

The simpler methods for calculation of the definite integral of f(x) from a to b utilize a division of the interval from a to b into N equal parts, e.g. by repeated halving. We denote the function values in the points of division like this:

```
x = a    f(0)
  .      f(1)
  .      f(2)
  .        .
  .        .
  .        .
x = b    f(N)
```

If the length of a sub-interval is called delx, the simplest formula (the trapezoidal formula) may be written:

```
    INT := delx×(0.5×f(0) + f(1) + f(2) + .....
         ........ f(N-1) + 0.5×f(N));
```

We also have Simpsons formula which is more accurate:

```
    INT := 1/3×delx×(f(0) + 4×f(1) + 2×f(2) + 4×f(3) + .....
         ........+ 4×f(N-1) + f(N));
```

We first used a procedure, SIMPS1, made by P. Naur and which contained Simpsons formula. Later, we have adopted the trapezoidal formula in a procedure Romberg, published by Gram (1964). In his version special precautions are made in order to reduce the rounding errors.

The declaration is:

```
real procedure SIMPS2(a, b, x, f, delta, max);
value a, b, delta;
real a, b, x, f, delta;
integer max;
begin
   real step, I1, I2, sum, error, f0;
   integer k, p, j;
   array trapez[1:max+1];
   step := b - a;
   x := a;
   I1 := f;
   x := b;
   trapez[1] := (I1+f)×step/2;
```

```
  I1 := 0;
  for k := 1 step 1 until max do
  begin
     sum := error := 0;
     step := step/2;
     p := 2↑k;
     for j := p-1 step -2 until 1 do
     begin
        x := j/p;
        x := x×a + (1-x)×b;
        f0 := f;
        I2 := sum + f0;
        error := error + (if abs (f0) > abs (sum)
        then sum - (I2-f0) else f0 - (I2-sum));
        sum := I2
     end for j;
     trapez[k+1] := trapez[k]/2 + (I2+error)×step;
     p := 1;
     for j := k step -1 until 1 do
     begin
        p := 4×p;
        trapez[j] := (trapez[j+1]×p - trapez[j])/(p-1)
     end for j;
     I2 := trapez[1];
     if abs (I2-I1) ≤ delta×abs(I2) then go to finis;
     I1 := I2
  end for k;
finis: max := k;
  SIMPS2 := I2
end of procedure;
```

The formal parameters are:

real a, b:	The integration limits.
real x:	The independent variable.
real f:	The expression defining the function to be integrated.
real delta:	The integration accuracy. The procedure divides the interval from a to b into 2, 4, 8, 16 etc. parts and stops when the relative deviation between two consecutive approximations to the integral becomes less than delta.

integer max: This is the maximum number of bisections. When the procedure is finished, max contains the actual number of bisections carried out.

We first give a very simple example of the use of SIMPS2. We wish to integrate the function:

```
func := 1 + 27×x + 39×sin(x);
```

from $x = 5$ to $x = 6$. As a check the program also calculates the analytic solution as FUNC(6) - FUNC(5), where FUNC is the indefinite integral:

```
FUNC := x + 13.5×x↑2 - 39×cos(x);
```

The program is:

```
Program d-207. Test of SIMPS2.
begin
   real NUMERICAL, ANALYTIC, x;
   comment library SIMPS2;
   real procedure func(x);
   value x;
   real x;
   func := 1 + 27×x + 39×sin(x);
   real procedure FUNC(x);
   value x;
   real x;
   FUNC := x + 13.5×x↑2 - 39×cos(x);
   NUMERICAL := SIMPS2(5, 6, x, func(x), 1₁₀-5, 8);
   ANALYTIC := FUNC(6) - FUNC(5);
   outcr;
   output(⟨-nddd.dddddd⟩, NUMERICAL, ANALYTIC)
end program;
```

The output is:

```
 123.116184  123.116183
```

The second program for testing of SIMPS2 integrates the square root of x from a to b. We keep b constant equal to 1 and let a = limit assume the two values 0.5 and 0. We also test various values of delta and max. The analytic solution:

```
        (1 - limit×sqrt(limit))/1.5
```

is also written out. The program is:

```
Program d-208. Test of SIMPS2.
begin
   integer max1, max2;
   real NUM, AN, x;
   comment library SIMPS2;
   procedure TEST(limit, delta);
   value limit, delta;
   real limit, delta;
   begin
      outcr;
      output(⟨-n.d⟩, limit);
      outsp(2);
      output(⟨-n₁₀-d⟩, delta);
      for max1 := 4 step 1 until 10 do
      begin
         if max1>4 then
         begin
            outcr;
            outsp(11)
         end if;
         max2 := max1;
         NUM := SIMPS2(limit, 1, x, sqrt(x), delta, max2);
         AN := (1 - limit×sqrt(limit))/1.5;
         output(⟨-nddd⟩, max1, max2);
         output(⟨-nd.dddddddd⟩, NUM, AN, NUM-AN);
         if max1 - max2 ≥ 2 then go to EXIT
      end for max1;
 EXIT: end TEST;
   outcr;
   outtext(⟨<limit delta max1 max2  numerical   analytic    difference⟩);
   outcr;
   TEST(0.5, 1⏨-5);
   TEST(0.5, 1⏨-7);
   TEST(0, 1⏨-5);
   TEST(0, 1⏨-7)
end program;
```

The output is:

```
limit delta max1 max2 numerical   analytic    difference

 0.5   1⏨-5    4    3  0.43096440  0.43096441 -0.00000001
               5    3  0.43096440  0.43096441 -0.00000001
 0.5   1⏨-7    4    4  0.43096441  0.43096441  0.00000000
               5    4  0.43096441  0.43096441  0.00000000
               6    4  0.43096441  0.43096441  0.00000000
 0.0   1⏨-5    4    5  0.66559287  0.66666667 -0.00107380
               5    6  0.66628769  0.66666667 -0.00037897
               6    7  0.66653274  0.66666667 -0.00013393
               7    8  0.66661932  0.66666667 -0.00004734
               8    9  0.66664992  0.66666667 -0.00001674
               9   10  0.66666074  0.66666667 -0.00000592
              10   10  0.66666457  0.66666667 -0.00000209
 0.0   1⏨-7    4    5  0.66559287  0.66666667 -0.00107380
               5    6  0.66628769  0.66666667 -0.00037897
               6    7  0.66653274  0.66666667 -0.00013393
               7    8  0.66661932  0.66666667 -0.00004734
               8    9  0.66664992  0.66666667 -0.00001674
               9   10  0.66666074  0.66666667 -0.00000592
              10   11  0.66666457  0.66666667 -0.00000209
```

The program writes the value of max before and after the procedure call. We see, that for limit = 0.5 we need only a few bisections, but for limit = 0 more are required. The reason for this is that the function sqrt(x) has a vertical tangent at x = 0, so that many bisections (small intervals) are required to reproduce the function near this point. For limit = 0 and delta = $1_{10}-5$ we required 10 bisections, but for delta = $1_{10}-7$ we have not attained the necessary accuracy.

3. SOLUTION OF DIFFERENTIAL EQUATIONS

3.1. Ordinary Differential Equations.

A differential equation is an equation which contains a derivative of a function. The numerical problem in connection with the use of differential equations will normally consist in calculating back to the values of the original function on the basis of the information contained in the differential equation. A very large number of physical and chemical laws may be expressed as differential equations and their solution is, therefore, of considerable importance in practical calculations.

The most important classification of differential equations divides them into ordinary differential equations and partial differential equations. Ordinary differential equations occur when the original function contains only a single independent variable, whereas partial differential equations concern functions of several variables. The latter are discussed in section 3. 2.

We now consider the ordinary differential equations. A given function:

$$y = F(x)$$

normally has a derivative of the first order:

$$\frac{dy}{dx} = f1(x, y)$$

and of the second order:

$$\frac{d2y}{dx2} = f2(x, y)$$

etc. For simplicity we write the first-order derivative as dy/dx, the second-order derivative as d2y/dx2, etc.

A differential equation is said to be of the order: N, if the highest order of the derivatives occurring in the equation is N.

A general expression for a differential equation of the order N is:

f(x, y, dy/dx, d2y/dx2, ..., dNy/dxN) = 0

in which f denotes an arbitrary functionality.

A differential equation of the order N may always be transformed into N simultaneous differential equations of the first order. We simply introduce derivatives as new functions. As an example we take the second-order equation:

d2y/dx2 = 3*sin(x)

Here we must find the function, y, which differentiated twice yields the function 3*sin(x). We introduce a new function, z, which is the first-order derivative of y: dy/dx, and we then have the two simultaneous equations:

dy/dx = z
dz/dx = 3*sin(x)

The problem is now to find the two unknown functions, y and z, which satisfy these two equations.

In the following we assume that the differential equations are written in an explicite form, i. e. the left-hand side of the equation is a first-order derivative and the right-hand side is an expression which only contains the independent variable, x, and the unknown functions. Three differential equations of this type containing three unknown functions may be written:

dy1/dx = F1(x, y1, y2, y3)
dy2/dx = F2(x, y1, y2, y3)
dy3/dx = F3(x, y1, y2, y3)

For simple differential equations we may often be able to find an analytic expression as a solution to the equation. For more complicated differential equations (and also for many simple equations) no analytic solution can be found. There are certain rules which can be used as a guide to find analytic solutions, but the rules are not complete, i. e. we can never be sure to get a clear answer to the question of whether a given differential equation has an analytic solution or not. In special cases where no analytic solution is known, the unknown solution has been given a special name and then further investigated in the form of table calculation, etc. This applies to Bessel functions and to many others.

When no analytic solution is known, we must use a numerical solution with a stepwise calculation of the solution. In the following we describe two numerical methods: The method of Euler which is very easy to understand but not very accurate, and the method of Runge-Kutta which is more complicated and more economical in use.

<u>3.1.1. Method of Euler.</u> This is the very simple and elementary method in which the significance of the derivative is utilized directly. If we have the differential equation:

dy/dx = f(x, y)

and know the value of the function y at the point x0:

y = y0 for x = x0

we may calculate the function value at the point x1 = x0 + delx from the formula:

```
y1 := y0 + f(x0, y0)×delx;
```

The increment in the y-function is simply calculated as the derivative multiplied by the increment in x. In the point (x1, y1) we calculate the derivative again and we can then make a new step of the length delx. This takes us to the point x2 = x0 + 2×delx with the y-value:

```
y2 := y1 + f(x1, y1)×delx;
```

We continue in this way until the required end point has been reached. This is the method of Euler. In view of the very simple nature of this method it is not necessary to have a special procedure for using it.

The formula used:

```
y1 := y0 + f(x0, y0)×delx;
```

is the more accurate the smaller the value of delx. If delx is not quite small, the formula must also contain the second-order derivative, and maybe the third-order derivative, etc. We may put up the so-called Taylors formula, which is an infinite series:

```
        y1 := y0 + (dy/dx)×delx
                  + 0.5×(dy2/dx2)×delx↑2
                  + ........
```

When we use the method of Euler, i.e. only the first-order derivative is included, it is always wise to carry out the calculation for different values of the step length, delx. When a further reduction of delx only gives a small change in the calculated y-value, delx is sufficiently small.

We now show a program example in which we solve the two simultaneous differential equations:

```
        dy1/dx = y1 - 3.75×y2
        dy2/dx = y1 - 3×y2
```

after the method of Euler. We start in the point x = 0 where we assume that the two y-values are known (y1 = y2 = 1). The integration is made from x = 0 to x = 1. We try 4 different values of the number of steps: 10, 100, 200, and 400. The program is:

Program d-209. Test of the method of Euler.

```
begin
   integer STEPS, i;
   real delx, x;
   array y, dely[1:2];
   outcr;
   outtext({<  STEPS    y[1]        y[2]});
   outcr;
   for STEPS := 10,  100, 200, 400 do
   begin
      outcr;
      output({-ndddd}, STEPS);
      delx := 1/STEPS;
      x := 0;
      y[1] := y[2] := 1;
      for i := 1 step 1 until STEPS do
      begin
         dely[1] := y[1] - 3.75×y[2];
```

```
          dely[2] := y[1] - 3×y[2];
          x := x + delx;
          y[1] := y[1] + delx×dely[1];
          y[2] := y[2] + delx×dely[2]
        end for i;
        output({-ndd.dddddd}, y[1], y[2])
      end for STEPS
    end program;
```

The output is:

```
  STEPS   y[1]        y[2]

    10  -0.305454  -0.004057
   100  -0.260843   0.028028
   200  -0.258476   0.029733
   400  -0.257297   0.030582
```

The difference in the results for 200 steps and 400 steps is small. Whether it is small enough, depends upon the use we are going to make of the results. A good survey of the accuracy of the calculations may often be obtained by plotting the calculated y-values as a function of delx. In the present case the values are very close to lying on a straight line. If we extrapolate from the two last y-values (200 steps and 400 steps) to zero step length we find:

```
      y[1]          y[2]
   -0.256118      0.031431
```

The two differential equations may also be solved analytically. We find:

```
      y1 := -1.25×exp(-0.5×x) + 2.25×exp(-1.5×x);
      y2 := -0.5×exp(-0.5×x) + 1.5×exp(-1.5×x);
```

Insertion of x = 1 gives:

```
      y[1]          y[2]
   -0.256120      0.031430
```

The error is about $2_{10}-6$. We have found this result on the basis of an integration with 200 steps and one with 400 steps, or a total of 600 calculations of the derivatives. In the next section we shall see how the method of Runge-Kutta requires fewer calculations of the derivatives.

<u>3.1.2. Method of Runge-Kutta.</u> In this more advanced integration method the procedure will perform some extra calculations of the derivative (varying x and y) in each integration step. On this basis the procedure builds up a polynomial approximation to the unknown function.

We use a fourth order method in a special variant which was first indicated by Merson (1960) and published as an algorithm by Naur (1961). In each integration step the derivative is calculated for five sets of values of x and y. The fifth calculation of the derivative makes it possible for the procedure to estimate the integration error. The required accuracy is specified as a parameter to the procedure, and it will then vary the step length, so that this accuracy is obtained. The procedure first tries to make the whole integration as a single step, and if this is too inaccurate, the step length is reduced. If the accuracy later becomes more than satisfactory, the step length is increased again.

At present we have four versions of the Runge-Kutta procedure. RUKU1 treats a single differential equation and RUKU2 several simultaneous equations. RUKU3 and RUKU4 are minor variants of RUKU2. In RUKU3 a test is made to see whether some of the functions fall outside a specified range. In RUKU4 the step length control is made for only one of the unknown functions, and the procedure is written in a more compact way.

<u>3.1.2.1. The procedure RUKU1.</u> This procedure integrates a single differential equation. The declaration is:

```
procedure RUKU1 (F, x, y, xe, delta, first);
value xe, delta, first;
real F, x, y, xe, delta;
boolean first;
begin
   boolean last;
   real h, x0, y0, ho3, k1, k3, k4, k5, eps;
   own real step;
   if first then
   begin
```

```
                try last:  last := true;
                h := xe - x
            end
            else
            begin
                h := step;
R:              if abs(xe - x) ≤ abs (h) then go to try last;
                last := false
            end;
            x0 := x;
            y0 := y;
            ho3 := h/3;
            k1 := F×ho3;
            x := x0 + ho3;
            y := y0 + k1;
            k3 := F×ho3;
            y := (k1 + k3)/2 + y0;
            k3 := F×ho3;
            x := h/2 + x0;
            y := 0.375×k1 + 1.125×k3 + y0;
            k4 := F×ho3;
            x := h + x0;
            y := 1.5×k1 - 4.5×k3 + 6×k4 + y0;
            k5 := F×ho3;
            eps := abs(0.2×k1 - 0.9×k3 + 0.8×k4 - 0.1×k5);
            if delta < eps then
            begin
                x := x0;
                y := y0;
                go to Q
            end;
            y := (k1 + k5)/2 + 2×k4 + y0;
            if last then
            begin
                if first then step := h;
                go to finis
            end;
            if eps = 0 then go to try last;
Q:          step := h := (delta/eps)↑0.2×0.8×h;
            first := false;
            go to R;
finis:  end of RUKU-1;
```

The procedure has the six formal parameters:

real F: This expression must yield the derivative, dy/dx, and will normally contain x or y or both. The parameter is called by name and is calculated many times in a single call of the procedure for different values of x and y.

real x, y: These are the independent variable (x) and the unknown function (y). Before the call of the procedure we must have inserted start values of x and y. When the procedure is finished, x contains the specified end value, xe, and y contains the corresponding value of the unknown function.

real xe: The required end value of x.

real delta: The required integration accuracy.

boolean first: This must be specified as true when the procedure is called the first time, and as false at later calls. The procedure stores the last used step length as an own real: step, and this will be tried at once, if first is false. Otherwise, the procedure begins with a single step of the length xe - x.

We first give a simple example of the use of RUKU1. We wish to integrate the differential equation:

$$dy/dx = -x \times y$$

from x = 0 to x = 1. The start value of y for x = 0 is 1. The value of y in the intermediate points: 0.1, 0.2, 0.3, etc. must be printed. The example is taken from Bennett et al. (1956), page 72.

The program is:

```
Program d-217. Test of RUKU1.
begin
   boolean FIRST;
   real x, y;
   integer i;
   comment library RUKU1;
   procedure PRINTING;
   begin
      outcr;
      output(⟨-n.d⟩, x);
      output(⟨-ndd.dddddddd⟩, y)
   end PRINTING;
   outcr;
   outtext({< x     y});
   outcr;
   x := 0;
   y := 1;
   FIRST := true;
   PRINTING;
   for i := 1 step 1 until 10 do
   begin
      RUKU1(-x×y, x, y, 0.1×i, 1₁₀-6, FIRST);
      PRINTING;
      FIRST := false
   end for i
end program;
```

The output is:

```
  x      y

 0.0    1.00000000
 0.1    0.99501248
 0.2    0.98019868
 0.3    0.95599748
 0.4    0.92311635
 0.5    0.88249690
 0.6    0.83527020
 0.7    0.78270451
 0.8    0.72614899
 0.9    0.66697674
 1.0    0.60653057
```

We see that it is very easy to use this procedure. We simply write the derivative (here: -x×y) at the proper place in the parameter list. If the derivative is more complicated than here, it may be necessary to declare a special procedure (a real procedure) the value of which is equal to the derivative. An example of this type is shown in the next program, where we wish to print the value of x, y, and dy/dx every time the derivative is calculated. An extra new line is inserted after every fifth calculation of dy/dx, so that we can see how the procedure operates in cycles of five calculations of the derivative.

The program is:

```
Program d-213. Detailed test of RUKU1;
begin
   boolean FIRST;
   real x, y, z;
   integer count;
   comment library RUKU1;
   procedure PRINTING;
   begin
      outcr;
      output(⟨-n.dddddd⟩, x);
      output(⟨-ndd.dddddddd⟩, y)
   end PRINTING;
   real procedure dydx;
   begin
      count := count + 1;
      if count:5×5 = count then outcr;
      PRINTING;
      dydx := z := - x×y;
      output(⟨-ndd.dddddddd⟩, z)
   end dydx;
   outcr;
   outtext(⟨⟨  x            y               dydx⟩);
   outcr;
   x := 0;
   y := 1;
   FIRST := true;
   PRINTING;
   count := -1;
   RUKU1(dydx, x, y, 1, 1₁₀-6, FIRST);
   PRINTING
end program;
```

The output is:

x	y	dydx
0.000000	1.00000000	
0.000000	1.00000000	0.00000000
0.333333	1.00000000	-0.33333333
0.333333	0.94444444	-0.31481481
0.500000	0.88194444	-0.44097222
1.000000	0.59027778	-0.59027778
0.000000	1.00000000	0.00000000
0.052221	1.00000000	-0.05222125
0.052221	0.99863647	-0.05215004
0.078332	0.99693624	-0.07809188
0.156664	0.98778670	-0.15475035
0.000000	1.00000000	0.00000000
0.032878	1.00000000	-0.03287818
0.032878	0.99945951	-0.03286040
0.049317	0.99878456	-0.04925732
0.098635	0.99514481	-0.09815564
0.098635	0.99514743	-0.09815589
0.128570	0.99220911	-0.12756799
0.128570	0.99176889	-0.12751139
0.143537	0.98975135	-0.14206616
0.188440	0.98240016	-0.18512341
0.188440	0.98240188	-0.18512374
0.218071	0.97691645	-0.21303722
0.218071	0.97650290	-0.21294704
0.232887	0.97324625	-0.22665606
0.277333	0.96227166	-0.26687005
0.277333	0.96227318	-0.26687047
0.307414	0.95424550	-0.29334857
0.307414	0.95384726	-0.29322615
0.322455	0.94933977	-0.30611892
0.367576	0.93467401	-0.34356347

0.367576	0.93467542	-0.34356399
0.398589	0.92402026	-0.36830460
0.398589	0.92363662	-0.36815168
0.414096	0.91783481	-0.38007182
0.460616	0.89934801	-0.41425452

0.460616	0.89934930	-0.41425512
0.493171	0.88586325	-0.43688244
0.493171	0.88549493	-0.43670080
0.509449	0.87829817	-0.44744803
0.558281	0.85569582	-0.47771897

0.558281	0.85569693	-0.47771959
0.593459	0.83889177	-0.49784802
0.593459	0.83853773	-0.49763791
0.611048	0.82970091	-0.50698719
0.663815	0.80225709	-0.53255025

0.663815	0.80225788	-0.53255077
0.704464	0.78061043	-0.54991160
0.704464	0.78025758	-0.54966303
0.724788	0.76900417	-0.55736489
0.785761	0.73439373	-0.57705779

0.785761	0.73439365	-0.57705772
0.857174	0.69318418	-0.59417935
0.857174	0.69257282	-0.59365531
0.892880	0.67124600	-0.59934239
1.000000	0.60655053	-0.60655053

0.785761	0.73439365	-0.57705772
0.828555	0.70969896	-0.58802455
0.828555	0.70946430	-0.58783013
0.849952	0.69683299	-0.59227458
0.914143	0.65847691	-0.60194218

0.914143	0.65847474	-0.60194020
0.942762	0.64124785	-0.60454419
0.942762	0.64121059	-0.60450906
0.957072	0.63255170	-0.60539726
1.000000	0.60653128	-0.60653128
1.000000	0.60653050	

In the first cycle the following x-values have been used:

x0, x0 + h/3, x0 + h/3, x0 + h/2, x0 + h

Here, x0 is the start value and the step length, h, is xe - x0. Of the corresponding y-values the first is the start value, y0, and the second is y0 + (dy/dx)×h/3. The last three y-values are more complicated and may be seen from the ALGOL procedure.

In the first cycle we use the full step length, h = 1, but this is not sufficiently accurate. The second cycle is made with h = 0.1566 and the third cycle with h = 0.0986. Now, the accuracy is satisfactory, and in the fourth cycle we integrate from x = 0.0986 to x = 0.1884. In the following cycles the step length is approximately constant, about 0.1.

In cycle no. 11 we have reached x = 0.7858, and the procedure now tries to complete the integration in a single step. But this cannot be done, and we get cycle 12 with the step length 0.1. Finally, the last step is made in cycle 13.

3.1.2.2. The procedure RUKU2. This is a simple extension of RUKU1, now integrating several simultaneous differential equations instead of a single equation. The declaration is:

```
    procedure RUKU2 (var, N, F, x, y, xe, delta, first);
    value N, xe, first;
    boolean first;
    integer var, N;
    real F, x, xe, delta;
    array y;
    begin
       boolean last, good;
       real h, x0, ho3, eps, epsmax, min, D;
       own real step;
       array y0[1:N], k[1:4, 1:N];
       if first then
       begin
TL:       last := true;
          h := xe - x
       end
```

```
        else
        begin
           h := step;
R:         if abs(xe - x) < abs (h) then go to TL;
           last := false
        end;
        x0 := x;
        for var := 1 step 1 until N do y0[var] := y[var];
        ho3 := h/3;
        for var := 1 step 1 until N do k[1, var] := F×ho3;
        x := x0 + ho3;
        for var := 1 step 1 until N do  y[var]  := y0[var]  +  k[1,   var];
        for var := 1 step 1 until N do k[2, var] := F×ho3;
        for var := 1 step 1 until N do y[var] := (k[1, var] + k[2, var])/2 + y0[var];
        for var := 1 step 1 until N do k[2, var] := F×ho3;
        x := h/2 + x0;
        for var := 1 step 1 until N do  y[var]  :=  0.375×k[1,   var]  +
                    1.125×k[2, var] + y0[var];
        for var := 1 step 1 until N do k[3, var] := F×ho3;
        x := h + x0;
        for var := 1 step 1 until N do  y[var]  :=  1.5×k[1,   var]  -
                    4.5×k[2, var] + 6×k[3, var] + y0[var];
        for var := 1 step 1 until N do k[4, var] := F×ho3;
        good := true;
        for var := 1 step 1 until N do
        begin
           eps := abs(0.2×k[1, var] - 0.9×k[2, var] + 0.8×k[3, var] - 0.1×k[4, var]);
           D := delta;
           if var = 1 then
           begin
              min := if eps = 0 then 100 else D/eps;
              epsmax := eps
           end;
           if eps > epsmax then epsmax := eps;
           if D < eps then
           begin
              good := false;
              if D/eps < min then min := D/eps
           end for if
        end for var;
        if ¬, good then
        begin
           x := x0;
           for var := 1 step 1 until N do y[var] := y0[var];
           go to Q
        end;
        for var := 1 step 1 until N do y[var] := (k[1, var] +
                    k[4, var])/2 + 2×k[3, var] + y0[var];
        if last then
        begin
           if first then step := h;
           go to finis
        end;
        if epsmax = 0 then go to TL;
Q:      step := h := min↑0.2×0.8×h;
        first := false;
        go to R;
finis: end of RUKU-2;
```

The formal parameters in RUKU2 are:

integer var: This is a running index used by the procedure for counting the dependent variables (unknown functions).

integer N: The number of simultaneous differential equations. N may be equal to 1.

real F: This expression must yield the value of the derivatives. F will normally be an expression containing the variables: x, y, and var.

real x: The independent variable. The start value must have been inserted before the call of the procedure, and after the call x will be equal to xe.

array y[1:N]: These are the dependent variables. The N start values must have been inserted before the call, and the required end values will be found here after the call.

real xe: The required end value of x.

real delta: The required integration accuracy. If this is the same for all N unknown functions, we simply write this value here. If different accuracies are required for the various functions, we must write delta as an expression or a real procedure containing the running index, var.

boolean first: The same as in RUKU1.

As an example of the use of RUKU2 we show the program d-215 below which solves the two simultaneous differential equations:

```
dy1/dx = y1 - 3.75×y2
dy2/dx = y1 - 3×y2
```

The same equations were solved on page 16 after the method of Euler in program d-209. The example is taken from Bennett et al. (1956), page 79. The new program is:

```
Program d-215. Test of RUKU2.
begin
   boolean FIRST;
   integer i, var;
   real x;
   array y[1:2];
   comment library RUKU2;
   procedure PRINTING;
   begin
      outcr;
      output(⟨-n.d⟩, x);
      output(⟨-ndd.dddddd⟩, y[1], y[2])
   end PRINTING;
   outcr;
   outtext(⟨<  x      y[1]          y[2]⟩);
   outcr;
   x := 0;
   y[1] := y[2] := 1;
   FIRST := true;
   PRINTING;
   for i := 1 step 1 until 10 do
   begin
      RUKU2(var, 2, if var = 1 then y[1] - 3.75×
            y[2] else y[1] - 3×y[2], x, y, 0.1×i, 1₁₀-6, FIRST);
      PRINTING;
      FIRST := false
   end for i
end program;
```

The output is:

```
 x      y[1]        y[2]

0.0   1.000000   1.000000
0.1   0.747556   0.815447
0.2   0.535795   0.658809
0.3   0.358779   0.526089
0.4   0.211413   0.413852
0.5   0.089324   0.319150
0.6  -0.011240   0.239446
0.7  -0.093500   0.172563
0.8  -0.160213   0.116632
0.9  -0.213744   0.070047
1.0  -0.256120   0.031430
```

The calculated result at x = 1 is in accordance with the analytic solution (see page 16).

In this example we could not see how many times the procedure had to calculate the derivatives in order to find the solution with the given accuracy. In the next test program (d-214) we calculate the same example for different values of delta. In each calculation we print a line with the value of delta, the number of calculations of the derivative (count), and the difference between the numerical solution and the true (analytic) solution. The program is:

Program d-214. Test of RUKU2.

```
begin
   boolean FIRST;
   integer count, i, d;
   real x, delta;
   array y, yanal[1:2];
   comment library RUKU2;
   real procedure dydx;
   begin
      count := count + 1;
      if count>1000 then go to EXIT;
      dydx := if i = 1 then
      y[1] - 3.75×y[2]
      else
```

```
    y[1] - 3×y[2]
  end dydx;
  yanal[1] := -1.25×exp(-0.5) + 2.25×exp(-1.5);
  yanal[2] := -0.5×exp(-0.5) + 1.5×exp(-1.5);
  outcr;
  outtext({<Delta count Error1   Error2>});
  outcr;
  for d := -1 step -1 until -8 do
  begin
     outcr;
     x := 0;
     y[1] := y[2] := 1;
     FIRST := true;
     count := 0;
     delta := 10↑d;
     RUKU2(i, 2, dydx, x, y, 1, delta, FIRST);
     output({-n⏨-d}, delta);
     output({-ndddd}, count);
     for i := 1,2 do
     output({-n.dd⏨-d}, outsp(1), y[i] - yanal[i])
  end for d;
EXIT: end program;
```

The output is:

```
Delta count Error1   Error2


 1⏨-1    10 -5.49⏨-3 -3.65⏨-3
 1⏨-2    30  2.53⏨-4  1.70⏨-4
 1⏨-3    40  1.15⏨-4  7.69⏨-5
 1⏨-4    50  2.42⏨-5  1.62⏨-5
 1⏨-5    70  5.07⏨-6  3.39⏨-6
 1⏨-6   100  9.51⏨-7  6.34⏨-7
 1⏨-7   150  1.69⏨-7  1.12⏨-7
 1⏨-8   220  2.70⏨-8  1.35⏨-8
```

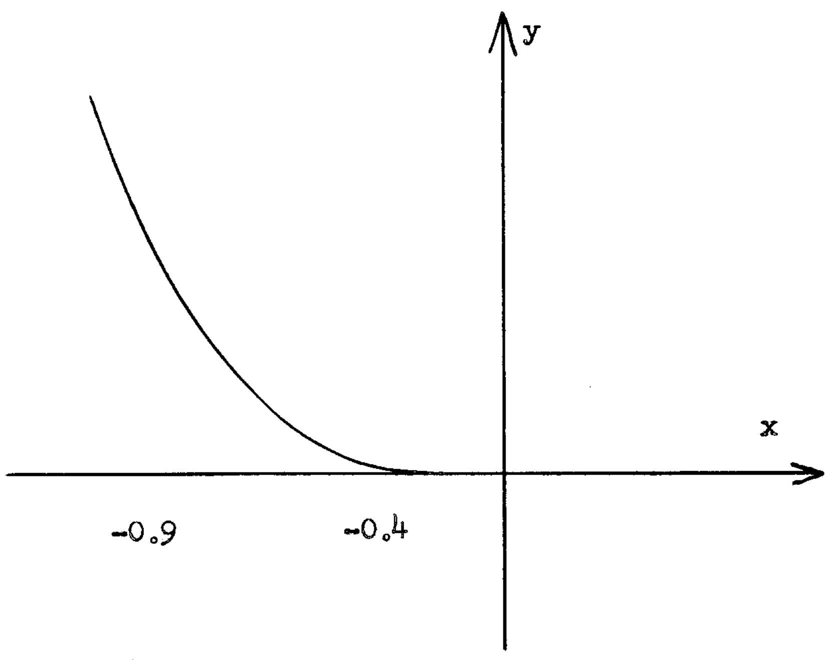
We see, that the calculation with delta = $1_{10}{-6}$ has required 100 calculations of the derivative. As each step makes five insertions in the derivative for each function, or a total of 10 insertions, count will always be a multiple of 10. The number of insertions is clearly smaller here than in the use of the method of Euler.

We also notice, that for relatively large values of delta the actual accuracy attained is somewhat bigger than the specified value. For small values of delta the accuracy is not quite satisfactory. For very small values of delta we simply cannot reach the required accuracy because of rounding errors. For difficult functions it is recommended to build in a counting variable (here: count) in the procedure for the derivative and to program a go to statement or the like if maximum value (here: 1000) is surpassed.

3.1.2.3. The procedure RUKU3. As the Runge-Kutta procedure calculates the derivative for many values of x and y, and as some of these insertions need not necessarily correspond to a point of the unknown function, we may risk that the derivative is to be calculated for such values of x and y where it is not defined. This risk is especially pronounced if we start with too long integration steps. As an example we take the differential equation:

dy/dx = x/(1 - sqrt(y))

We wish to calculate the function from $x = -0.9$ to $x = -0.4$ and with the start value $y = 0.9$ at $x = -0.9$. The function has the shape:



x	y
-0.9	0.90000
-0.8	0.44109
-0.7	0.25832
-0.6	0.14114
-0.5	0.06078
-0.4	0.00611

The curve is very steep at the start point. Incidentally, the equation may be solved analytically, but we are not concerned about this now. In the program below we wish to carry out the integration with RUKU2 from x = -0.9 to x = -0.4. The program is:

```
Program d-222. Test of RUKU2.
begin
   integer var;
   real x;
   array y[1:1];
   comment library RUKU2;
   x := -0.9;
   y[1] := 0.9;
   RUKU2(var, 1, x/(1-sqrt(y[1])), x, y, -0.4, 1₁₀-6, true);
   outcr;
   output(⟨-n.dddddd₁₀-d⟩, y[1])
end program;
```

When this program is run on the computer, we at once get the error message: sqrt, indicating that the machine has tried to calculate the square root of a negative number. The procedure increases x by one third of the interval length from - 0.9 to - 0.4, and at the same time y is increased by an amount corresponding to the derivative at x = - 0.9. The new y-value becomes:

$$0.9 + (-0.9)/(1-\mathrm{sqrt}(0.9))\times 1/3\times 0.5 = -2.02$$

which is negative and causes the error message.

It is easy to correct this shortcoming in the procedure. We have done this in the procedure RUKU3 which has the declaration:

```
    procedure RUKU3 (var, N, F, x, y, xe, delta, first, outside);
    value N, xe, first;
    boolean first, outside;
    integer var, N;
    real F, x, xe, delta;
    array y;
    begin
       boolean last, good;
       real h, x0, ho3, eps, epsmax, min, D;
       own real step;
       array y0[1:N], k[1:4, 1:N];
       if first then
       begin
TL:       last := true;
          h := xe - x
       end
       else
       begin
          h := step;
R:        if abs(xe - x) ≤ abs (h) then go to TL;
          last := false
       end;
       outside := false;
       x0 := x;
       for var := 1 step 1 until N do y0[var] := y[var];
       ho3 := h/3;
       for var := 1 step 1 until N do k[1, var] := F×ho3;
       x := x0 + ho3;
       for var := 1 step 1 until N do  y[var]  := y0[var]  +  k[1,    var];
       for var := 1 step 1 until N do k[2, var] := F×ho3;
       for var := 1 step 1 until N do y[var] := (k[1, var] + k[2, var])/2 + y0[var
       for var := 1 step 1 until N do k[2, var] := F×ho3;
       x := h/2 + x0;
       for var := 1 step  1 until  N do  y[var]  :=  0.375×k[1,    var]  +
                  1.125×k[2, var] + y0[var];
       for var := 1 step 1 until N do k[3, var] := F×ho3;
       x := h + x0;
       for var := 1 step  1 until  N do  y[var]  :=  1.5×k[1,    var]  -
                  4.5×k[2, var] + 6×k[3, var] + y0[var];
```

```
        for var := 1 step 1 until N do k[4, var] := F×ho3;
        good := true;
        for var := 1 step 1 until N do
        begin
           eps := abs(0.2×k[1, var] - 0.9×k[2, var] + 0.8×k[3, var] - 0.1×k[4, var]);
           D := delta;
           if var = 1 then
           begin
              min := if eps = 0 then 100 else D/eps;
              epsmax := eps
           end;
           if eps > epsmax then epsmax := eps;
           if D < eps then
           begin
              good := false;
              if D/eps < min then min := D/eps
           end for if
        end for var;
        if ¬ good ∨ outside then
        begin
           x := x0;
           for var := 1 step 1 until N do y[var] := y0[var];
           go to Q
        end;
        for var := 1 step 1 until N do y[var] := (k[1, var] +
                    k[4, var])/2 + 2×k[3, var] + y0[var];
        if last then
        begin
           if first then step := h;
           go to finis
        end;
        if epsmax = 0 then go to TL;
Q:      step := h := if outside then 0.5×h else min↑0.2×0.8×h;
        first := false;
        go to R;
finis:  end of RUKU3;
```

The parameters are the same as in RUKU2, but a new parameter has been added:

```
boolean outside;
```

This must be a simple logical variable (not a procedure). Before the calculation of each step RUKU3 sets the value of outside to false. The procedure F, which gives the value of the derivative, must in each call test if one or more of the dependent variables: y[1], y[2], etc. has a value outside a permissible range, so that the derivative becomes undefined. If this is the case, F must set outside to true. RUKU3 will then try again with half of the old step length. Here, too, it is recommended to use a counter to prevent the halving to go on for ever.

If we replace RUKU2 by RUKU3 in the program d-222 we get:

```
Program d-223. Test of RUKU3.
begin
   boolean outside;
   integer var;
   real x;
   array y[1:1];
   comment library RUKU3;
   real procedure DERIV;
   begin
      outside:= y[1]<0;
      DERIV := if outside then 0 else x/(1-sqrt(y[1]))
   end DERIV;
   x := -0.9;
   y[1] := 0.9;
   RUKU3(var, 1, DERIV, x, y, -0.4, 1₁₀-6, true, outside);
   outcr;
   output(⟨-n.dddddd₁₀-d⟩, y[1])
end program;
```

The calculation can now be carried through and we get the result:

```
6.108760₁₀-3
```

3.1.2.4. The procedure RUKU4. As the last variant of the Runge-Kutta procedures we discuss RUKU4, which contains only small changes from RUKU3. The declaration is:

```
 procedure RUKU4(var, N, START, F1, F, x, y, xe, delta, crit, first, outside);
 value N, xe, delta, crit;
 boolean first, outside;
 integer var, N, crit;
 real F1, F, x, xe, delta;
 array y;
 procedure START;
 begin
   boolean last;
   real h, x0, ho3, eps;
   own real step;
   array y0[1:N], k[1:4, 1:N];
   integer i;
   procedure NEW;
   begin
     START;
     k[i,1] := F1×ho3;
     for var := 2 step 1 until N do
     k[i, var] := F×ho3;
     i := i + 1
   end NEW;
   procedure SET(E);
   real E;
   for var := 1 step 1 until N do y[var] := E;
   if first then
   begin
TL:  last := true;
     h := xe - x
   end
   else
   begin
     h := step;
R:   if abs(xe-x) ≤ abs (h) then go to TL;
     last := false
   end;
   outside := false;
   x0 := x;
   for var := 1 step 1 until N do y0[var] := y[var];
   ho3 := h/3;
   i := 1;
```

```
    NEW;
    x := x0 + ho3;
    SET(y0[var] + k[1, var]);
    NEW;
    SET((k[1, var] + k[2, var])/2 + y0[var]);
    i := 2;
    NEW;
    x := h/2 + x0;
    SET(0.375×k[1, var] + 1.125×k[2, var] + y0[var]);
    NEW;
    x := h + x0;
    SET(1.5×k[1, var] - 4.5×k[2, var] + 6×k[3, var] + y0[var]);
    NEW;
    eps := abs(0.2×k[1, crit] - 0.9×k[2, crit] + 0.8×k[3, crit] - 0.1×k[4, crit]);
    if delta < eps ∨ outside then
    begin
       x := x0;
       SET(y0[var]);
       go to Q
    end;
    SET((k[1, var] + k[4, var])/2 + 2×k[3, var] + y0[var]);
    if last then
    begin
       if first then step := h;
       go to finis
    end;
    if eps = 0 then go to TL;
Q:     step  :=  h  := if  outside then  0.5×h else  (delta/eps)↑0.2×0.8×h;
    first := false;
    go to R;
finis: end RUKU4;
```

The parameters in RUKU4 are:

integer var, N: As in RUKU2 and RUKU3.

procedure START: This procedure is called once by RUKU4 before the carrying out of the for-statement:

```
        for var := 1 step 1 until N do
```

START can be used to transfer the actual set of y-values as simple variables:

```
z := y[1];
tcat := y[2];
etc.
```

The procedures F1 and F which calculate the derivatives may then operate upon the simple variables instead of the y-values. This saves some computer time. Counting and outside check may also be placed in START.

real F1: This expression or real procedure must yield the derivative of the first dependent variable, i.e. for var = 1. In many calculations involving reaction kinetics the first derivative is a complicated rate expression whereas the others are simpler (often linear).

real F: This expression must give the derivative for var = 2, 3, etc.

real x: As in RUKU2 and RUKU3.

array y[1:N]: As in RUKU2 and RUKU3.

real xe: As in RUKU2 and RUKU3.

real delta: The required integration accuracy. This is controlled for only one of the functions, for var = crit. In RUKU2 and RUKU3 delta may be a real procedure dependent upon var.

integer crit: This is the number of the function for which the accuracy delta is controlled ($1 \leq$ crit $\leq$ N).

boolean first: As in RUKU2 and RUKU3.

boolean outside: As in RUKU3.

In RUKU4 we have made the change that the many for-statements of the form:

```
for var := 1 step 1 until N do
```

have been replaced by two local procedures: NEW and SET. This change makes the procedure smaller, but the calculation time is somewhat longer because of the many procedure calls. If the core space available is small, RUKU4 may be faster than RUKU3.

For comparison of RUKU3 and RUKU4 we show two programs, d-218 and d-219, which both illustrate the solution of the same problem: the four simultaneous differential equations:

$$dz/dx = 1_{10}12\times\exp(-20135/(tcat+273))\times(1.5-z-0.0025\times tcat)$$

$$dtcat/dx = 900\times(dz/dx)-0.24\times(tcat-ttub)-0.01\times(tcat-tann)$$

$$dttub/dx = -0.24\times(tcat-ttub)$$
$$dtann/dx = 0.01\times(tcat-tann)$$

Here z is the ammonia mole fraction, and the three other variables: tcat, ttub, and tann, are the temperatures in different channels of an ammonia converter.

Start values of the variables are:

```
z := 0.03;
tcat := ttub := 400;
tann := 20;
```

We integrate from x = 0 to x = 3. Note, that the rate expression for dz/dx is only an approximation to the correct formula. The factor:

$$1.5 - z - 0.0025\times tcat$$

is a linear approximation to the equilibrium curve, which is far from being correct.

The first program is:

```
Program d-218. Test of RUKU3.
begin
   integer var, count;
   real z, tcat, ttub, tann, dfdx, x;
   array y[1:4];
   comment library RUKU3;
   real procedure DERIV;
   begin
      if var = 1 then
      begin
         count := count + 1;
         z := y[1];
         tcat := y[2];
         ttub := y[3];
         tann := y[4];
```

```
          DERIV := dfdx := -1₁₀12×exp(-20135/(tcat+273))×(z+0.0025×tcat-1.5)
        end
        else
        if var = 2 then
        DERIV := 900×dfdx - 0.24×(tcat-ttub) - 0.01×(tcat-tann)
        else
        if var = 3 then
        DERIV := -0.24×(tcat-ttub)
        else
        DERIV := 0.01×(tcat-tann)
     end DERIV;
     x := 0;
     y[1] := 0.03;
     y[2] := y[3] := 400;
     y[4] := 20;
     count := 0;
     RUKU3(var, 4, DERIV, x, y, 3,
     if var = 1 then 1₁₀-5 else 0.01, true, false);
     outcr;
     output(⟨-ndddd⟩, count);
     for var := 1 step 1 until 4 do
     output(⟨-nddd.ddd00⟩, y[var])
  end program;
```

The calculation time is 23 seconds and the result is:

```
     190    0.25124  496.1349   310.9243    33.90138
```

The other program is:

```
  Program d-219.  Test of RUKU4.
  begin
     integer var, count;
     real z, tcat, ttub, tann, dfdx, x;
     array y[1:4];
     comment library RUKU4;
     procedure START;
     begin
        count := count + 1;
        z := y[1];
        tcat := y[2];
```

```
    ttub := y[3];
    tann := y[4]
  end START;
  real procedure F1;
  F1 := dfdx := -1₁₀12×exp(-20135/(tcat + 273))×(z + 0.0025×tcat - 1.5);
  real procedure F;
  F := if var = 2 then
  900×dfdx - 0.24×(tcat-ttub) - 0.01×(tcat-tann)
  else
  if var = 3 then -0.24×(tcat-ttub)
  else 0.01×(tcat-tann);
  x := 0;
  y[1] := 0.03;
  y[2] := y[3] := 400;
  y[4] := 20;
  count := 0;
  RUKU4(var, 4, START, F1, F, x, y, 3, 1₁₀-5, 1, true, false);
  outcr;
  output(⌊-ndddd⌋, count);
  for var := 1 step 1 until 4 do
  output(⌊-nddd.ddd00⌋, y[var])
end program;
```

For this program the calculation time was 24 seconds, and the result was the same as for the first program.

3.1.3. Boundary conditions. In the previous examples we have assumed that the value of all the dependent variables (unknown functions) was known at the start point. In the example with the two equations:

```
dy1/dx = y1 - 3.75×y2
dy2/dx = y1 - 3×y2
```

we used the start values:

```
y1 := 1;
y2 := 1;
```

at x = 0. The integration was made from x = 0 to x = 1. The situation is then:

x:	0	1
y1:	known: 1	calculated: -0.256
y2:	known: 1	calculated: 0.031

In many practical problems we have the so-called two-point boundary condition, i.e. some of the y-values are known at one of the end points (x = 0) and the others at the other end point (x = 1):

x:	0	1
y1:	known: 1	unknown:
y2:	unknown:	known: 0.031

Here we know the value of y1 at x = 0 and of y2 at x = 1. As we do not know the value of y2 at x = 0, we cannot start the integration.

A simple way to solve the problem is the following. We make a guess of the start value of y2 at x = 0, carry out the integration from x = 0 to x = 1 as usual, and check the value of y2 calculated for x = 1. This value will normally be different from the required value, and we can calculate the error, i.e. the difference between the calculated value and the required value at the end point. We must then vary the start value, until the error becomes zero. This is obviously a root determination, and in this case for a function of one variable. The problem may be solved with the procedures discussed in chapter 4 of volume 2, e.g. the procedure NOLEQ3 for several unknowns.

As an example we show the program below. Here both start values are unknown and we wish to obtain the end values:

```
y1 := 0.21;
y2 := 0.28;
```

at x = 1. We could have started at the point x = 1 without any guess and integrated backwards to x = 0, but the program illustrates the essential feature of the problem: one or more of the start values must be guessed and the errors calculated after the integration. The guessed start values are adjusted until the errors become zero. It should be noted that this guessing method can also be used in more complicated cases. As an example we may require that the y1-value at x = 0.5 must be 17 times the y2-value at x = 0.8. We must then use the error in this condition as a feed-back to NOLEQ3.

The program is:

Program d-220. Integration with boundary conditions.

```
begin
   boolean fin, FIRST;
   integer count, i, var;
   real x;
   array xstart, del0x,  xact,  epsx,  yact,  y0[1:2],  yold[1:2,  1:2];
   comment library INVERT2;
   comment library NOLEQ3;
   comment library RUKU2;
   procedure PRINTING;
   if fin ∨ i = 0 ∨ i = 10 then
   begin
      outcr;
      output(⟨-n.d⟩, x);
      output(⟨-ndd.dddddd⟩, yact[1], yact[2])
   end PRINTING;
   xstart[1] := xstart[2] := 3;
   del0x[1] := del0x[2] := 0.1;
   epsx[1] := epsx[2] := 1₁₀-4;
   count := 0;
   outcr;
   outtext(⟨<  x      y[1]          y[2]⟩);
   outcr;
   fin := false;
   FIRST := true;
H1: NOLEQ3(2, count, 50, false, xstart, del0x, xact, epsx, yact, yold, y0,
           1₁₀-8, 10, F1, E1);
   go to G2;
E1: outcr;
   outtext(⟨<ERROR⟩);
F1: fin := true;
G2: x := 0;
   yact[1] := xact[1];
   yact[2] := xact[2];
   outcr;
   i := 0;
   PRINTING;
```

```
    for i := 1 step 1 until 10 do
    begin
       RUKU2(var, 2, if var = 1 then
       yact[1] - 3.75×yact[2] else yact[1] - 3×yact[2], x, yact, 0.1×i, 1₁₀-6, FIRST);
       PRINTING;
       FIRST := false
    end for i;
    yact[1] := yact[1] - 0.21;
    yact[2] := yact[2] - 0.28;
    if ¬ fin then go to H1
 end program;
```

The output is:

```
  x      y[1]          y[2]



 0.0    3.000000    3.000000
 1.0   -0.768360    0.094291

 0.0    3.100000    3.000000
 1.0   -0.650197    0.132631

 0.0    3.000000    3.100000
 1.0   -0.912135    0.059094

 0.0    2.503107    2.000000
 1.0    0.082247    0.255752

 0.0    2.603107    2.000000
 1.0    0.200410    0.294092

 0.0    2.503107    2.100000
 1.0   -0.061528    0.220555

 0.0    2.428459    1.849794
 1.0    0.210000    0.280000

 0.0    2.528459    1.849794
 1.0    0.328163    0.318340
```

```
0.0     2.428459    1.949794
1.0     0.066225    0.244803

0.0     2.428459    1.849794
0.1     2.011841    1.560791
0.2     1.657076    1.313573
0.3     1.355362    1.102244
0.4     1.099131    0.921734
0.5     0.881881    0.767684
0.6     0.698020    0.636343
0.7     0.542746    0.524488
0.8     0.411931    0.429345
0.9     0.302031    0.348532
1.0     0.210000    0.280000
```

The program prints the estimated start values and the calculated end values for each integration. When the adjustment is finished, the intermediate results from the integration are also printed.

In the example shown here we can always calculate the derivatives, even if the start values are not correct. We could, therefore, use the simpler RUKU2 instead of RUKU3. In other cases we must use RUKU3 to get the necessary control that we do not enter a region where the derivatives are undefined. An especially difficult case occurs if the derivatives become undefined because the start values are not correct. It is then of no use that RUKU3 reduces the step lentgh if we have not started correctly. We must then interrupt the integration, but now we get new troubles because the y-value we have reached is not the proper y-value to be used for the calculation of the error.

If there is only one boundary condition, i.e. a root determination in a function of a single variable, we can use the procedure ROOT7, which is described in section 4.3.3. in volume 2.

As an example of this we show the program d-221 which solves the two simultaneous differential equations:

$$dy1/dr = -\ 1_{10}8 \times y2/r\uparrow 2$$
$$dy2/dr = \ 1_{10}6 \times r\uparrow 2/sqrt(y1)$$

The independent variable is r, and we must integrate from r = 1 to r = 0. The example illustrates the problems which occur in connection with the calculation of diffusion in catalyst particles, where we integrate from the surface of the sphere (r = 1) to its center (r = 0). The two unknown functions are:

y1: The mole fraction of the key component.

y2: The amount of product formed which flows through a spherical surface in the distance, r.

The first differential equation is a mass balance and the second equation is a reaction rate expression. We have taken a rate expression where the reaction rate is inversely proportional to the square root of the mole fraction of the key component, i.e. the component which has the largest influence upon the reaction rate.

This is a typical two-point boundary value problem. We know y1 at the surface ($r = 1$), where it must be equal to the concentration in the gas phase (we neglect a small difference caused by the mass transfer restriction), and we know y2 at the center ($r = 0$), where it must be zero. The value of y2 at $r = 1$ must be guessed, and we find it by means of the procedure ROOT7.

The program is:

Program d-221. Boundary condition solution with ROOT7.

```
begin
   boolean fin, FIRST, outside, trouble;
   integer cact, var, outcount, rootcount;
   real x, r;
   array y[1:2];
   comment library ROOT7;
   comment library RUKU3;
   real procedure Y;
   begin
      real procedure delta;
      begin
         if outcount > 5 then go to EX1;
         delta := if var = 1 then 1₁₀-4 else 1₁₀-8
      end delta;
      procedure PRINTING;
      begin
         outcr;
         output({-n.d}, r);
         for var := 1, 2 do
```

```
              output(⟨-n.dddddd₁₀-dd⟩, outsp(1), y[var])
          end PRINTING;
          if rootcount > 16 then go to EX2;
          r := 1;
          y[1] := 0.05;
          y[2] := x;
          outcount := 0;
          rootcount := rootcount + 1;
          Y := 0;
          outcr;
          for cact := 10 step -1 until 0 do
          begin
             r := 0.1×cact;
             PRINTING;
             if cact ≥ 1 then
             RUKU3(var, 2, DERIV,  r,  y,  0.1×(cact-1), delta, true,  outside)
          end for cact;
          cact := 0;
  EX1:    Y := y[2]
      end Y;
      real procedure DERIV;
      begin
         DERIV := 0;
         if ¬ outside then
         begin
            if var = 1 then
            begin
               if y[1] ≤0 ∨ y[1] > 100 then
               begin
                  outside := true;
                  outcount := outcount + 1;
                  go to EX3
               end if outside;
               if r > 0 then DERIV := - 1₁₀8×y[2]/r↑2
            end if var = 1
            else
            DERIV := 1₁₀-6×r↑2/sqrt(y[1])
         end if not outside;
  EX3:   end DERIV;
     outcr;
     outtext(⟨⟨ r    y[1]          y[2]⟩⟩);
```

```
   rootcount := 0;
   x := 2.3⏨-7;
   ROOT7(Y, x, 2⏨-9, 2⏨-7, 3⏨-7, 2⏨-8, 1⏨-10, 10, 0, cact, trouble);
EX2: end program;
```

The output is:

```
 r   y[1]           y[2]

1.0  5.000000 ⏨-2  2.300000 ⏨-7
0.9  1.799041        1.177291 ⏨-7
0.8  3.069973        7.099623 ⏨-8
0.7  4.040609        4.105341 ⏨-8
0.6  4.749731        2.085167 ⏨-8
0.5  5.189132        7.255703 ⏨-9
0.4  5.297444       -1.600572 ⏨-9
0.3  4.896636       -7.024963 ⏨-9
0.2  3.413194       -1.005797 ⏨-8

1.0  5.000000 ⏨-2  2.320000 ⏨-7
0.9  1.824574        1.202580 ⏨-7
0.8  3.133527        7.391848 ⏨-8
0.7  4.159740        4.433640 ⏨-8
0.6  4.951318        2.447961 ⏨-8
0.5  5.517562        1.121902 ⏨-8
0.4  5.832890        2.691570 ⏨-9
0.3  5.804693       -2.405162 ⏨-9
0.2  5.123732       -5.085282 ⏨-9
0.1  2.189159       -6.221791 ⏨-9

1.0  5.000000 ⏨-2  2.340000 ⏨-7
0.9  1.850059        1.227783 ⏨-7
0.8  3.196838        7.682119 ⏨-8
0.7  4.278150        4.758534 ⏨-8
0.6  5.151197        2.805417 ⏨-8
0.5  5.842102        1.510206 ⏨-8
0.4  6.359428        6.864866 ⏨-9
0.3  6.691434        2.036745 ⏨-9
```

0.2	6.774128	-4.003181$_{10}$-10
0.1	6.251960	-1.304835 $_{10}$-9

1.0	5.000000 $_{10}$-2	2.360000 $_{10}$-7
0.9	1.875498	1.252903 $_{10}$-7
0.8	3.259915	7.970515 $_{10}$-8
0.7	4.395870	5.080198 $_{10}$-8
0.6	5.349405	3.157866 $_{10}$-8
0.5	6.162903	1.891151 $_{10}$-8
0.4	6.877797	1.093313 $_{10}$-8
0.3	7.559406	6.330706 $_{10}$-9
0.2	8.375670	4.075202 $_{10}$-9
0.1	1.013150 $_{10}$ 1	3.292568 $_{10}$-9

1.0	5.000000 $_{10}$-2	2.350000 $_{10}$-7
0.9	1.862784	1.240353 $_{10}$-7
0.8	3.228405	7.826547 $_{10}$-8
0.7	4.337095	4.919759 $_{10}$-8
0.6	5.250505	2.982246 $_{10}$-8
0.5	6.002954	1.701558 $_{10}$-8
0.4	6.619596	8.911349 $_{10}$-9
0.3	7.127625	4.200636 $_{10}$-9
0.2	7.580470	1.860279 $_{10}$-9
0.1	8.210950	1.024118 $_{10}$-9

1.0	5.000000 $_{10}$-2	2.345000 $_{10}$-7
0.9	1.856423	1.234071 $_{10}$-7
0.8	3.212629	7.754391 $_{10}$-8
0.7	4.307644	4.839246 $_{10}$-8
0.6	5.200903	2.893986 $_{10}$-8
0.5	5.922643	1.606107 $_{10}$-8
0.4	6.489763	7.891286 $_{10}$-9
0.3	6.910072	3.123096 $_{10}$-9
0.2	7.178719	7.360357$_{10}$-10
0.1	7.236558	-1.320227$_{10}$-10

1.0	5.000000 $_{10}$-2	2.347500 $_{10}$-7
0.9	1.859604	1.237212 $_{10}$-7
0.8	3.220519	7.790483 $_{10}$-8
0.7	4.322374	4.879527 $_{10}$-8
0.6	5.225716	2.938154 $_{10}$-8
0.5	5.962827	1.653888 $_{10}$-8
0.4	6.554741	8.402099 $_{10}$-9
0.3	7.018989	3.662945 $_{10}$-9
0.2	7.379973	1.299590 $_{10}$-9
0.1	7.725017	4.479860 $_{10}$-10

1.0	5.000000 $_{10}$-2	2.346250 $_{10}$-7
0.9	1.858014	1.235642 $_{10}$-7
0.8	3.216574	7.772441 $_{10}$-8
0.7	4.315010	4.859393 $_{10}$-8
0.6	5.213313	2.916080 $_{10}$-8
0.5	5.942742	1.630011 $_{10}$-8
0.4	6.522268	8.146891 $_{10}$-9
0.3	6.964566	3.393294 $_{10}$-9
0.2	7.279399	1.018179 $_{10}$-9
0.1	7.481170	1.584527 $_{10}$-10

1.0	5.000000 $_{10}$-2	2.345625 $_{10}$-7
0.9	1.857218	1.234856 $_{10}$-7
0.8	3.214601	7.763417 $_{10}$-8
0.7	4.311327	4.849321 $_{10}$-8
0.6	5.207108	2.905035 $_{10}$-8
0.5	5.932694	1.618062 $_{10}$-8
0.4	6.506019	8.019135 $_{10}$-9
0.3	6.937327	3.258261 $_{10}$-9
0.2	7.229091	8.772098 $_{10}$-10
0.1	7.358895	1.336573 $_{10}$-11

We see how **ROOT7** begins by increasing the start value of y2 in fixed steps as long as the calculated error remains negative. As soon as a positive error is obtained (for y2 = 2.36$_{10}$-7), the range of the root is narrowed down by continued bisections. It should be noted that we never reach the point r = 0, but this is not strictly necessary in order to fix the start value of y2. This start value is proportional to the diffusion effectiveness factor which is the factor that tells us how large a fraction of the particle volume is utilized. We also notice, that in the first guess the integration could only be carried through to r = 0.2. The start value must be very near the correct value if the integration shall be successful. The reason for this is that the y1-function changes very quickly in the outermost layer and the y1-function then becomes poorly determined at the center of the sphere. We can also say that the original differential equations, which have been put up for a spherical geometry, are no longer valid, but should be reformulated for a geometry where nearly all of the reaction takes place in the surface layer and the inner parts of the particles are inactive, i.e. an infinite plane slab.

3.1.4. Discussion of program DE-1. If we must solve the same type of a differential equation many times for different values of the numerical constants in the equation, it is recommended to prepare a special program where the variable coefficients are input material. If, on the other hand, we shall only solve differential equations in very rare cases, we can make a small ad hoc program, e.g. as the program d-215 shown on page 27. The program must be corrected so that it contains the proper expressions for the derivatives and the proper start values. The program must then be translated, and the calculation is carried out.

It may sometimes be an advantage if we can avoid the trouble of correcting the program and translating it again. For this purpose we have made a special program, DE-1, for solution of ordinary differential equations with or without boundary conditions. The program has been translated only once, but as we in GIER ALGOL cannot read an ALGOL expression as input to a translated program and get it translated and evaluated, we have solved this problem in a special way. The expression which defines the derivative is read as a list of small integers, and each integer has a special significance, such as addition, subtraction, etc. We can also say that we have defined a system of orders (commands) which must be interpreted and calculated by the program.

3. 1. 4. 1. The procedure EXPRES1. This procedure interprets a list of these small integers and calculates the numerical value of the corresponding expression.

The declaration is:

```
real procedure EXPRES1(terms, j, OP, x, ar1, ar2, failure);
value terms, x;
integer terms, j, OP;
real x;
array ar1, ar2;
```

```
procedure failure;
begin
   integer k, type, p;
   real A, B;
   array stack[1:10];
   procedure DO1(expr);
   real expr;
   begin
      k := k + 1;
      if k = type then
      begin
         A := expr;
         go to EX
      end if k = type
   end DO1;
   procedure DO2(expr, not);
   real expr;
   boolean not;
   begin
      k := k + 1;
      if k = type then
      begin
         if not then failure
         else
         begin
            A := expr;
            go to EX
         end if good
      end if k = type
   end DO2;
   integer procedure next;
   begin
      j := j + 1;
      next := OP
   end next;
   p := 0;
   A := 0;
   for j := 1 step 1 until terms do
   begin
      type := OP;
      if type < 6 then
      begin
         A := stack[p-1];
         B := stack[p];
         k := 0;
         p := p - 1;
         DO1(A+B);
         DO1(A-B);
         DO1(A×B);
         DO2(A/B, B = 0);
         DO2(A↑B, A ≤ 0)
      end if type < 6
      else
      if type < 10 then
      begin
         p := p + 1;
         k := 5;
         DO1(1);
         DO1(x);
         DO1(ar1[next]);
         DO1(ar2[next])
      end if type < 10
      else
      begin
         k := 9;
         A := stack[p];
         DO1(-A);
         DO2(1/A, A = 0);
         DO2(sqrt(A), A < 0);
         DO1(sin(A));
         DO1(cos(A));
         DO1(arctan(A));
         DO2(ln(A), A ≤ 0);
         DO2(exp(A), A > 354)
      end type ≥ 10;
EX:   stack[p] := A
   end for j;
   EXPRES1 := A
end EXPRES1;
```

The formal parameters in EXPRES1 are:

integer terms: The number of operations in the operation list.

integer j: A counter which is used by the procedure for fetching the various operations. The procedure contains the for-statement:

```
for j := 1 step 1 until terms do
```

integer OP: This parameter must yield the value of the operation in the operation list corresponding to a given j-value. If the operation list has been declared as:

```
integer array OPLIST[1:terms];
```

we can use the actual parameter OPLIST[j] instead of OP. It is called by name.

real x: One of the possible operations (no. 7) is: Fetch x, where x is a simple variable of type real. In the program DE-1 x is the independent integration variable.

array ar1, ar2: Two of the other operations are:

No. 8: Fetch ar1[n]
No. 9: Fetch ar2[n]

The procedure automatically fetches the value of n as the next number in the operation list. Thus, two operation numbers are required for operations 8 and 9. In program DE-1 ar1 is a list of numerical constants and ar2 is the set of dependent variables: y[1:N].

procedure failure: If the actual operation cannot be carried out from purely arithmetic reasons (division by zero, etc.) the procedure failure will be carried out instead.

The 17 possible operations can be seen from the list:

Operation list to EXPRES1

No.	Operation	Condition for failure
1	A := A + B	
2	A := A - B	
3	A := A×B	
4	A := A/B	B = 0
5	A := A↑B	$A \leq 0$

6	A := 1	
7	A := x	
8	A := ar1[n]	
9	A := ar2[n]	
10	A := - A	
11	A := 1/A	A = 0
12	A := sqrt(A)	A < 0
13	A := sin(A)	
14	A := cos(A)	
15	A := arctan(A)	
16	A := ln(A)	A ≤ 0
17	A := exp(A)	A > 354

The significance of A and B can be seen from the following description of how the procedure works.

The procedure operates with a local array:

```
array stack[1:10];
```

and a corresponding counter:

```
integer p;
```

During the evaluation of the expression the numerical values and the constants will be placed in this stack. When a new number is put into the stack, p is increased by 1. Similarly, p is reduced by 1 when the two last numbers in the stack are operated upon to give a single number. The last actual element in the stack is always stack[p].

The procedure first puts p equal to zero and then carries out the for-statement:

```
for j := 1 step 1 until terms do
```

For each new value of j we fetch the next operation with the statement:

```
type := OP;
```

If the actual parameter corresponding to **OP** is **OPLIST**[j], the statement is then:

```
type := OPLIST[j];
```

What further happens depends upon the numerical value of type. The operations may be divided into three classes:

Class 1 comprises the operations no. 1 to no. 5. The two last elements in the stack are fetched:

```
A := stack[p-1];
B := stack[p];
```

We then reduce p by 1. A and B are local variables. After this, the operation itself is carried out. This is done by means of two local procedures: DO1 and DO2. The actual parameter here is the expression which must be calculated: A + B, A-B, etc. In the procedure DO2 we also make a further test for failure. Finally, the new A-value is stored as stack[p]. This applies to all three classes of operations.

Class 2 contains the operations no.6 to no. 9. These are the fetch-operations. We increase p by 1 and calculate A as 1, x, ar1[n], or ar2[n].

Class 3 contains the operations 10 to 17. The height of the stack is not changed here. We first fetch A:

```
A := stack[p];
```

The operation is then carried out on A.

An example of the use of EXPRES1 is shown below.

3.1.4.2. Input material. In the appendix are shown the input specifications, a typical calculation, and the ALGOL program to DE-1. There are twelve input data groups. As usual, groups 0 and 1 are the cover and headline data.

Group 2 contains the following calculation parameters:

N: The number of functions to be integrated simultaneously ($N \geq 1$).

P: The number of boundary condition points ($P \geq 0$).

C: The number of boundary conditions ($C \geq P$).

Z: The number of integration zones. The calculated function values will be printed at the end of each zone ($Z \geq 1$).

K: The number of constants in the expressions for the derivative ($K \geq 0$).

x1: The start value of the independent variable, x.

Group 3 is the list of the Z zone end point coordinates: xpoint[1:Z].

If the integration starts at x = 3 and ends at x = 7 and we wish to know the calculated function values in the intermediate points: 5, 6, and 6.5, we must specify **Z** = 4 and the zone end points:

```
xpoint[1] := 5;
xpoint[2] := 6;
xpoint[3] := 6.5;
xpoint[4] := 7;
```

Group 4 is the start values, ystart[1:N], of the N dependent variables for x = x1.

Group 5 is a list, delta[1:N], in which delta[i] is the required accuracy in the integration of function no. i.

Group 6 is the necessary numerical constants, **CONST**[1:K], which enter the expressions for the derivatives. It is not necessary to include the number 1.

Group 7 contains the list, **TERM**[1:N], in which **TERM**[i] is the number of operations in the expression for the derivative for function no. i. The operation numbers are specified in the next data group.

Group 8 contains the **TERM**[1] + **TERM**[2] + ... + **TERM**[N] operations. As an example we take the solution of the two simultaneous differential equations:

$$dy1/dx = -1_{10}8 \times y[2]/x\uparrow 2$$
$$dy2/dx = 1_{10}{-6} \times x\uparrow 2/sqrt(y[1])$$

There are two functions (N = 2) and two constants in the expressions (K = 2). The data groups 6, 7, and 8 are specified thus:

$-1_{10}8$		$1_{10}{-6}$			e					
9		11			e					
9	2	7	4	7	4	8	1	3		
9	1	12	11	7	3	7	3	8	2	3
e										

The two constants are $-1_{10}8$ and $1_{10}{-6}$. The two expressions contain 9 and 11 operations, respectively. The operations in the stack are:

Derivative	Operation	Significance	stack[1]	stack[2]
1	9	Fetch y[n]		
	2	n := 2	y[2]	
	7	Fetch x	y[2]	x
	4	Divide	y[2]/x	
	7	Fetch x	y[2]/x	x
	4	Divide	$y[2]/x\uparrow 2$	
	8	Fetch k[n]	$y[2]/x\uparrow 2$	
	1	n := 1	$y[2]/x\uparrow 2$	$-1_{10}8$
	3	Multiply	$-1_{10}8\times y[2]/x\uparrow 2$	
2	9	Fetch y[n]		
	1	n := 1	y[1]	
	12	Take sqrt	sqrt(y[1])	
	11	Take reciprocal	1/sqrt(y[1])	
	7	Fetch x	1/sqrt(y[1])	x
	3	Multiply	x/sqrt(y[1])	
	7	Fetch x	x/sqrt(y[1])	x
	3	Multiply	$x\uparrow 2$/sqrt(y[1])	
	8	Fetch k[n]	$x\uparrow 2$/sqrt(y[1])	
	2	n := 2	$x\uparrow 2$/sqrt(y[1])	$1_{10}-6$
	3	Multiply	$1_{10}-6\times x\uparrow 2$/sqrt(y[1])	

If there are no boundary conditions (C = 0), the data groups 9-12 and the corresponding stop-ees are skipped.

Group 9 contains the list, xbound[1:P], of the points at which the boundary conditions must be satisfied. The start value of x (x1) may be included here.

Group 10 contains the list, yinc[1:N], of the function increments at x = x1. The program satisfies the boundary conditions by varying the start values of the functions (the dependent variables). If the start value of a function must be varied, the increment for this function must be specified different from zero. If it is zero, the start value is not varied. The number of functions for which yinc is different from zero must be equal to C. If not, we get the error message:

Inconsistent boundary condition data

on the typewriter and the calculation is stopped.

Group 11 is a list, yerror[1:N], of the permissible errors in the functions at the start point. The adjustment of the start values is continued until the changes become less than specified in this group. We may specify zero or anything as the permissible errors for those functions for which the start values are not to be varied.

Group 12 contains the boundary condition coefficients which are written as an array: B[1:C, 1:P×N + 1] with a row for each of the C conditions.

Each row contains P×N + 1 elements. For P = 2 (two boundary condition points) and N = 3 (three functions) the boundary condition no. j has the form:

```
    B[j, 1]×y[1] + B[j, 2]×y[2] + B[j, 3]×y[3]
  + B[j, 4]×y[1] + B[j, 5]×y[2] + B[j, 6]×y[3] + B[j, 7]  =  0
```

Here, the y-values in the three first terms must be evaluated at the first boundary condition point (xbound[1]) and the three next terms must contain the y-values at the second boundary condition point (xbound[2]). In other words, the boundary conditions are written as linear combinations of the function values in the various boundary condition points.

In this data group the characters d and q may be used only within a single row.

3.1.4.3. Program description. The program is divided into the four blocks:

1. Cover page and headlines.
2. Data input.
3. Check and printing of data.
4. Integration.

If there are boundary conditions, block 3 will check that each boundary condition point (group 9) is contained among the zone end points (group 3) or is the start value of x. If not, we get the error message:

```
xbound no. i not listed among zone points
```

on the typewriter, and the calculation is stopped.

The essential part of the integration block is the Z calls of RUKU3.

The calculation of the derivatives is made with a local procedure, **FUNCTION**, which contains a call of the procedure **EXPRES1**.

The adjustment of the boundary conditions is made with the procedure **NOLEQ3**. As long as the conditions are not fulfilled, the program prints only the start values and the errors. When the conditions are satisfied, the integration is repeated with printing of the y-values. The number of cycles in **NOLEQ3** is restricted to 50.

If more than 500 calls of the procedure **FUNCTION** are made, we get the remark:

```
More than 500 calls of Runge-Kutta
```

in the output report, the integration is stopped, and the program proceeds to the next section.

<u>3.1.4.4. Typical result.</u> The typical output report shown contains six sections.

Section 1 is the same calculation as was made with the program d-215 on page 27. Similarly, section 2 corresponds to program d-220 on page 42.

In section 3 we show how the differential equation of second order:

d2y/dx2 = y

can be solved after transformation into the two simultaneous first-order equations:

dy1/dx = y2
dy2/dx = y1

Sections 4 and 5 illustrate the two difficult differential equations with boundary conditions which were calculated on program d-221 on page 45.

Finally, section 6 shows the calculation of a definite integral. If we must calculate the definite integral of f(x) from a to b, we may integrate:

dy/dx = f(x)

We put y = 0 at x = a and integrate from a to b.

<u>3.1.5. Other integration methods.</u> In the Runge-Kutta method we calculate the derivative five times in each integration step. When we proceed

to the next step we discard all the old information available with the exception of the new y-values. It is more economical to store a few of the older generations of the y-values and to use an integration method which utilizes this old information. There exists methods of this type (predictor-corrector methods), but they have the disadvantage that the old y-values must be available when the integration is started. Normally, these will not be known, and it becomes necessary to make the first few steps by the Runge-Kutta method. It should be possible to make a combined method starting with the Runge-Kutta method and then switching over to a predictor-corrector method. No publications have been made of such combined methods.

3.2. Partial Differential Equations.

Partial differential equations occur when we have functions of several variables. If we have a function of three variables:

$$y = F(x1, x2, x3)$$

we may calculate the partial derivatives with respect to each of the three variables. They are denoted by dF/dx1, dF/dx2, and dF/dx3. Strictly speaking, special curved d-letters must be used here, but these are not available on the Flexowriter, and we must use the normal form. The significance of the partial derivative:

$$dF/dx1$$

is, that we must differentiate F with respect to x1, the other independent variables (x2 and x3) being kept constant. Normally, dF/dx1 will then be a new function of x1, x2, and x3.

A partial differential equation is an equation containing one or more partial derivatives. On the basis of one or more of these equations we must then calculate backwards to the original function, y.

The situation is much more confused in the solution of partial differential equations than for ordinary differential equations. It is not posible to indicate a single method - as the Runge-Kutta method for ordinary differential equations - which can be used automatically by a computer in all practical cases. The difficulties become quite overwhelming if we put forward the reasonable claim that the solution found must be correct within a given tolerance. Much mathematical work is still to be done before these problems have been solved, if this is possible at all.

We can refer to different works on this subject: Gram (1962), Forsythe and Wasow (1960), Lapidus (1962), and Modern Computing Methods (1961).

The partial differential equations are normally divided into three main types, depending upon the signs of the terms when they are written in a standard way. The three types are called elliptic, parabolic, and hyperbolic. We may also classify them according to the boundary conditions, but as the equation type and the boundary type often go together, this gives about the same classification.

In this book we only consider parabolic partial differential equations with a boundary condition system as shown in figure 1.

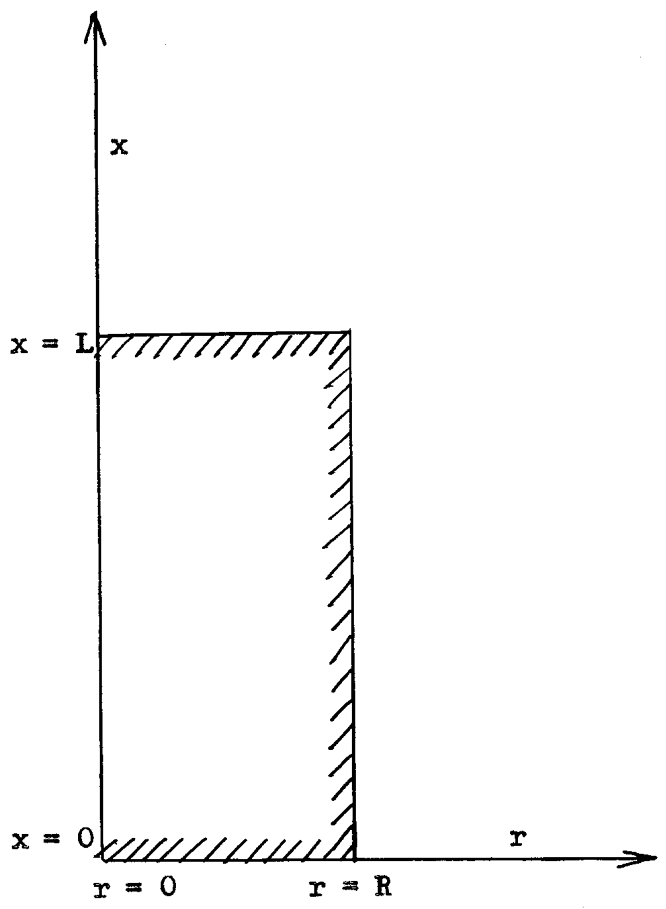


Figure 1.

Boundary conditions with parabolic differential equations.

The problem occurs in connection with the calculation of temperature and conversion distribution in cylindric, catalytic reactors. The functions used here have two independent variables, the radial coordinate, r, in the cylinder, and the axial coordinate, x. The boundary conditions normally occur at the end circles of the cylinder:

$$x = 0, \qquad 0 \le r \le R$$
$$x = L, \qquad 0 \le r \le R$$

and at its surface:

$$r = R, \qquad 0 \le x \le L$$

In the stepwise integration method we make one step at a time in the axial direction, and for each axial step we calculate the new radial profile of the function. We describe two procedures for this: an explicite method (PAPADEQ1), where the radial distribution is also calculated stepwise, and an implicite method (PAPADEQ2), where we put up a set of equations which after solution gives the complete new radial profile. For comparison we also discuss an analytic solution method, but this can only be used in very simple cases.

3.2.1. Differential equations from a catalytic converter. This problem is described in detail in Kjær(1958), chapter 6. A gas flows in axial direction through a cylindric, catalytic converter. The axial coordinate is x, and the radial coordinate, r. There are three functions of these two variables:

t:	The catalyst temperature.
t0:	The temperature of the gas flowing in the cooling tubes.
z:	The mole fraction of the key component.

There is also a function of the single variable, x:

tann: The temperature of the shell cooling gas.

The converter is of the well-known TVA-type in which the cold gas is preheated by passing along the cylinder surface (temperature: tann). It is then further preheated by passing through a special heat exchanger, which is not calculated here, and is given a final preheating by passing through a number of cooling tubes which are mounted coaxially in the catalyst layer. The temperature in the cooling tubes is t0, and the gas flows countercurrently to the gas in the catalyst layer.

After passing the cooling tubes the gas changes direction at x = 0 and then passes through the catalyst layer with the temperature, t.

We assume that the reaction rate is a function of only two variables, the catalyst temperature, t, and the mole fraction, z, of the key component. We cannot handle several simultaneous reactions. We take into account the diffusion of the key component, but not of the other components.

The catalyst temperature, t, satisfies the differential equation:

```
dt/dx = tau×RATE - lambda1×(t-t0)  +  beta×(2dt/dr2  +  1/r×dt/dr)
```

Here, dt/dx is the partial derivative of t with respect to x, dt/dr is the corresponding radial derivative, and d2t/dr2 is the second-order derivative of t with respect to r. The other variables are:

tau: The adiabatic temperature increase (deg. C).

RATE: The reaction rate expression. For ammonia synthesis this must be an expression giving df/dx, where f is the fractional conversion.

lambda1: The so-called cooling constant valid for heat transfer from the catalyst layer to the cooling tubes (/m).

beta: This factor has the units meter and is a measure of the radial heat diffusion rate. It is calculated as:

```
beta := K/(G×Cp)
```

where K is the effective thermal conductivity of the catalyst layer (kcal/mhC), G is the gas mass velocity in the catalyst layer (kg/sq.mh), and Cp is the specific heat of the gas (kcal/kgC).

For the temperature in the cooling tubes (t0) we have the differential equation:

```
dt0/dx = - lambda1(t-t0)
```

Here, dt0/dx is the partial derivative of t0 with respect to x. As the individual cooling tubes are connected only at the end of the cylinder, the radial gradient, dt0/dr, is not used.

The mole fraction, z, of the key component satisfies the following differential equation:

dz/dx = ZFUNC×(RATE + gamma×(d2z/dr2 + 1/r×dz/dr))

We have the first-order and the second-order derivative, dz/dr and d2z/dr2, of z with respect to r, and the first-order derivative, dz/dx, of z with respect to x.

As the reaction rate, RATE, refers to the fractional conversion, f, not to z, it is necessary to apply the correction function, ZFUNC, for the transformation from f to z. For ammonia synthesis we may have:

ZFUNC = 0.5×(1 + z)↑2

The factor gamma (units: meter) measures the radial diffusion rate of the key component. A formula for gamma may be found in Kjær (1958), page 81.

The differential equation for the shell cooling gas temperature, tann, is:

dtann/dx = lambda2×(t[NRAD] - tann)

As tann depends upon x only, dtann/dx is the normal derivative, not the partial derivative.

lambda2 is the cooling constant for heat transfer from the catalyst layer to the shell cooling gas (/m).

t[NRAD] is the catalyst temperature at the surface of the catalyst layer (r = R).

Besides the differential equations we must also have some expressions for the boundary conditions. We assume that we know the radial distribution of t, t0, and z at x = 0 and for all values of r in the range: $0 \leq r \leq R$.
Because of the symmetry around the cylinder axis we also have:

dt/dr = 0 for r = 0 and $0 \leq x \leq L$
dz/dr = 0 for r = 0 and $0 \leq x \leq L$

At the cylinder surface the amount of flowing heat must equal the amount of heat taken up by the shell cooling gas. We then have:

UB×(t[NRAD] - tann) = - K×(dt/dr)

Here, UB is the overall heat transfer coefficient (kcal/sq.mhC) for transfer from the catalyst layer to the shell cooling gas, and dt/dr is the radial gradient at the surface. K is explained above.

As the gas cannot diffuse through the cylinder wall, the boundary condition for z at this point becomes:

```
dz/dr = 0
```

3.2.2. Analytic solution. We now give an analytic solution to a strongly simplified version of the equations above. We consider only the catalyst temperature, t, and neglect the two other functions, t0 and z. The value of tann is kept constant. The solution is explained in chapter 5 in Kjær (1958).

We assume that the reaction rate decreases exponentially with the axial distance through the cylinder, but is independent of temperature:

```
RATE := R0×exp(-c×x);
```

R0 and c are constants. We normalize the radial coordinate, so that it goes from 0 to 1. It is called y:

```
y := r/R;
```

The differential equation may then be written:

```
dt/dx = tau×R0×exp(-c×x) + beta/R↑2×(d2t/dy2 + 1/y×dt/dy)
```

The boundary condition at the cylinder surface may be written:

```
t[NRAD] - tann = - m×(dt/dy)
```

with m given by:

```
m := K/(UB×R);
```

The value of tann is assumed constant, corresponding to an infinite velocity of the shell cooling gas. For x = 0 the catalyst temperature is constant equal to: tinlet.

The analytic solution may then be written:

```
t := tann + F1(x,y) + F2(x, y) + F3(x, y);
```

The three functions of x and y are given by:

```
F1 := (tinlet - tann)×SUM(2×J0(lam[n]×y)×exp(p[n]×x)/B[n]);

F2 := tau×R0/c×(J0(w×y)/(J0(w) - m×w×J1(w)) - 1)×exp(-c×x);

F3 := - tau×R0×R↑2/beta×SUM(2×J0(lam[n]×y)×exp(p[n]×x)/
        (B[n]×(lam[n]↑2 - w↑2)));
```

The procedure SUM denotes a summation from n = 1 and as far as required to give convergence. J0 and J1 denote the two Bessel functions of the first kind and order 0 and 1. The constant w is:

```
w := R×sqrt(c/beta);
```

The coefficients lam[n] are the roots in the equation:

```
J0(lam[n]) = m×lam[n]×J1(lam[n])
```

and the two other coefficients, p[n] and B[n], are found from lam[n]:

```
p[n] := beta/R↑2×lam[n]↑2;

B[n] := lam[n]×((lam[n]×m)↑2 + 1)×J1(lam[n]);
```

The calculations are made by the program d-224:

Program d-224. Analytic solution of temperature distribution.

```
begin
   boolean f1, f3;
   integer nrad, nax, q, n, r, cmax, rmax, imax, i;
   real x, J0, J1, w, tann, tinlet, tau, R0,  c,  m, R, beta, delx, DEN, y,
        F1, F2, F3, T1, T3, t;
   array lam, p, B[1:25], diffx[0:5];
   comment library BESS1;
   comment library ROOT4;
   real procedure BESSROOT;
   begin
      BESS1(x, J0, J1, 0, 0, 1, 1);
      BESSROOT := m×x×J1 - J0
   end BESSROOT;
```

```
input(tann, tinlet, tau, R0, c, m, R, beta, nrad, nax, delx);
w := R×sqrt(c/beta);
q := 0;
outcr;
outtext(⌈< n   lam[n]          p[n]            B[n]⌉);
outcr;
for n := 1 step 1 until 25 do
begin
   outcr;
   output(⌈-nd⌉, n);
   ROOT4(q, 4, 1, 0.05, 0, 1000, 1₁₀-6, false, diffx, x, BESSROOT);
   lam[n] := x;
   p[n] := - beta/R↑2×lam[n]↑2;
   BESS1(x, J0, J1, 0, 0, 1, 1);
   B[n] := lam[n]×((lam[n]×m)↑2 + 1)×J1;
   output(⌈-n.dddddd₁₀-dd⌉, outsp(2), lam[n], outsp(2), p[n], outsp(2), B[n]);
   diffx[0] := diffx[0] + 3
end for n;
outcr;
outcr;
outtext(⌈< y   ⌉);
begin
   array YFUNC[0:nrad];
   BESS1(w, J0, J1, 0, 0, 1, 1);
   DEN := J0 - m×w×J1;
   for r := 0 step 1 until nrad do
   begin
      y := r/nrad;
      output(⌈-nddd.dd⌉, y);
      BESS1(w×y, J0, J1, 0, 0, 1, 1);
      YFUNC[r] := tau×R0/c×(J0/DEN - 1)
   end for r;
   outcr;
   outtext(⌈< x   ⌉);
   outcr;
   outcr;
   output(⌈-nd.dd⌉, 0);
   for r := 0 step 1 until nrad do
   output(⌈-nddd.dd⌉, tinlet);
   cmax := 0;
   for i := 1 step 1 until nax do
```

```
      begin
         x := i×delx;
         outcr;
         output(⟨-nd.dd⟩, x);
         for r := 0 step 1 until nrad do
         begin
            y := r/nrad;
            F1 := 0;
            F2 := YFUNC[r]×exp(- c×x);
            F3 := 0;
            f1 := f3 := true;
            n := 0;
            for n := n + 1 while (f1 ∨ f3) ∧ n ≤ 25 do
            begin
               BESS1(y×lam[n], J0, 0, 0, 0, 1, 0);
               T1 := (tinlet-tann)×2×J0×exp(p[n]×x)/B[n];
               T3 := - tau×R0×R↑2/beta×2×J0×exp(p[n]×x)/(B[n]×(lam[n]↑2-w↑2));
               f1 := abs(T1) > 1₁₀-4;
               f3 := abs(T3) > 1₁₀-4;
               F1 := F1 + T1;
               F3 := F3 + T3
            end for n;
            if n > cmax then
            begin
               cmax := n;
               rmax := r;
               imax := i
            end if;
            t := tann + F1 + F2 + F3;
            output(⟨-nddd.dd⟩, t)
         end for r
      end for i
   end block;
   outcr;
   outcr;
   outtext(⟨<  cmax  rmax  imax⟩);
   outcr;
   output(⟨-nddd⟩, cmax, rmax, imax);
   outcr;
end program;
```

The program was tested with the input tape:

Ff

350 400 1 40 0.25 1.8 0.5 $7_{10}{-4}$ 5 2 1 e

corresponding to the following values of the variables:

tann	350
tinlet	400
tau	1
RO	40
c	0.25
m	1.8
R	0.5
beta	$7_{10}{-4}$
nrad	5
nax	2
delx	1

The result is:

n	lam[n]	p[n]	B[n]
1	$9.851463_{10}{-1}$	$-2.717437_{10}{-3}$	1.776819
2	3.973135	$-4.420024_{10}{-2}$	$-1.155288_{10}\,1$
3	7.094173	$-1.409164_{10}{-1}$	$2.726985_{10}\,1$
4	$1.022788_{10}\,1$	$-2.929065_{10}{-1}$	$-4.708770_{10}\,1$
5	$1.336530_{10}\,1$	$-5.001675_{10}{-1}$	$7.027090_{10}\,1$
6	$1.650431_{10}\,1$	$-7.626986_{10}{-1}$	$-9.638250_{10}\,1$
7	$1.964415_{10}\,1$	-1.080500	$1.251229_{10}\,2$
8	$2.278448_{10}\,1$	-1.453570	$-1.562677_{10}\,2$
9	$2.592511_{10}\,1$	-1.881911	$1.896481_{10}\,2$
10	$2.906595_{10}\,1$	-2.365522	$-2.251197_{10}\,2$
11	$3.220693_{10}\,1$	-2.904402	$2.625648_{10}\,2$
12	$3.534803_{10}\,1$	-3.498552	$-3.018859_{10}\,2$
13	$3.848920_{10}\,1$	-4.147972	$3.429962_{10}\,2$
14	$4.163044_{10}\,1$	-4.852662	$-3.858253_{10}\,2$
15	$4.477173_{10}\,1$	-5.612621	$4.302946_{10}\,2$
16	$4.791306_{10}\,1$	-6.427851	$-4.763655_{10}\,2$
17	$5.105442_{10}\,1$	-7.298350	$5.239633_{10}\,2$
18	$5.419580_{10}\,1$	-8.224119	$-5.730386_{10}\,2$

```
19   5.733722₁₀1  -9.205157          6.235793₁₀2
20   6.047864₁₀1  -1.024147₁₀1  -6.755205₁₀2
21   6.362009₁₀1  -1.133304₁₀1   7.288249₁₀2
22   6.676155₁₀1  -1.247989₁₀1  -7.834641₁₀2
23   6.990302₁₀1  -1.368201₁₀1   8.393818₁₀2
24   7.304450₁₀1  -1.493940₁₀1  -8.966161₁₀2
25   7.618599₁₀1  -1.625205₁₀1   9.550632₁₀2


y       0.00     0.20     0.40     0.60     0.80     1.00
x


0.00  400.00   400.00   400.00   400.00   400.00   400.00
1.00  435.39   435.39   435.39   435.39   435.39   432.94
2.00  462.96   462.96   462.96   462.95   462.86   458.56


cmax  rmax  imax
 19    0     1
```

The program first calculates 25 values of lam[n] by means of the procedure ROOT4. The Bessel functions are calculated by the procedure BESS1 which was made at the Risø GIER installation. A table is printed of the lam-values and the corresponding values of p[n] and B[n]. Finally, the temperature function is calculated in 6 radial points and 2 axial points. During the calculation of the t-values the program checks how big a value of n must be used in the series expansion with lam[n]. The maximum value of n is printed. It is 19 and occurs in the first axial point at the cylinder axis.

3.2.3. The procedure PAPADEQ1. When the rate expression is not as simple as exp(-c×x) which was used in the analytic solution, but also depends upon the temperature, it is normally not possible to find an analytic solution. We now show the procedure PAPADEQ1 which solves the problem by a stepwise numerical integration using the explicite method. The declaration is:

```
procedure PAPADEQ1(r, NRAD, i, NAX, tann, delr, x, xstart, delx, tcat, ttub,
          zcat, RATE, ZFUNC, beta, gamma, lambda1, tau, lambda2, lambda3, t,
          t0, z, CHECK);
value NRAD, delr, NAX, xstart, delx, beta, gamma, lambda1, tau, lambda2,
    lambda3;
integer r, NRAD, i, NAX;
real tann, delr, x, xstart, delx, tcat, ttub, zcat, RATE, ZFUNC, beta, gamma,
    lambda1, tau, lambda2, lambda3;
array t, t0, z;
procedure CHECK;
begin
   real delr2, rate, zfunc, tgrad, zgrad, del, deltann, alfa1, alfa2, r1, r2;
   array delt, delt0, delz[0:NRAD];
   delr2 := delr↑2;
   alfa1  := lambda3/(lambda3 + 2/delr);
   alfa2  := 1 - alfa1;
   for i  := 1 step 1 until NAX do
   begin
      x := xstart + (i-1)×delx;
      for r  := 0 step 1 until NRAD do
      begin
         tcat  := t[r];
         ttub  := t0[r];
         zcat  := z[r];
         rate  := RATE;
         zfunc  := ZFUNC;
         if r = 0 then
         begin
            tgrad  := 4×(t[1] - tcat)/delr2;
            zgrad  := 4×(z[1] - zcat)/delr2
         end if r = 0
         else
         if r ≠ NRAD then
         begin
            r1  := 1/r;
            r1  := 1 + r1;
            r2  := 1 + r1;
            tgrad  := (r1×t[r+1] - r2×tcat + t[r-1])/delr2;
            zgrad  := (r1×z[r+1] - r2×zcat + z[r-1])/delr2
         end normal point
         else
         zgrad  := (z[NRAD-1] - zcat)×(2+1/NRAD)/delr2;
         del  := - lambda1×(tcat-ttub);
```

```
            delt[r] := (tau×rate + del + beta×tgrad)×delx;
            delt0[r] := del×delx;
            delz[r] := zfunc×(rate + gamma×zgrad)×delx
        end for r;
        delt[NRAD] := 0;
        for r := 0 step 1 until NRAD do
        begin
            t[r] := t[r] + delt[r];
            t0[r] := t0[r] + delt0[r];
            z[r] := z[r] + delz[r]
        end for r;
        deltann := lambda2×(t[NRAD] - tann)×delx;
        t[NRAD] := alfa1×tann + alfa2×t[NRAD-1];
        tann := tann + deltann;
        x := xstart + i×delx;
        CHECK
    end for i
end PAPADEQ1;
```

The procedure makes steps in the axial direction and in the radial direction. The accuracy of the integration is not controlled. For the two directions we have the variables:

Direction	Number of steps	Running index	Step length
Axial	NAX	i	delx
Radial	NRAD	r	delr

Before the call of the procedure the start values at x = xstart must have been inserted in the three arrays:

```
array t, t0, z[0:NRAD];
```

After the calculation the arrays will contain the calculated arrays for the other end of the cylinder.

The procedure has the following parameters:

<u>integer</u> r: Running index for the radial points.

<u>integer</u> NRAD: The number of radial steps.

<u>integer</u> i: Running index for the axial points.

<u>integer</u> NAX: The number of axial steps.

<u>real</u> tann: The temperature of the shell cooling gas. This is integrated by the procedure.

real delr: The radial step length.

real x: The actual value of the axial coordinate.

real xstart: The start value of x.

real delx: The axial step length.

real tcat, ttub, zcat: The actual values of the catalyst temperature, the cooling tube temperature, and the mole fraction of the key component. They are assigned by the procedure for each value of r and i.

real RATE: A reaction rate expression called by name. For ammonia synthesis the expression must yield df/dx. It will normally be a function of tcat and zcat (and of x).

real ZFUNC: An expression called by name which enters the differential equation for the transformation from f to z.

real beta: The radial thermal diffusivity factor (m).

real gamma: The radial mass diffusivity factor (m).

real lambda1: The cooling constant for heat transfer from the catalyst layer to the cooling tubes (/m).

real tau: The adiabatic temperature increase (deg.C).

real lambda2: The cooling constant for heat transfer from the catalyst layer to the shell cooling gas (/m).

real lambda3: This is the factor 2×UB/K which enters the boundary condition at the cylinder surface. The unit is /m. The relation between lambda3 and the m-factor is:

```
lambda3 := 2/(m×R);
```

array t, t0, z[0:NRAD]: The radial profiles of the catalyst temperature, the cooling tube temperature, and the mole fraction of the key component. They are integrated by the procedure.

procedure CHECK: This is called by the procedure after each axial step. It may be used for printing or storing of the calculated radial profiles.

The procedure PAPADEQ1 works as follows. Outermost in the procedure we have the for-statement:

```
for i := 1 step 1 until NAX do
```

which counts the necessary axial steps. For each value of i we first calculate the actual x-value:

```
x := xstart + (i-1)×delx;
```

and then follows the inner for-statement:

```
for r := 0 step 1 until NRAD do
```

For each value of r we must now calculate the increment in the three functions: t, t0, and z over the axial step. The increments are calculated as the local array:

```
array delt, delt0, delz[0:NRAD];
```

The characteristic feature of the explicite calculation method used in PAPADEQ1 is that the increment, delt[r], is calculated on the basis of t[r-1], t[r], and t[r+1], i.e. the three nearest points in the previous radial profile. For r = 0 and r = NRAD special formulas must be used. For each value of r we fetch the actual values of the three profiles and store them as the simple variables:

```
tcat := t[r];
ttub := t0[r];
zcat := z[r];
```

We then calculate:

```
rate := RATE;
zfunc := ZFUNC;
```

RATE is normally an expression containing tcat and zcat (and sometimes x). ZFUNC is a function of zcat.

At the inner points (0<r<NRAD) we then calculate the two gradient expressions:

```
tgrad := d2t/dr2 + 1/r×dt/dr
zgrad := d2z/dr2 + 1/r×dz/dr
```

We calculate the first-order derivative:

```
dt/dr = (t[r+1] - t[r])/delr
```

and the second-order derivative:

```
d2t/dr2 = ((t[r+1] - t[r]) - (t[r] - t[r-1]))/delr↑2
```

Insertion gives:

```
tgrad := ((1+1/r)×t[r+1] - (2+1/r)×t[r] + t[r-1])/delr↑2;
```

The formula for zgrad is quite similar. At the cylinder axis we write:

```
tgrad := 4×(t[1] - t[0])/delr↑2;
```

and similarly for zgrad.

At the surface of the cylinder we have:

```
dz/dr = 0
```

which is written as:

```
zgrad := (z[NRAD-1] - z[NRAD])×(2 + 1/NRAD)/delr↑2;
```

We do not calculate tgrad for r = NRAD. The last point in the for-statement is the calculation of the increments in the three radial profiles:

```
delt[r] := (tau×rate-lambda1×(t[r]-t0[r]) + beta×tgrad)×delx;

delt0[r] := - lambda1×(t[r] - t0[r])×delx;

delz[r] := zfunc×(rate + gamma×zgrad)×delx;
```

When the for-statement is finished, we preliminarily put delt[NRAD] equal to zero and add the increments in a new for-statement with r going from 0 to NRAD:

```
t[r] := t[r] + delt[r];
t0[r] := t0[r] + delt0[r];
z[r] := z[r] + delz[r];
```

Finally, the increment in the shell cooling gas temperature is calculated as:

```
deltann := lambda2×(t[NRAD] - tann)×delx;
```

The new value of the catalyst temperature, t[NRAD], at the cylinder surface is calculated to:

```
t[NRAD] := alfa1×tann + alfa2×t[NRAD-1];
```

The two coefficients are:

```
alfa1 := lambda3/(lambda3 + 2/delr);
alfa2 := 1 - alfa1;
```

The significance of lambda3 has been explained above.

Finally, we add deltann to tann, calculate the new x-value corresponding to the end point of the step, and call the procedure CHECK. This is a formal parameter in the procedure PAPADEQ1 and is used for printing or storing of the radial profiles. We then proceed to the next axial step.

We have a special program, DE-2, which performs calculations with PAPADEQ1 and PAPADEQ2. It is discussed in section 3.2.5. A direct simple use of PAPADEQ1 is the following program:

```
Program d-211. Test of PAPADEQ1.
begin
   integer r, i;
   real x, tann, tcat, ttub, zcat;
   array t, t0, z[0:20];
   procedure CHECK;
   begin
      outcr;
      output(⟨-n.dd⟩, x);
      for r := 0 step 4 until 20 do
      output(⟨-nddd.dd⟩, t[r])
   end CHECK;
   comment library PAPADEQ1;
   x := 0;
   for r := 0 step 1 until 20 do
   begin
      t[r] := 400;
      t0[r] := 410 - 0.05×r↑2;
      z[r] := 0.03
   end for r;
   tann := 20;
   outcr;
   outtext({< x   });
   for r := 0 step 4 until 20 do
   output({nd}, outtext({<  t[}), r, outtext({<] }));
   outcr;
   CHECK;
   PAPADEQ1(r, 20, i, 4,  tann,  0.025,  x,  0,  0.05,  tcat,  ttub,  zcat,
   -1.2_{10}12×exp(-20135/(tcat+273.16))×(zcat+0.0025×tcat-1.5), 1, 7_{10}-4, 1_{10}-3, 0.24,
   900, 0.01, 2.22222, t, t0, z, CHECK)
end program;
```

The result is:

```
x     t[ 0]   t[ 4]   t[ 8]   t[12]   t[16]   t[20]

 0.00  400.00  400.00  400.00  400.00  400.00  400.00
 0.05  402.72  402.71  402.68  402.63  402.56  392.16
 0.10  405.67  405.65  405.59  405.49  405.35  394.22
 0.15  408.91  408.88  408.78  408.62  408.40  396.49
 0.20  412.51  412.46  412.32  412.09  411.77  399.02
```

This program simulates the calculation of an ammonia converter with a simplified kinetics. We can also test PAPADEQ1 with the differential equation which was solved analytically by the program d-224 on page 67. This is done by the program:

Program d-225. Test of PAPADEQ1 with analytic solution.

```
begin
   integer nrad, nax, i, r;
   real tann, tinlet, tau, R0,  c,  m,  R, beta, delx, x, tcat, ttub, zcat;
   comment library PAPADEQ1;
   input(tann, tinlet, tau,  R0,  c,  m,  R,  beta,  nrad,  nax,  delx);
   begin
      array t, t0, z[0:nrad];
      procedure CHECK;
      if (2×i):nax×nax = 2×i then
      begin
         outcr;
```

```
          output(⟨-nd.dd⟩, x);
          for r := 0 step nrad:5 until nrad do
          output(⟨-nddd.dd⟩, t[r])
       end if CHECK;
       outcr;
       outcr;
       outtext(⟨<  y   ⟩);
       for r := 0 step 1 until 5 do
       output(⟨-nddd.dd⟩, r/5);
       outcr;
       outtext(⟨<  x   ⟩);
       outcr;
       x := 0;
       for r := 0 step 1 until nrad do
       begin
          t[r] := tinlet;
          t0[r] := z[r] := 0
       end for r;
       i := 0;
       CHECK;
       PAPADEQ1(r, nrad, i, nax, tann, R/nrad, x, 0, delx, tcat, ttub, zcat,
                RO×exp(-c×x), 1, beta, 0, 0, tau, 0, 2/(R×m), t, t0, z, CHECK)
    end block;
    outcr;
end program;
```

The test was made with the same input material as for d-224, except for the changes:

nrad	20
nax	80
delx	0.025

i.e. 20 radial points and 80 axial points. The result is:

```
 y      0.00    0.20    0.40    0.60    0.80    1.00
 x

 0.00  400.00  400.00  400.00  400.00  400.00  400.00
 1.00  435.50  435.50  435.50  435.50  435.47  431.73
 2.00  463.15  463.15  463.15  463.15  462.94  457.09
```

The difference between the numerical solution and the analytic solution is largest at the cylinder surface (1.5 deg.) and smallest at the axis. For PAPADEQ2 we give a more detailed analysis of the influence of the step length on page 89.

3.2.4. The procedure PAPADEQ2. This procedure solves the same problem as PAPADEQ1, but is more advanced and accurate. An implicite method is used which is somewhat similar to that first described by Crank and Nicolson (1947). We have also utilized a procedure published by Gram (1962), page 83, but we have not tried to introduce an automatic fixation of the step length.

The declaration is:

```
procedure PAPADEQ2(r, NRAD, i, NAX, count, cmax, tann, delr, x, xstart, delx, tca
            ttub, zcat, RATE, ZFUNC, beta, gamma, lambda1, tau, lambda2, lambda3,
            eps, t, t0, z, CHECK);
value NRAD, delr, NAX, cmax, xstart, delx, beta, gamma, lambda1, tau, lambda2,
      lambda3, eps;
integer r, NRAD, i, NAX, count, cmax;
real tann, delr, x, xstart, delx, tcat, ttub, zcat, RATE, ZFUNC, beta, gamma,
     lambda1, tau, lambda2, lambda3, eps;
array t, t0, z;
procedure CHECK;
begin
   boolean zero;
   real tannold, tannnew, delr2, rate, zfunc, delexpr, A1, A2, B1, B2, C1, C2,
        D1, D2, delt0, tannmean, N1, N2, max, diff, Q1, Q2, Q3, Q4;
   array told, tnew, t0old, t0new, zold, znew, E1, E2[0:NRAD];
   for r := 0 step 1 until NRAD do
   begin
      told[r] := t[r];
      t0old[r] := t0[r];
      zold[r] := z[r]
   end for r;
   tannold := tann;
   delr2 := delr↑2;
   Q1 := 1/delr2;
   Q3 := Q1/2;
   Q4 := 1/delx;
   for i := 1 step 1 until NAX do
```

```
begin
   x := xstart + (i-1)×delx;
   count := 0;
   max := 1₁₀100;
   for count := count + 1 while max > eps ∧ count ≤ cmax do
   begin
      for r := 0 step 1 until NRAD do
      begin
         tcat := 0.5×(t[r] + told[r]);
         ttub := 0.5×(t0[r] + t0old[r]);
         zcat := 0.5×(z[r] + zold[r]);
         rate := RATE;
         zfunc := ZFUNC;
         zero := r = 0;
         Q2 := if zero then Q1 else Q3/r;
         A1 := beta×(Q1-Q2);
         A2 := zfunc×gamma×(Q1-Q2);
         C1 := beta×(Q1+Q2);
         C2 := zfunc×gamma×(Q1+Q2);
         delexpr := (if zero then 4 else 2)/delr2;
         B1 := - beta×delexpr - Q4;
         B2 := - zfunc×gamma×delexpr - Q4;
         if zero then
         begin
            C1 := beta×delexpr;
            C2 := zfunc×gamma×delexpr
         end if r = 0;
         delt0 := lambda1×(tcat-ttub);
         D1 := - tau×rate + delt0 - t[r]/delx;
         D2 := - zfunc×rate - z[r]/delx;
         t0new[r] := t0[r] - delt0×delx;
         if r = NRAD then
         begin
            tannmean := 0.5×(tann + tannold);
            tannnew := tann + lambda2×(tcat-tannmean)×delx;
            B1 := B1 - C1×lambda3×delr;
            A1 := A1 + C1;
            D1 := D1 - C1×lambda3×delr×tannnew;
            A2 := A2 + C2;
            C1 := C2 := 0
         end if r = NRAD;
```

```
            N1 := B1 + (if zero then 0 else A1×E1[r-1]);
            E1[r] := - C1/N1;
            tnew[r] := (D1 + (if zero then 0 else - A1×tnew[r-1]))/N1;
            N2 := B2 + (if zero then 0 else A2×E2[r-1]);
            E2[r] := - C2/N2;
            znew[r] := (D2 + (if zero then 0 else -A2×znew[r-1]))/N2
         end for r;
         for r := NRAD - 1 step -1 until 0 do
         begin
            tnew[r] := E1[r]×tnew[r+1] + tnew[r];
            znew[r] := E2[r]×znew[r+1] + znew[r]
         end for r;
         max := 0;
         for r := 0 step 1 until NRAD do
         begin
            diff := abs(tnew[r] - told[r]);
            if diff > max then max := diff;
            told[r] := tnew[r];
            t0old[r] := t0new[r];
            zold[r] := znew[r]
         end for r;
         tannold := tannnew
      end for count;
      for r := 0 step 1 until NRAD do
      begin
         told[r] := 2×told[r] - t[r];
         t[r] := tnew[r];
         t0old[r] := 2×t0old[r] - t0[r];
         t0[r] := t0new[r];
         zold[r] := 2×zold[r] - z[r];
         z[r] := znew[r]
      end for r;
      tannold := 2×tannold - tann;
      tann := tannnew;
      x := xstart + i×delx;
      CHECK
   end for i
end PAPADEQ2;
```

The parameters are the same as in PAPADEQ1, but three new ones have been added. Each axial step is carried out several times, until the temperature changes become constant. This iteration requires the parameters:

<u>integer</u> count: The procedure resets this variable to zero before each axial step and increases it by 1 for each iteration. When the procedure CHECK is called, count is 1 higher than the number of iterations carried out.

<u>integer</u> cmax: If count > cmax, no further iterations are made. If the CHECK procedure finds that count = cmax + 1, the iteration has not converged.

<u>real</u> eps: The iterations are stopped when the changes in all radial t-values become less than eps.

The procedure works as follows. In the explicite method in PAPADEQ1 we calculated the new value of t[r] from the values of t[r-1], t[r], and t[r+1] in the previous radial profile, but the implicite method in PAPADEQ2 calculates all new t[r]-values in the next radial profile ($0 \leq r \leq NRAD$) from all the old t[r]-values. The procedure puts up the necessary NRAD + 1 equations for the determination of the new t-values, and then solves these linear equations. The matrix describing this system of equations is particularly simple. It is tridiagonal, i.e. it contains the diagonal proper, and the two diagonals above and below. The other elements are zero:

$$\begin{matrix} B & C & 0 & 0 & 0 \\ A & B & C & 0 & 0 \\ 0 & A & B & C & 0 \\ 0 & 0 & A & B & C \\ 0 & 0 & 0 & A & B \end{matrix}$$

The values of A, B, and C are different in different rows and columns. Written out in full, the t-equations for NRAD = 4 are:

$$\begin{array}{lllll} B*t[0] + C*t[1] & & & & = D \\ A*t[0] + B*t[1] + C*t[2] & & & & = D \\ & A*t[1] + B*t[2] + C*t[3] & & & = D \\ & & A*t[2] + B*t[3] + C*t[4] & & = D \\ & & & A*t[3] + B*t[4] & = D \end{array}$$

We shall now find the values of A, B, C, and D for a given row with index r. We first consider an inner point: $0 < r < NRAD$. Here, A, B, and C are all different from zero. The original differential equation for t is:

```
dt/dx = tau×RATE - lambda1×(t-t0) + beta×(d2t/dr2 + 1/r×dt/dr)
```

We assume, that we know the radial profiles of t, t0, and z corresponding to the axial point, i, and must find the new values:

```
tnew, t0new, znew[0:NRAD];
```

at the next axial point, i + 1. The gradient is written as:

```
dt/dx = (tnew[r] - t[r])/delx
```

The radial first-order derivative is found from the central difference formula:

```
dt/dr = (tnew[r+1] - tnew[r-1])/(2×delr)
```

The second-order derivative is:

```
d2t/dr2 = ((tnew[r+1] - tnew[r]) - (tnew[r] - tnew[r-1]))/delr↑2
```

When these expressions are inserted into the differential equation, we get an equation of the form:

```
A×tnew[r-1] + B×tnew[r] + C×tnew[r+1] = D
```

The coefficients are:

```
A := beta×(1 - 1/(2×r))/delr↑2;
B := - 2×beta/delr↑2 - 1/delx;
C := beta×(1 + 1/(2×r))/delr↑2;
D := - tau×RATE + lambda1×(tav[r] - t0av[r]) - t[r]/delx
```

Here, tav[r] is the average value of t[r] and tnew[r]. The value of RATE is calculated at this temperature. As the exact value of tnew[r] is not known when the integration is started, each axial step is calculated more than once, and this gives the iteration feature mentioned above.

At the cylinder axis (r = 0) the coefficients are slightly different. We write the gradient expression:

```
d2t/dr2 + 1/r×dt/dr
```

as:

```
2×(d2t/dr2)
```

We express d2t/dr2 at r = 0 by means of the three temperatures: tnew[-1], tnew[0], and tnew[1]:

```
d2t/dr2 = ((tnew[1] - tnew[0]) - (tnew[0] - tnew[-1]))/delr↑2
```

Because of the axial symmetry we have that tnew[-1] = tnew[1] or:

```
d2t/dr2 = 2×(tnew[1] - tnew[0])/delr↑2
```

The coefficients then become:

```
A := 0;
B := - 4×beta/delr↑2 - 1/delx;
C := 4×beta/delr↑2;
D := As above;
```

At the surface (r = NRAD), we must find the three coefficients in the equation:

```
A×tnew[NRAD-1] + B×tnew[NRAD] = D
```

In order to introduce the boundary condition at the surface, we use the temperature, tnew[NRAD + 1], at a fictive point outside the cylinder, and we put up the two equations:

```
AP×tnew[NRAD-1] + BP×tnew[NRAD] + CP×tnew[NRAD+1] = DP
AQ×tnew[NRAD-1] + BQ×tnew[NRAD] + CQ×tnew[NRAD+1] = DQ
```

We must then eliminate tnew[NRAD + 1] from the two equations. The four coefficients AP, BP, CP, and DP are the same as in the normal case (0 < r < NRAD). Their values are given above.

The second equation expresses the boundary condition:

```
UB×(tnew[NRAD] - tannnew) = - K×dt/dr
```

which was explained on page 65. The shell cooling gas temperature in the new axial point is now called tannnew. The radial gradient is written as:

```
dt/dr = (tnew[NRAD+1] - tnew[NRAD-1])/(2×delr)
```

The coefficients are found by insertion and calculation:

```
AQ := 1;
BQ := - 2×UB×delr/K;
CQ := -1;
DQ := - 2×UB×delr/K×tannnew;
```

The final values of A, B, and D become:

```
A := AP + CP;
B := BP - CP×lambda3×delr;
D := DP - CP×lambda3×delr×tannnew;
```

We have inserted the formal parameter:

```
lambda3 := 2×UB/K;
```

The value of tannnew is found from:

```
tannnew := tann + lambda2×(tav[NRAD] - tannav)×delx;
```

Here, tav[NRAD] and tannav are the average values at the beginning and at the end of the step.

The formulas for z are very nearly the same as for t, except at the cylinder surface. The differential equation for z is:

```
dz/dx = ZFUNC×(RATE + gamma×(d2z/dr2 + 1/r×dz/dr))
```

In the normal case $(0 < r < nrad)$ we find:

```
A := ZFUNC×gamma×(1 - 1/(2×r))/delr↑2;
B := - 2×ZFUNC×gamma/delr↑2 - 1/delx;
C := ZFUNC×gamma×(1 + 1/(2×r))/delr↑2;
D := - ZFUNC×RATE - z[r]/delx;
```

At r = 0 we get the same changes as in the t-equations:

```
A := 0;
B := - 4×ZFUNC×gamma/delr↑2 - 1/delx;
C := 4×ZFUNC×gamma/delr↑2;
D := As above;
```

At the cylinder surface (r = NRAD) the radial z-gradient is zero. Here the equation may be written:

```
AP×znew[NRAD-1] + BP×znew[NRAD] + CP×znew[NRAD+1]  =  DP
```

The zero gradient is expressed by:

```
znew[NRAD-1] = znew[NRAD+1]
```

This gives the coefficients:

```
A := AP + CP;
B := BP;
C := 0;
D := DP;
```

Solution of the tridiagonal system of equations is made as indicated by Gram (1962), page 85. For 5 equations:

```
B  C  0  0  0    D
A  B  C  0  0    D
0  A  B  C  0    D
0  0  A  B  C    D
0  0  0  A  B    D
```

the solution can be found from a program of the form:

```
begin
   integer j;
   real DEN;
   array A, B, C, D, E, X[1:5];
   input(A, B, C, D);
   for j := 1 step 1 until 5 do
```

```
    begin
      DEN := if j = 1 then B[j] else B[j] + A[j]×E[j-1];
      E[j] := -C[j]/DEN;
      X[j] := (if j = 1 then D[j] else D[j] - A[j]×X[j-1])/DEN
    end for j;
    for j := 4 step -1 until 1 do X[j] := E[j]×X[j+1] + X[j]
end program;
```

The same method is used in PAPADEQ2, except that we do not store the complete arrays: A, B, C, and D. Only the two arrays E and X are stored as such, and each new value of A, B, C, and D is utilized immediately after it has been calculated. Two systems of equations are solved, one for t and the other for z. It should be noted, that we do not make any exchange of rows or of columns (pivoting). This should not be necessary because ill-conditioned equations are not likely to occur in practice.

The test program below is quite similar to the program d-211 on page 77, except that we now use PAPADEQ2:

Program d-212. Test of PAPADEQ2.

```
begin
   integer r, i, count;
   real x, tann, tcat, ttub, zcat;
   array t, t0, z[0:20];
   procedure CHECK;
   begin
      outcr;
      output(⟨-n.dd⟩, x);
      for r := 0 step 4 until 20 do
      output(⟨-nddd.dd⟩, t[r]);
      output(⟨-ndd⟩, count)
   end CHECK;
   comment library PAPADEQ2;
   x := 0;
   count := 0;
   for r := 0 step 1 until 20 do
   begin
      t[r] := 400;
      t0[r] := 410 - 0.05×r↑2;
      z[r] := 0.03
   end for r;
```

```
    tann := 20;
    outcr;
    outtext(≮< x  ≯);
    for r := 0 step 4 until 20 do
    output(≮nd≯, outtext(≮<  t[≯), r, outtext(≮<] ≯));
    outcr;
    CHECK;
    PAPADEQ2(r, 20, i, 4, count, 6, tann, 0.025, x, 0, 0.05, tcat, ttub, zcat,
    -1.2⏨12×exp(-20135/(tcat+273.16))×(zcat+0.0025×tcat-1.5), 1, 7⏨-4, 1⏨-3, 0.24,
    900, 0.01, 2.22222, 0.1, t, t0, z, CHECK)
end program;
```

The result is:

```
 x     t[ 0]   t[ 4]   t[ 8]   t[12]   t[16]   t[20]

 0.00  400.00  400.00  400.00  400.00  400.00  400.00   0
 0.05  402.84  402.83  402.80  402.75  402.68  401.44   4
 0.10  405.95  405.93  405.87  405.76  405.61  403.12   3
 0.15  409.40  409.37  409.27  409.09  408.85  405.07   3
 0.20  413.28  413.23  413.08  412.83  412.48  407.32   3
```

We have also tested PAPADEQ2 for comparison with the analytic solution found by the program on page 67. For this we used the program:

Program d-226. Test of PAPADEQ2 with analytic solution:

```
begin
   integer nrad, nax, i, r, count;
   real tann, tinlet, tau, R0, c, m, R, beta, delx, x, tcat, ttub, zcat;
   comment library PAPADEQ2;
   input(tann, tinlet, tau, R0, c, m, R, beta);
   go to BB;
AA: begin
      array t, t0, z[0:nrad];
      procedure CHECK;
      if (2×i):nax×nax = 2×i then
      begin
         outcr;
         output(≮-nd.dd≯, x);
         for r := 0 step nrad:5 until nrad do
```

```
            output(⦃-nddd.dd⦄, t[r])
        end if CHECK;
        outcr;
        outcr;
        outtext(⦃<  y   ⦄);
        for r := 0 step 1 until 5 do
        output(⦃-nddd.dd⦄, r/5);
        outcr;
        outtext(⦃<  x   ⦄);
        outcr;
        x := 0;
        for r := 0 step 1 until nrad do
        begin
           t[r] := tinlet;
           t0[r] := z[r] := 0
        end for r;
        i := 0;
        CHECK;
        PAPADEQ2(r, nrad, i, nax, count, 20, tann, R/nrad, x, 0, delx, tcat, ttub, z
                R0×exp(-c×x), 1, beta, 0, 0, tau, 0, 2/(R×m), 0.01, t, t0, z, CHECK)
     end block;
  BB: outcr;
     input(nrad, nax);
     delx := 2/nax;
     outcr;
     output(⦃-nddd⦄, outtext(⦃<nrad:⦄),  nrad,  outtext(⦃<   nax:⦄),   nax);
     outcr;
     go to AA
  end program;
```

We used the same input material as in the programs d-224 and d-225, but we varied the number of radial and axial steps. The result is:

```
nrad:   15   nax:   40


  y       0.00     0.20     0.40     0.60     0.80     1.00
  x


  0.00  400.00   400.00   400.00   400.00   400.00   400.00
  1.00  435.61   435.61   435.61   435.61   435.60   433.50
  2.00  463.35   463.35   463.35   463.35   463.23   459.26

nrad:   20   nax:   40


  y       0.00     0.20     0.40     0.60     0.80     1.00
  x


  0.00  400.00   400.00   400.00   400.00   400.00   400.00
  1.00  435.61   435.61   435.61   435.61   435.60   433.35
  2.00  463.35   463.35   463.35   463.35   463.24   459.11

nrad:   25   nax:   40


  y       0.00     0.20     0.40     0.60     0.80     1.00
  x


  0.00  400.00   400.00   400.00   400.00   400.00   400.00
  1.00  435.61   435.61   435.61   435.61   435.60   433.27
  2.00  463.35   463.35   463.35   463.35   463.24   459.04

nrad:   15   nax:   80


  y       0.00     0.20     0.40     0.60     0.80     1.00
  x


  0.00  400.00   400.00   400.00   400.00   400.00   400.00
  1.00  435.50   435.50   435.50   435.50   435.49   433.39
  2.00  463.15   463.15   463.15   463.15   463.04   459.07
```

```
nrad:   20   nax:   80


  y       0.00    0.20    0.40    0.60    0.80    1.00
  x


  0.00  400.00  400.00  400.00  400.00  400.00  400.00
  1.00  435.50  435.50  435.50  435.50  435.49  433.24
  2.00  463.15  463.15  463.15  463.15  463.04  458.92

nrad:   25   nax:   80

  y       0.00    0.20    0.40    0.60    0.80    1.00
  x


  0.00  400.00  400.00  400.00  400.00  400.00  400.00
  1.00  435.50  435.50  435.50  435.50  435.49  433.17
  2.00  463.15  463.15  463.15  463.15  463.04  458.86

nrad:   15   nax:  160


  y       0.00    0.20    0.40    0.60    0.80    1.00
  x


  0.00  400.00  400.00  400.00  400.00  400.00  400.00
  1.00  435.45  435.45  435.45  435.45  435.43  433.34
  2.00  463.05  463.05  463.05  463.05  462.94  458.97

nrad:   20   nax:  160


  y       0.00    0.20    0.40    0.60    0.80    1.00
  x


  0.00  400.00  400.00  400.00  400.00  400.00  400.00
  1.00  435.45  435.45  435.45  435.45  435.44  433.18
  2.00  463.05  463.05  463.05  463.05  462.94  458.83
```

nrad: 25 nax: 160

y / x	0.00	0.20	0.40	0.60	0.80	1.00
0.00	400.00	400.00	400.00	400.00	400.00	400.00
1.00	435.45	435.45	435.45	435.45	435.44	433.11
2.00	463.05	463.05	463.05	463.05	462.95	458.76

A comparison of the points at x = 2 and y = 1 shows the following differences between the analytic solution (page 71) and the solution with PAPADEQ2:

nrad: / nax:	15	20	25
40	0.70	0.55	0.48
80	0.51	0.36	0.30
160	0.31	0.27	0.20

If we express the error as a linear function of the reciprocal value of the step numbers:

```
error := A + B/nrad + C/nax;
```

we can calculate the coefficients A, B, and C on the program PA-7. The result is:

```
A := -0.1861;
B :=  6.755;
C := 16.610;
```

The error function is not quite linear, but it is clear that we get nearly the correct answer by extrapolation to zero step length.

3.2.5. Discussion of program DE-2. In a similar way as the program DE-1 can solve a set of ordinary differential equations in which the derivative expressions are specified as a list of the necessary operations, the program DE-2 performs the solution of partial differential equations which can be solved by the procedures PAPADEQ1 and PAPADEQ2. The expressions for RATE and ZFUNC are specified as an operation list. The input specifications to DE-2 are shown in the appendix, together with a typical output report and the ALGOL program.

There are seven data groups in the input material. Groups 0 and 1 are as usual cover page and headline data. Group 2 contains the calculation parameters:

NRAD: The number of radial steps.

NAX: The number of axial steps.

nradp: The number of radial points where the profiles must be printed.

naxp: The number of axial points where the profiles must be printed.

Height and diameter of the cylinder.

Three logical variables indicating whether the initial profiles of t, t0, and z are constant or not.

The number of operations in the RATE expression.

The number of operations in the ZFUNC expression.

K: The number of numerical constants in the expressions for RATE and ZFUNC.

A parameter which selects PAPADEQ1 or PAPADEQ2.

A parameter indicating whether the start profiles of t, t0, and z must be taken as the end profiles from the previous section or not. Using this trick we can change the expressions for RATE and ZFUNC when a certain x-value has been reached, or we may change the step length in axial direction. The radial step length cannot be changed.

The last parameter is cmax in PAPADEQ2.

Data group 3 contains the values of tau, beta, gamma, lambda1, lambda2, lambda3, and eps.

Data group 4 contains the constant start values of t, t0, tann, and z. Note, that we specify z as mole per cent, whereas the rate expression assumes that z is the mole fraction.

Data group 5 is the list of the K numerical constants.

Data group 6 is the list of operations in RATE and ZFUNC. Most of these have the same significance as in program DE-1. Operation 9 now means: fetch func[n], where func[1] is the catalyst temperature and func[2] is the mole fraction, z.

In data group 7 we specify the start profiles of t, t0, or z, if these have been declared not to be constant in group 2.

The program is divided into the four blocks:

1. Cover and headline.
2. Data input.
3. Printing of data.
4. Integration.

The typical output report contains four sections. The two first sections compare PAPADEQ1 and PAPADEQ2 and are identical to the calculations on the programs d-211 (page 77) and d-212 (page 88). Sections 3 and 4 illustrate the feature of transferring a profile from one section to the next.

4. REFERENCES

Bennett, A.A., Milne, W. E., and Bateman, H: Numerical Integration of Differential Equations, New York (1956).

Crank, J. and Nicolson, P.: A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. Proc. Cambr. Phil. Soc., 43, 50-67 (1947).

Forsythe, G.E. and Wasow, W.R.: Finite-Difference Methods for Partial Differential Equations, New York (1960).

Gram, Chr. (Editor): Selected Numerical Methods, Regnecentralen (1962).

Gram, Chr.: Definite integral by Rombergs method. BIT, 4, 54-60 (1964).

Kjær, J.: Measurement and Calculation of Temperature and Conversion in Fixed-Bed Catalytic Reactors, København (1958).

Lapidus, L.: Digital Computation for Chemical Engineers, New York (1962).

Merson, R.H.: Note in Lance, G.N.: Numerical Methods for High Speed Computers, London (1960), pag. 56.

Modern Computing Methods, H.M.S.O. , London (1961).

Naur, P.: Eigenvalue and table of differential equation, Regnecentralen (1961).

5. APPENDIX

This appendix contains information about the two programs, DE-1 and DE-2.

5.1. Input Specifications.

5.1.1. Input specifications to DE-1.

0. Cover Data:

```
Ff
.............  Calculation No.
[............  File No.
[......................................................  Cover Page Text
[......................................................    -    -    -
[......................................................    -    -    -
e              Stop e
```

1. Headline Data:

```
[......................................................  Section Headline Text
[......................................................     -        -      -
[......................................................     -        -      -
e       Stop e
```

2. Calculation Parameters:

```
 (0) .............  Number of functions (N)
 (1) .............  Number of boundary points (P)
 (2) .............  Number of boundary conditions (C)
 (3) .............  Number of integration zones (Z)
 (4) .............  Number of constants (K)
 (5) .............  Start value of x (x1)
e                   Stop e
```

3. List of Integration Zone Points, xpoint[1:Z]:

```
 (0) .............  xpoint[1]
 (1) .............  xpoint[2]
 (2) .............  xpoint[3]
          .
          .
          .
          .
          .
e                   Stop e
```

4. Start Values of Functions, ystart[1:N]:

```
(0) ............. ystart[1]
(1) ............. ystart[2]
(2) ............. ystart[3]
        .
        .
        .
        .
        .
e                 Stop e
```

5. Permissible Integration Errors, delta[1:N]:

```
(0) ............. delta[1]
(1) ............. delta[2]
(2) ............. delta[3]
        .
        .
        .
        .
        .
e                 Stop e
```

6. Constants [1:K]:

```
(0) ............. K[1]
(1) ............. K[2]
(2) ............. K[3]
        .
        .
        .
        .
        .
e                 Stop e
```

7. List of Special Operation Terms, TERM[1:N]:

(0) .... TERM[1]

(1) .... TERM[2]

(2) .... TERM[3]

.

.

.

.

.

e Stop e

8. List of Special Operations:

....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....

e Stop e

1:	+	7:	Fetch x	13:	Take sin
2:	-	8:	Fetch k[n]	14:	Take cos
3:	×	9:	Fetch y[n]	15:	Take arctan
4:	/	10:	Take minus	16:	Take ln
5:	↑	11:	Take recipr.	17:	Take exp
6:	Fetch 1	12:	Take sqrt		

In operations 8 and 9 specify n as the next term.

Data groups 9-12 and the stop ees are skipped, if there are no boundary conditions (C = 0).

9. List of Boundary Points, xbound[1:P]:

```
(0) .............  xbound[1]
(1) .............  xbound[2]
(2) .............  xbound[3]
          .
          .
          .
          .
          .
e                  Stop e
```

10. List of Function Increments at Start Point, yinc[1:N]:

```
(0) .............  yinc[1]
(1) .............  yinc[2]
(2) .............  yinc[3]
          .
          .
          .
          .
          .
e                  Stop e
```

11. List of Permissible Function Errors at Start Point, yerror[1:N]:

```
(0) .............  yerror[1]
(1) .............  yerror[2]
(2) .............  yerror[3]
          .
          .
          .
          .
          .
e                  Stop e
```

12. Boundary Condition Coefficients, BOUNDCOEF[1:C, 1:P×N + 1]:

oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo
oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo
oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo
oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo
oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo
oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo
oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo
oooooooo	oooooooo	oooooooo	oooooooo	oooooooo	oooooooo

e	Stop e
z	Final Stop

5.1.2. Input specifications to DE-2.

0. Cover Data:

Ff

....	Calculation No.
[.............	File No.
[...	Cover Page Text
[...	- - -
[...	- - -
e	Stop e

1. Headline Data:

[...	Section Headline Text
[...	- - -
[...	- - -
e	Stop e

2. Calculation Parameters:

(0)	....	Number of radial steps (NRAD)
(1)	....	Number of axial steps (NAX)
(2)	....	Number of radial printing points
(3)	....	Number of axial printing points
(4)	....	Cylinder height (m)
(5)	....	Cylinder diameter (m)
(6)	....	Non-uniform t-inlet, 0: no, 1: yes
(7)	....	Non-uniform t0-inlet, 0: no, 1: yes
(8)	....	Non-uniform z-inlet, 0: no, 1: yes
(9)	....	Number of RATE terms
(10)	....	Number of ZFUNC terms
(11)	....	Number of constants (K)
(12)	....	Integration method, 1: explicite, 2: implicite
(13)	....	Transfer from last section, 0: no, 1: yes
(14)	....	Maximum number of iterations
e		Stop e

3. Heat Transfer and Diffusion Coefficients:

(0)	Adiabatic temperature increase, tau (deg.C)
(1)	Thermal diffusivity, beta (m)
(2)	Mass diffusivity, gamma (m)
(3)	Bed cooling constant, lambda1 (/m)
(4)	Wall cooling constant, lambda2 (/m)
(5)	Wall ratio, lambda3 = 2 U/K (/m)
(6)	Permissible error in integration (deg.C)
e	Stop e

4. Inlet Temperatures and Composition:

(0)	Catalyst temperature (deg.C)
(1)	Cooling tube temperature (deg.C)
(2)	Annulus temperature (deg.C)
(3)	Mole per cent of key component
e	Stop e

5. List of Constants [1:K]:

(0)	CONST[1]
(1)	CONST[2]
(2)	CONST[3]
(3)	CONST[4]
.	
.	
.	
e	Stop e

6. List of Operations in RATE and ZFUNC:

....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....

e	Stop e

1:	+	7:	Fetch x	13:	Take sin
2:	-	8:	Fetch CONST[n]	14:	Take cos
3:	×	9:	Fetch func[n]	15:	Take arctan
4:	/	10:	Take minus	16:	Take ln
5:	↑	11:	Take recipr.	17:	Take exp
6:	Fetch 1	12:	Take sqrt		

In the operations 8 and 9 specify n as the next term. The significance of func is:

func[1]	Catalyst temperature
func[2]	Mole fraction of key component

7. Non-Uniform Profiles:

If non-uniformity of t, t0, and z at converter inlet has been specified in data group 2, the non-uniform arrays[0:NRAD] are specified here, each ending with a stop e.

....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....
....	....	....	....	....

e	Stop e
z	Final stop

5.2. Typical Calculations

5.2.1. Typical calculation from **DE-1**.

File No. 358

Calculation Example

GIER Calculation No. 9579

Solution of Ordinary Simultaneous Differential Equations

GIER Program DE-1

SECTION 1

Two Equations:
dy1/dx = y1 - 3.75*y2
dy2/dx = y1 - 3*y2

PERMISSIBLE INTEGRATION ERRORS

	1	2
1	$1.00000_{10}-6$	$1.00000_{10}-6$

CONSTANTS

	1	2
1	-3.75000	-3.00000

SPECIAL OPERATION TERMS

	1	2
1	8.00000	8.00000

LIST OF SPECIAL OPERATIONS

```
Derivative  operation

    1        9: Fetch y[n]  n :=   1
             8: Fetch k[n]  n :=   1
             9: Fetch y[n]  n :=   2
             3: Multiply
             1: Add

    2        9: Fetch y[n]  n :=   1
             8: Fetch k[n]  n :=   2
             9: Fetch y[n]  n :=   2
             3: Multiply
             1: Add
```

INTEGRATION RESULTS

x	y1	y2
0.00000	1.000000	1.000000
$1.00000_{10}-1$	$7.475564_{10}-1$	$8.154474_{10}-1$
$2.00000_{10}-1$	$5.357946_{10}-1$	$6.588089_{10}-1$
$3.00000_{10}-1$	$3.587788_{10}-1$	$5.260885_{10}-1$
$4.00000_{10}-1$	$2.114132_{10}-1$	$4.138524_{10}-1$
$5.00000_{10}-1$	$8.932431_{10}-2$	$3.191498_{10}-1$
$6.00000_{10}-1$	$-1.124048_{10}-2$	$2.394458_{10}-1$
$7.00000_{10}-1$	$-9.349961_{10}-2$	$1.725630_{10}-1$
$8.00000_{10}-1$	$-1.602125_{10}-1$	$1.166317_{10}-1$
$9.00000_{10}-1$	$-2.137441_{10}-1$	$7.004668_{10}-2$
1.00000	$-2.561199_{10}-1$	$3.143026_{10}-2$

SECTION 2

Same Problem with Boundary Conditions:
y1 = 0.21 at x = 1
y2 = 0.28 at x = 1

BOUNDARY POINTS

	1
1	1.00000

PERMISSIBLE FUNCTION ERRORS AT START POINT

	1	2
1	$1.00000_{10}-4$	$1.00000_{10}-4$

BOUNDARY CONDITION COEFFICIENT MATRIX

	1	2	3
1	1.00000	0.00000	$2.10000_{10}-1$
2	0.00000	1.00000	$2.80000_{10}-1$

INTEGRATION RESULTS

```
Boundary condition errors:


ystart:      1.000000             1.000000
error:      -4.661199 10-1       -2.485697 10-1

ystart:      1.100000             1.000000
error:      -3.479568 10-1       -2.102297 10-1

ystart:      1.000000             1.100000
error:      -6.098950 10-1       -2.837667 10-1


ystart:      1.400000             1.237961
error:      -3.355961 10-1       -1.789648 10-1

ystart:      1.500000             1.237961
error:      -2.174330 10-1       -1.406247 10-1

ystart:      1.400000             1.337961
error:      -4.793712 10-1       -2.141618 10-1


ystart:      1.800000             1.475922
error:      -2.050723 10-1       -1.093598 10-1

ystart:      1.900000             1.475922
error:      -8.690921 10-2       -7.101975 10-2

ystart:      1.800000             1.575922
error:      -3.488474 10-1       -1.445568 10-1


ystart:      2.200000             1.713883
error:      -7.454854 10-2       -3.975480 10-2

ystart:      2.300000             1.713883
error:       4.361455 10-2       -1.414776 10-3

ystart:      2.200000             1.813883
error:      -2.183236 10-1       -7.495180 10-2


ystart:      2.428459             1.849794
error:       2.421439 10-8        3.725290 10-9

ystart:      2.528459             1.849794
error:       1.181631 10-1        3.834003 10-2

ystart:      2.428459             1.949794
error:      -1.437751 10-1       -3.519700 10-2

ystart:      2.428459             1.849794
```

x	y1	y2
0.00000	2.428459	1.849794
$1.00000_{10}-1$	2.011841	1.560791
$2.00000_{10}-1$	1.657076	1.313573
$3.00000_{10}-1$	1.355362	1.102244
$4.00000_{10}-1$	1.099131	$9.217344_{10}-1$
$5.00000_{10}-1$	$8.818809_{10}-1$	$7.676841_{10}-1$
$6.00000_{10}-1$	$6.980203_{10}-1$	$6.363433_{10}-1$
$7.00000_{10}-1$	$5.427461_{10}-1$	$5.244875_{10}-1$
$8.00000_{10}-1$	$4.119313_{10}-1$	$4.293448_{10}-1$
$9.00000_{10}-1$	$3.020308_{10}-1$	$3.485318_{10}-1$
1.00000	$2.100000_{10}-1$	$2.800000_{10}-1$
error:	$2.421439_{10}-8$	$3.725290_{10}-9$

SECTION 3

Second-Order Equation:
d2y/dx2 = y, y = 1 and dy/dx = 0 at x = 0
Transformation into:
y = y1, dy/dx = y2

PERMISSIBLE INTEGRATION ERRORS

	1	2
1	$1.00000_{10}-5$	$1.00000_{10}-5$

SPECIAL OPERATION TERMS

	1	2
1	2.00000	2.00000

LIST OF SPECIAL OPERATIONS

Derivative	operation
1	9: Fetch y[n] n := 2
2	9: Fetch y[n] n := 1

INTEGRATION RESULTS

x	y1	y2
0.00000	1.000000	0.000000
$2.00000_{10}{-1}$	1.020067	$2.013356_{10}{-1}$
$4.00000_{10}{-1}$	1.081072	$4.107514_{10}{-1}$
$6.00000_{10}{-1}$	1.185464	$6.366520_{10}{-1}$
$8.00000_{10}{-1}$	1.337433	$8.881036_{10}{-1}$
1.00000	1.543078	1.175198

SECTION 4

```
dy1/dx = -k1×y[2]/x↑2
dy2/dx =  k2×x↑2/sqrt(y[1])
```

PERMISSIBLE INTEGRATION ERRORS

	1	2
1	$1.00000_{10}{-4}$	$1.00000_{10}{-8}$

CONSTANTS

	1	2
1	$-1.00000_{10}8$	$1.00000_{10}{-6}$

SPECIAL OPERATION TERMS

	1	2
1	9.00000	$1.10000_{10}1$

LIST OF SPECIAL OPERATIONS

```
Derivative  operation

   1         9: Fetch y[n]  n :=    2
             7: Fetch x
             4: Divide
             7: Fetch x
             4: Divide
             8: Fetch k[n]  n :=    1
             3: Multiply

   2         9: Fetch y[n]  n :=    1
            12: Take sqrt
            11: Take reciprocal
             7: Fetch x
             3: Multiply
             7: Fetch x
             3: Multiply
             8: Fetch k[n]  n :=    2
             3: Multiply
```

INTEGRATION RESULTS

```
     x                  y1                y2

1.00000            5.000000 10-2     2.347500 10-7
9.00000 10-1       1.859604          1.237213 10-7
8.00000 10-1       3.220519          7.790484 10-8
7.00000 10-1       4.322374          4.879528 10-8
6.00000 10-1       5.225717          2.938155 10-8
5.00000 10-1       5.962827          1.653888 10-8
4.00000 10-1       6.554742          8.402105 10-9
3.00000 10-1       7.018990          3.662951 10-9
2.00000 10-1       7.379975          1.299595 10-9
1.00000 10-1       7.725022          4.479921 10-10
1.00000 10-3       4.039834 10 1     3.294474 10-10
```

SECTION 5

INTEGRATION RESULTS

```
     x                  y1                y2

1.00000            5.000000 10-2     2.345000 10-7
9.00000 10-1       1.856423          1.234071 10-7
8.00000 10-1       3.212629          7.754391 10-8
7.00000 10-1       4.307644          4.839246 10-8
6.00000 10-1       5.200903          2.893986 10-8
5.00000 10-1       5.922643          1.606107 10-8
4.00000 10-1       6.489763          7.891287 10-9
3.00000 10-1       6.910072          3.123097 10-9
2.00000 10-1       7.178720          7.360371 10-10
1.00000 10-1       7.236559         -1.320212 10-10
More than 500 calls of Runge-Kutta
```

SECTION 6

Example of Calculation of Finite Integral
Integral of sqrt(x) from x = 0.5 to x = 1

PERMISSIBLE INTEGRATION ERRORS

```
          1

1    1.00000 10-6
```

```
                          SPECIAL OPERATION TERMS

           1

     1   2.00000

                          LIST OF SPECIAL OPERATIONS

Derivative  operation

     1          7:  Fetch x
               12:  Take sqrt

                          INTEGRATION RESULTS

      x              y1

5.00000 10-1    0.000000
6.00000 10-1    7.413638 10-2
7.00000 10-1    1.547390 10-1
8.00000 10-1    2.413255 10-1
9.00000 10-1    3.335077 10-1
1.00000         4.309643 10-1
```

5.2.2. Typical calculation from DE-2.

File No. 358

TVA-Converter with Simplified Kinetics

RATE := -1.2$_{10}$12×exp(-20135/(tcat + 273.16))×(z + 0.0025×tcat - 1.5)

GIER Calculation No. 10074

Solution of Partial Differential Equations in Cylindric Catalytic Reactors

GIER Program DE-2

SECTION 1

Explicite Solution

CALCULATION PARAMETERS

Number of radial steps	20
Number of axial steps	4
Number of radial printing points	20
Number of axial printing points	4
Cylinder height (m)	$200.0_{10}{-3}$
Cylinder diameter (m)	1.000
Non-uniform t0-inlet, 0: no, 1: yes	1
Number of RATE terms	24
Number of ZFUNC terms	1
Number of constants	5
Integration method, 1: explicite, 2: implicite	1
Maximum number of iterations	6

HEAT TRANSFER AND DIFFUSION COEFFICIENTS

Adiabatic temperature increase, tau (deg.C)	900.00
Thermal diffusivity, beta (m)	$700.00_{10}{-6}$
Mass diffusivity, gamma (m)	$1.0000_{10}{-3}$
Bed cooling constant, lambda1 (/m)	$240.00_{10}{-3}$
Wall cooling constant, lambda2 (/m)	$10.000_{10}{-3}$
Wall ratio, lambda3 = 2U/K (/m)	2.2222
Permissible error in integration (deg.C)	0.1000

INLET TEMPERATURES AND COMPOSITION

Catalyst temperature (deg.C)	400.00
Cooling tube temperature (deg.C)	400.00
Annulus temperature (deg.C)	20.00
Mole per cent of key component	3.0000

LIST OF CONSTANTS

Constant no. 1	$-1.2000_{10}12$
Constant no. 2	$2.5000_{10}{-3}$
Constant no. 3	-1.5000
Constant no. 4	$-20.135_{10}3$
Constant no. 5	273.16

LIST OF OPERATIONS

```
Function    Operation

RATE        9: Fetch FUNCT[n]   n :=   2
            9: Fetch FUNCT[n]   n :=   1
            8: Fetch CONST[n]   n :=   2
            3: Multiply
            1: Add
            8: Fetch CONST[n]   n :=   3
            1: Add
            8: Fetch CONST[n]   n :=   1
            3: Multiply
            8: Fetch CONST[n]   n :=   4
            8: Fetch CONST[n]   n :=   5
```

```
                 9:  Fetch FUNCT[n]  n :=    1
                 1:  Add
                 4:  Divide
                17:  Take exp
                 3:  Multiply

ZFUNC            6:  Fetch 1
```

INTEGRATION RESULTS

```
        Radial direction -->
x(m)

 0.00 t:  400.00  400.00  400.00  400.00  400.00  400.00  400.00  400.00
          400.00  400.00  400.00  400.00  400.00  400.00  400.00  400.00
          400.00  400.00  400.00  400.00  400.00
      t0: 410.00  409.95  409.80  409.55  409.20  408.75  408.20  407.55
          406.80  405.95  405.00  403.95  402.80  401.55  400.20  398.75
          397.20  395.55  393.80  391.95  390.00
       z:   3.00    3.00    3.00    3.00    3.00    3.00    3.00    3.00
            3.00    3.00    3.00    3.00    3.00    3.00    3.00    3.00
            3.00    3.00    3.00    3.00    3.00
   tann:   20.00
 0.05 t:  402.72  402.71  402.71  402.71  402.71  402.70  402.69  402.69
          402.68  402.67  402.66  402.64  402.63  402.61  402.60  402.58
          402.56  402.54  402.52  402.50  392.16
      t0: 410.12  410.07  409.92  409.66  409.31  408.85  408.30  407.64
          406.88  406.02  405.06  404.00  402.83  401.57  400.20  398.73
          397.17  395.50  393.73  391.85  389.88
       z:   3.29    3.29    3.29    3.29    3.29    3.29    3.29    3.29
            3.29    3.29    3.29    3.29    3.29    3.29    3.29    3.29
            3.29    3.29    3.29    3.29    3.29
   tann:   20.19
 0.10 t:  405.67  405.67  405.67  405.66  405.65  405.64  405.63  405.61
          405.59  405.57  405.54  405.52  405.49  405.46  405.42  405.39
          405.35  405.31  405.26  404.61  394.22
      t0: 410.21  410.16  410.00  409.75  409.39  408.93  408.37  407.70
          406.93  406.06  405.09  404.01  402.84  401.56  400.17  398.69
          397.10  395.41  393.62  391.73  389.85
       z:   3.61    3.61    3.61    3.61    3.61    3.61    3.61    3.61
            3.61    3.61    3.61    3.61    3.61    3.61    3.61    3.61
            3.61    3.60    3.60    3.60    3.50
   tann:   20.38
 0.15 t:  408.91  408.91  408.91  408.90  408.88  408.86  408.84  408.82
          408.78  408.75  408.71  408.67  408.62  408.57  408.52  408.46
          408.40  408.33  408.22  406.94  396.49
      t0: 410.26  410.21  410.06  409.80  409.43  408.97  408.40  407.73
          406.95  406.07  405.08  404.00  402.80  401.51  400.11  398.61
          397.00  395.29  393.48  391.57  389.80
       z:   3.96    3.96    3.96    3.96    3.96    3.96    3.96    3.96
            3.96    3.96    3.96    3.96    3.96    3.96    3.96    3.96
            3.96    3.95    3.95    3.94    3.74
   tann:   20.56
 0.20 t:  412.51  412.50  412.49  412.48  412.46  412.43  412.40  412.37
          412.32  412.27  412.22  412.16  412.09  412.02  411.94  411.86
          411.77  411.67  411.46  409.53  399.02
      t0: 410.28  410.23  410.07  409.81  409.44  408.97  408.39  407.71
          406.93  406.04  405.04  403.94  402.73  401.42  400.01  398.49
          396.87  395.14  393.30  391.39  389.72
       z:   4.36    4.36    4.36    4.36    4.36    4.36    4.36    4.36
            4.36    4.35    4.35    4.35    4.35    4.35    4.35    4.35
```

```
           4.34    4.34    4.34    4.29    4.02
  tann:   20.75

Radial mean values:

t:  410.54     t0:  399.32      z:    4.31
```

SECTION 2

Implicite Solution

CALCULATION PARAMETERS

```
Integration method, 1: explicite, 2: implicite            2
```

INTEGRATION RESULTS

```
        Radial direction -->
x(m)

 0.00 t:  400.00  400.00  400.00  400.00  400.00  400.00  400.00  400.00
          400.00  400.00  400.00  400.00  400.00  400.00  400.00  400.00
          400.00  400.00  400.00  400.00  400.00
     t0:  410.00  409.95  409.80  409.55  409.20  408.75  408.20  407.55
          406.80  405.95  405.00  403.95  402.80  401.55  400.20  398.75
          397.20  395.55  393.80  391.95  390.00
      z:    3.00    3.00    3.00    3.00    3.00    3.00    3.00    3.00
            3.00    3.00    3.00    3.00    3.00    3.00    3.00    3.00
            3.00    3.00    3.00    3.00    3.00
   tann:   20.00
  count:   0
 0.05 t:  402.84  402.84  402.84  402.83  402.83  402.82  402.81  402.81
          402.80  402.79  402.77  402.76  402.75  402.73  402.71  402.70
          402.68  402.65  402.63  402.55  401.44
     t0:  410.10  410.05  409.90  409.65  409.29  408.84  408.28  407.62
          406.87  406.01  405.04  403.98  402.82  401.55  400.19  398.72
          397.15  395.48  393.71  391.84  389.87
      z:    3.30    3.30    3.30    3.30    3.30    3.30    3.30    3.30
            3.30    3.30    3.30    3.30    3.30    3.30    3.30    3.30
            3.30    3.30    3.30    3.30    3.30
   tann:   20.19
  count:   3
 0.10 t:  405.95  405.95  405.95  405.94  405.93  405.92  405.90  405.89
          405.87  405.84  405.82  405.79  405.76  405.72  405.69  405.65
          405.61  405.56  405.50  405.28  403.12
     t0:  410.17  410.12  409.97  409.71  409.35  408.89  408.33  407.66
          406.90  406.03  405.05  403.98  402.80  401.52  400.14  398.65
          397.07  395.38  393.58  391.69  389.72
      z:    3.64    3.64    3.64    3.64    3.64    3.64    3.64    3.64
            3.64    3.64    3.64    3.64    3.64    3.64    3.64    3.64
            3.64    3.64    3.64    3.63    3.61
   tann:   20.38
  count:   2
 0.15 t:  409.40  409.40  409.40  409.38  409.37  409.35  409.33  409.30
          409.27  409.23  409.19  409.14  409.09  409.04  408.98  408.92
          408.85  408.78  408.67  408.25  405.07
     t0:  410.20  410.15  410.00  409.74  409.37  408.91  408.34  407.66
          406.89  406.01  405.02  403.94  402.74  401.45  400.05  398.55
          396.94  395.23  393.42  391.51  389.55
      z:    4.02    4.02    4.02    4.02    4.02    4.02    4.02    4.02
```

```
          4.02     4.02     4.02     4.02     4.02     4.02     4.01     4.01
          4.01     4.01     4.01     4.00     3.95
   tann:  20.57
  count:  2
0.20 t:  413.28   413.28   413.27   413.25   413.23   413.20   413.17   413.13
         413.08   413.03   412.97   412.90   412.83   412.75   412.67   412.57
         412.48   412.37   412.20   411.51   407.32
    t0:  410.19   410.14   409.98   409.72   409.35   408.88   408.30   407.62
         406.84   405.95   404.95   403.85   402.65   401.34   399.92   398.40
         396.78   395.05   393.22   391.29   389.35
     z:    4.45     4.45     4.45     4.45     4.45     4.45     4.45     4.45
           4.45     4.45     4.45     4.44     4.44     4.44     4.44     4.44
           4.43     4.43     4.42     4.40     4.32
   tann:  20.77
  count:  2


Radial mean values:

t:  412.11       t0:  399.20       z:    4.43


Maximum number of iterations:    3
```

SECTION 3

Implicite Solution with Transfer

CALCULATION PARAMETERS

```
Number of axial steps                                            24
Number of axial printing points                                   6
Cylinder height (m)                                           600.0   10^-3
Maximum number of iterations                                      8
```

INTEGRATION RESULTS

```
        Radial direction -->
x(m)

0.00 t:  400.00   400.00   400.00   400.00   400.00   400.00   400.00   400.00
         400.00   400.00   400.00   400.00   400.00   400.00   400.00   400.00
         400.00   400.00   400.00   400.00   400.00
    t0:  410.00   409.95   409.80   409.55   409.20   408.75   408.20   407.55
         406.80   405.95   405.00   403.95   402.80   401.55   400.20   398.75
         397.20   395.55   393.80   391.95   390.00
     z:    3.00     3.00     3.00     3.00     3.00     3.00     3.00     3.00
           3.00     3.00     3.00     3.00     3.00     3.00     3.00     3.00
           3.00     3.00     3.00     3.00     3.00
   tann:  20.00
  count:  0
0.10 t:  405.95   405.94   405.94   405.93   405.92   405.91   405.90   405.88
         405.86   405.84   405.81   405.78   405.75   405.72   405.68   405.64
         405.60   405.56   405.50   405.30   403.06
    t0:  410.17   410.12   409.97   409.71   409.35   408.89   408.33   407.67
         406.90   406.03   405.05   403.98   402.80   401.52   400.14   398.65
         397.07   395.38   393.59   391.69   389.72
     z:    3.64     3.64     3.64     3.64     3.64     3.64     3.64     3.64
           3.64     3.64     3.64     3.64     3.64     3.64     3.64     3.64
           3.64     3.64     3.64     3.63     3.61
   tann:  20.38
  count:   1
0.20 t:  413.27  413.26  413.25  413.24  413.22  413.19  413.15  413.11
         413.07  413.01  412.95  412.89  412.81  412.74  412.65  412.56
         412.46  412.36  412.20  411.53  407.21
    t0:  410.19  410.14  409.98  409.72  409.35  408.88  408.31  407.62
         406.84  405.95  404.95  403.85  402.65  401.34  399.92  398.40
         396.78  395.05  393.22  391.29  389.35
     z:    4.45    4.45    4.45    4.45    4.45    4.45    4.45    4.45
           4.45    4.45    4.44    4.44    4.44    4.44    4.44    4.43
           4.43    4.43    4.42    4.40    4.32
  tann:   20.77
 count:    2
0.30 t:  422.80  422.79  422.78  422.75  422.71  422.65  422.59  422.51
         422.43  422.33  422.22  422.10  421.97  421.82  421.67  421.50
         421.32  421.11  420.75  419.30  412.79
    t0:  410.01  409.95  409.79  409.53  409.15  408.67  408.08  407.38
         406.58  405.67  404.65  403.53  402.30  400.96  399.51  397.96
         396.30  394.53  392.66  390.71  388.85
     z:    5.53    5.53    5.53    5.53    5.53    5.53    5.52    5.52
           5.52    5.51    5.51    5.50    5.50    5.49    5.48    5.48
           5.47    5.46    5.44    5.37    5.17
  tann:   21.15
 count:    2
0.40 t:  436.33  436.32  436.29  436.23  436.16  436.06  435.95  435.81
         435.65  435.48  435.28  435.06  434.82  434.56  434.28  433.98
         433.66  433.26  432.47  429.68  420.48
    t0:  409.54  409.49  409.32  409.05  408.67  408.18  407.58  406.86
         406.04  405.12  404.08  402.93  401.67  400.30  398.83  397.24
         395.55  393.75  391.84  389.89  388.18
     z:    7.09    7.09    7.08    7.08    7.08    7.07    7.06    7.06
           7.05    7.03    7.02    7.01    6.99    6.98    6.96    6.94
           6.92    6.90    6.84    6.66    6.28
  tann:   21.55
 count:    2
0.50 t:  458.53  458.51  458.45  458.34  458.19  458.00  457.76  457.49
         457.17  456.81  456.41  455.98  455.50  454.99  454.44  453.85
         453.20  452.36  450.57  445.10  431.78
    t0:  408.65  408.59  408.43  408.15  407.76  407.26  406.65  405.92
         405.09  404.14  403.09  401.92  400.64  399.25  397.75  396.13
         394.41  392.57  390.65  388.75  387.27
     z:    9.65    9.65    9.65    9.64    9.63    9.61    9.59    9.57
           9.54    9.51    9.48    9.45    9.41    9.37    9.32    9.27
           9.22    9.15    8.99    8.55    7.82
  tann:   21.95
 count:    2
0.60 t:  501.73  501.68  501.55  501.34  501.03  500.64  500.16  499.60
         498.94  498.20  497.37  496.44  495.43  494.34  493.15  491.87
         490.43  488.42  484.00  472.10  450.49
    t0:  406.97  406.91  406.74  406.46  406.07  405.56  404.94  404.21
         403.37  402.42  401.35  400.17  398.87  397.47  395.95  394.32
         392.58  390.73  388.84  387.09  385.98
     z:   14.64   14.64   14.62   14.60   14.57   14.54   14.49   14.44
          14.37   14.30   14.22   14.14   14.04   13.94   13.82   13.70
          13.56   13.35   12.90   11.79   10.27
  tann:   22.37
 count:    3
```

```
Radial mean values:

t:  487.71      t0:  395.35      z:   13.34

Maximum number of iterations:    3
```

SECTION 4

CALCULATION PARAMETERS

```
Number of axial steps                                   16
Number of axial printing points                          4
Cylinder height (m)                                   400.0   10^-3
Transfer from last section, 0: no, 1: yes                1
```

INTEGRATION RESULTS

```
        Radial direction -->
x(m)

 0.60 t:  501.73  501.68  501.55  501.34  501.03  500.64  500.16  499.60
          498.94  498.20  497.37  496.44  495.43  494.34  493.15  491.87
          490.43  488.42  484.00  472.10  450.49
     t0:  406.97  406.91  406.74  406.46  406.07  405.56  404.94  404.21
          403.37  402.42  401.35  400.17  398.87  397.47  395.95  394.32
          392.58  390.73  388.84  387.09  385.98
      z:   14.64   14.64   14.62   14.60   14.57   14.54   14.49   14.44
           14.37   14.30   14.22   14.14   14.04   13.94   13.82   13.70
           13.56   13.35   12.90   11.79   10.27
   tann:   22.37
  count:    0
 0.70 t:  527.48  527.48  527.46  527.44  527.41  527.36  527.31  527.25
          527.17  527.08  526.97  526.84  526.69  526.52  526.32  526.09
          525.81  525.35  523.88  515.55  486.15
     t0:  404.24  404.18  404.01  403.72  403.32  402.81  402.18  401.44
          400.59  399.62  398.54  397.34  396.04  394.61  393.08  391.43
          389.67  387.82  385.97  384.50  384.03
      z:   17.81   17.81   17.81   17.81   17.81   17.81   17.81   17.82
           17.82   17.82   17.83   17.83   17.83   17.83   17.83   17.82
           17.81   17.77   17.63   16.94   14.75
   tann:   22.82
  count:    8
 0.80 t:  527.00  527.00  526.99  526.98  526.96  526.93  526.90  526.87
          526.82  526.78  526.72  526.67  526.60  526.53  526.46  526.38
          526.31  526.24  526.07  525.03  518.88
     t0:  401.24  401.19  401.01  400.72  400.31  399.79  399.15  398.39
          397.51  396.52  395.42  394.20  392.86  391.40  389.83  388.15
          386.35  384.46  382.58  381.14  381.07
      z:   18.09   18.09   18.09   18.09   18.09   18.10   18.11   18.11
           18.12   18.13   18.15   18.16   18.17   18.19   18.20   18.22
           18.23   18.25   18.28   18.45   18.85
   tann:   23.30
  count:    5
 0.90 t:  526.05  526.05  526.04  526.02  526.00  525.97  525.93  525.89
          525.84  525.79  525.73  525.66  525.59  525.51  525.42  525.34
          525.25  525.16  524.91  523.90  520.35
     t0:  398.20  398.14  397.96  397.66  397.25  396.71  396.05  395.28
          394.39  393.37  392.24  390.99  389.62  388.13  386.53  384.80
          382.96  381.02  379.10  377.66  377.69
     z:  18.32   18.32   18.32   18.32   18.33   18.34   18.34   18.35
         18.36   18.38   18.39   18.40   18.42   18.44   18.46   18.47
         18.49   18.51   18.57   18.81   19.45
  tann:  23.80
 count:  3
1.00 t: 525.07  525.07  525.06  525.04  525.01  524.98  524.94  524.89
        524.84  524.78  524.71  524.63  524.55  524.46  524.36  524.26
        524.16  524.03  523.68  522.53  519.11
    t0: 395.11  395.05  394.86  394.56  394.13  393.58  392.91  392.12
        391.20  390.17  389.01  387.73  386.33  384.81  383.16  381.40
        379.52  377.54  375.58  374.12  374.24
     z:  18.55   18.55   18.56   18.56   18.57   18.57   18.58   18.59
         18.61   18.62   18.63   18.65   18.67   18.69   18.71   18.73
         18.76   18.79   18.87   19.15   19.76
  tann:  24.29
 count:  1


Radial mean values:

t:  523.75     t0:  382.76      z:    18.84


Maximum number of iterations:     8
```

5.3. Programs in ALGOL

5.3.1. ALGOL program DE-1.

Program DE-1.

```
begin
   boolean boundcond, bnew, opnew;
   integer lRest, calcno, sectno, pageno, dtop, N, P, C, Z, K, i, j, opterm;
   real x1;
   comment library LINE;
   comment library PSHIFT;
   sectno := 1;
AA: begin comment cover and headline;
      integer i;
      array B[0:39];
      comment library CENTEXT;
      comment library COVER1;
      comment library HEADLINE;
      if sectno > 1 then go to BB;
      COVER1(12, 31,
      {<Solution of Ordinary Simultaneous Differential Equations},
      {<GIER Program DE-1});
      for i := 0 step 1 until 39 do B[i] := 0;
      dtop := drumplace;
      for i := to drum(B) while drumplace > 2679 do;
BB:   HEADLINE(20, ZZ)
   end of cover and headline;
   begin comment input block;
      boolean more;
      integer t;
      array DATA[0:5];
      comment library READ5;
      comment library READ6;
      procedure FIND E;
      for t := inchar while char ≠ 53 do;
      procedure READMAT(new, line, size, place);
      value line, size, place;
      boolean new;
      integer line, size, place;
      begin
         array COLD, CNEW[1:size];
         more := true;
comment
```

```
;
            new := false;
            for i := 1 step 1 until line do
            begin
               drumplace := place - (i-1)×size;
               from drum(COLD);
               drumplace := drumplace + size;
               from drum(CNEW);
               if more then
               for j := 1 step 1 until size do
               READ5(CNEW[j], j, L1);
L2:            for j := 1 step 1 until size do
               if CNEW[j] ≠ COLD[j] then new := true;
               drumplace := drumplace + size;
               to drum(CNEW);
               go to L3;
L1:            more := false;
               go to L2;
L3:         end for i;
            if more then FIND E
         end READMAT;
         READ6(67, 72);
         drumplace := 2685;
         from drum(DATA);
         N := DATA[0];
         P := DATA[1];
         C := DATA[2];
         Z := DATA[3];
         K := DATA[4];
         x1 := DATA[5];
         boundcond := C ≠ 0;
         begin comment special operations;
            array TERM[1:N];
            drumplace := 72×40 - 1 + N;
            from drum(TERM);
            opterm := 0;
            for i := 1 step 1 until N do
            opterm := opterm + TERM[i];
comment
```

```
;
          if opterm > 0 then
          begin
             READMAT(opnew, 1, opterm, dtop - 1000)
          end
          else
          begin
             opnew := false;
             FIND E
          end
       end special operations;
       if boundcond then
       begin
          READ6(73, 75);
          READMAT(bnew, C, P×N + 1, dtop)
       end if boundcond
    end input;
    begin comment data check and printing;
       boolean procedure NEW(chan);
       value chan;
       integer chan;
       begin
          integer new;
          array E[1:1];
          drumplace := 40×chan + 39;
          from drum(E);
          new := E[1];
          NEW := new > 0
       end NEW;
       procedure PRMAT(new, line, size, place, space, text);
       value new, line, size, place, space;
       boolean new;
       integer line, size, place, space;
       string text;
       begin
          array COEF[1:size];
          if new then
          begin
             PSHIFT(12);
             outsp(space);
comment
```

```
;
              outtext(text);
              LINE(2);
              outsp(9);
              for j := 1 step 1 until size do
              begin
                 output({nd}, j);
                 if j:5×5 = j ∧ j < size then
                 begin
                    LINE(1);
                    outsp(9)
                 end
                 else
                 outsp(12)
              end for j;
              LINE(1);
              for i := 1 step 1 until line do
              begin
                 LINE(1);
                 output({-nd}, i);
                 drumplace := place - (i-1)×size;
                 from drum(COEF);
                 for j := 1 step 1 until size do
                 begin
                    output({-n.ddddd₁₀-dd}, outsp(2), COEF[j]);
                    if j:5×5 = j ∧ j < size then
                    begin
                       LINE(1);
                       outsp(3)
                    end if
                 end for j
              end for i;
              LINE(3)
           end if new
        end PRMAT;
        comment library EXPRES2;




     comment
```

```
;

        PRMAT(NEW(70), 1, N, 70×40-1 + N, 25,
        ‹<PERMISSIBLE INTEGRATION ERRORS>›);
        PRMAT(NEW(71), 1, K, 71×40 - 1 + K, 35,
        ‹<CONSTANTS>›);
        PRMAT(NEW(72), 1, N, 72×40-1 + N, 28,
        ‹<SPECIAL OPERATION TERMS>›);
        if opnew then
        begin
           integer opcount;
           integer array TERM[1:N], SPECOP[1:opterm];
           procedure START;
           if j = 1 then output(‹-nddd›, i, outsp(4)) else outsp(9);
           drumplace := 72×40 - 1 + N;
           from drum(TERM);
           drumplace := dtop - 1000;
           from drum(SPECOP);
           PSHIFT(12);
           outsp(27);
           outtext(‹<LIST OF SPECIAL OPERATIONS>›);
           LINE(2);
           outtext(‹<Derivative  operation>›);
           LINE(2);
           opcount := 0;
           for i := 1 step 1 until N do
           begin
              EXPRES2(TERM[i], j, SPECOP[opcount + j], ‹<x>›, ‹<k>›, ‹<y>›, START);
              LINE(1);
              opcount := opcount + TERM[i]
           end for i;
           LINE(2)
        end if opnew;
```

comment

```
;
      if boundcond then
      begin
         real xb;
         array xpoint[1:Z], xbound[1:P];
         drumplace := 68×40 - 1 + Z;
         from drum(xpoint);
         drumplace := 73×40 - 1 + P;
         from drum(xbound);
         for i := 1 step 1 until P do
         begin
            xb := xbound[i];
            if xb = x1 then go to EX;
            for j := 1 step 1 until Z do
            if xb = xpoint[j] then go to EX;
            writecr;
            writetext({<xbound no.});
            write({-nd}, i);
            writetext({< not listed among zone points});
            go to ZZ;
EX:      end for i;
         PRMAT(NEW(73), 1, P, 73×40 - 1 + P, 32, {<BOUNDARY POINTS});
         PRMAT(NEW(75), 1, N, 75×40 - 1 + N,
         19, {<PERMISSIBLE FUNCTION ERRORS AT START POINT});
         PRMAT(bnew, C, P×N + 1, dtop, 21,
         {<BOUNDARY CONDITION COEFFICIENT MATRIX})
      end if boundcond
   end data check and printing;
   begin comment integration;
      boolean first, final, outside;
      integer var, k, count, term, dercount, opcount, zo;
      real x, xe, R;
      array delta, y, ystart, yinc, TERM[1:N], xbound[1:P],
            xpoint[1:Z], corstart, delcor, cor, epscor, ERROR, EO[1:C], CONST[1:K],
            Eold[1:C, 1:C], SPECOP[1:opterm];
      real procedure FUNCTION;
      begin
         procedure failure;
         begin
            outside := true;
comment
```

```
;
          go to EX
       end failure;
       R := 0;
       if var = 1 then
       begin
          opcount := 0;
          dercount := dercount + 1;
          if dercount > 500 then
          begin
             LINE(1);
             outtext(⟨⟨More than 500 calls of Runge-Kutta⟩);
             LINE(2);
             go to G1
          end if
       end if var = 1;
       term := TERM[var];
       if term > 0 ∧ ¬, outside then
       R := R + EXPRES1(term, j, SPECOP[opcount+j], x, CONST, y, failure);
EX:    FUNCTION := R;
       opcount := opcount + term
    end FUNCTION;
    procedure PR;
    if final then
    begin
       output(⟨-n.dddddₗ₀-dd⟩, x);
       for j := 1 step 1 until N do
       begin
          output(⟨-n.ddddddₗ₀-dd⟩, outsp(2), y[j]);
          if j:4×4 = j ∧ j < N then
          begin
             LINE(1);
             outsp(12)
          end
       end for j;
       LINE(1)
    end PR;
    procedure CHECKERROR;
comment
```

```
;
       if boundcond then
       begin
          for i := 1 step 1 until P do
          if x = xbound[i] then
          begin
             array BCOEF[1:N];
             for j := 1 step 1 until C do
             begin
                drumplace := dtop - (j-1)×(P×N+1) - (P-1)×N - 1;
                from drum(BCOEF);
                R := 0;
                for k := 1 step 1 until N do
                R := R + BCOEF[k]×y[k];
                ERROR[j] := ERROR[j] + R
             end for j;
             go to EX
          end if;
EX:    end CHECKERROR;
       comment library INVERT2;
       comment library RUKU3;
       comment library NOLEQ3;
       comment library EXPRES1;
       first := true;
       drumplace := 68×40 - 1 + Z;
       from drum(xpoint);
       drumplace := 69×40 - 1 + N;
       from drum(ystart);
       drumplace := 70×40 - 1 + N;
       from drum(delta);
       drumplace := 71×40 - 1 + K;
       if K > 0 then from drum(CONST);
       drumplace := 72×40 - 1 + N;
       from drum(TERM);
       if opterm > 0 then
       begin
          drumplace := dtop - 1000;
          from drum(SPECOP)
       end if opterm > 0;
comment
```

```
;
        if boundcond then
        begin
           drumplace := 73×40 - 1 + P;
           from drum(xbound);
           drumplace := 74×40 - 1 + N;
           from drum(yinc);
           begin comment yerror-transfer;
              array yerror[1:N];
              drumplace := 75×40 - 1 + N;
              from drum(yerror);
              j := 0;
              for i := 1 step 1 until N do
              begin
                 R := yinc[i];
                 if R ≠ 0 then
                 begin
                    if j < C then j := j + 1
                    else
                    begin
L1:                    writecr;
                       writetext(≮<Inconsistent boundary condition data≯);
                       go to ZZ
                    end error;
                    corstart[j] := 0;
                    delcor[j] := R;
                    epscor[j] := yerror[i]
                 end if R ≠ 0
              end for i;
              if j < C then go to L1
           end yerror-transfer;
           count := 0
        end if boundcond;
        PSHIFT(20);
        outsp(30);
        outtext(≮<INTEGRATION RESULTS≯);
        LINE(2);
        final := ¬ boundcond;
comment
```

```
;
     if boundcond then
     begin
        outtext(⟨⟨Boundary condition errors:⟩);
        LINE(2)
     end if boundcond;
H1:  x := x1;
     dercount := 0;
     for i := 1 step 1 until N do
     y[i] := ystart[i];
     if boundcond then
     begin
        NOLEQ3(C, count, 50, false, corstart, delcor, cor, epscor, ERROR,
               Eold, E0, 1₁₀-20, 4, F1, E1);
K1:     if (count-1)÷(C+1)×(C+1) = count - 1 then LINE(2);
        outtext(⟨⟨ystart:      ⟩);
        j := 0;
        for i := 1 step 1 until N do
        begin
           if yinc[i] ≠ 0 then
           begin
              j := j + 1;
              y[i] := y[i] + cor[j]
           end if not zero;
           output(⟨-n.dddddd₁₀-dd⟩, outsp(2), y[i]);
           if i÷4×4 = i ∧ i < N then
           begin
              LINE(1);
              outsp(12)
           end if
        end for i;
        LINE(1);
        for j := 1 step 1 until C do
        begin
           array ELEM[1:1];
           drumplace := dtop - (j-1)×(P×N+1);
           from drum(ELEM);
           ERROR[j] := - ELEM[1]
        end for j
     end if boundcond;
comment
```

```
;
      if final then
      begin
         LINE(1);
         PSHIFT(10);
         outtext({<     x        });
         for j := 1 step 1 until N do
         begin
            outsp(7);
            outtext({<y});
            if j < 10 then
            begin
               output({n}, j);
               outsp(5)
            end
            else
            begin
               output({nd}, j);
               outsp(4)
            end;
            if j÷4×4 = j ∧ j < N then
            begin
               LINE(1);
               outsp(12)
            end if
         end for j;
         LINE(2)
      end if final;
      PR;
      CHECKERROR;
      for zo := 1 step 1 until Z do
      begin
         xe := xpoint[zo];
         RUKU3(var, N, FUNCTION, x, y, xe, delta[var], first, outside);
         PR;
         CHECKERROR
      end for zo;
      if final then LINE(2);
      if boundcond then
      begin
         outtext({<error:       });
comment
```

```
;
          for i := 1 step 1 until C do
          begin
             output(⟨-n.dddddd₁₀-dd⟩, outsp(2), ERROR[i]);
             if i:4×4 = i ∧ i < N then
             begin
                LINE(1);
                outsp(12)
             end if
          end for i;
          LINE(2);
          if ¬ final then go to H1;
          go to G1;
E1:       LINE(1);
          outtext(⟨⟨ERROR⟩);
          LINE(3);
F1:       final := true;
          go to K1;
G1:    end if boundcond
    end calculation;
    sectno := sectno + 1;
    go to AA;
ZZ: end of program;
```

5.3.2. ALGOL program DE-2.

Program DE-2.

```
begin
   boolean nont, nont0, nonz, explicite, newterm, transfer;
   integer LRest, calcno, sectno, pageno, dtop, NRAD, NAX, nradp, naxp, RATERM,
           ZTERM, K, i, j, r, climit;
   real delx, delr, tau, beta, gamma, lambda1, lambda2, lambda3, eps;
   comment library LINE;
   comment library PSHIFT;
   sectno := 1;
AA: begin comment cover and headline;
      integer i;
      array B[0:39];
      comment library CENTEXT;
      comment library COVER1;
      comment library HEADLINE;
      if sectno > 1 then go to BB;
      COVER1(3, 31,
      {<Solution of Partial Differential Equations in Cylindric Catalytic Reactors
      {<GIER Program DE-2}>);
      for i := 0 step 1 until 39 do B[i] := 0;
      dtop := drumplace;
      for i := to drum (B) while drumplace > 2679 do;
BB:   HEADLINE(20, ZZ)
   end of cover and headline;
   begin comment input block;
      integer m, sum;
      array DATA[0:39];
      comment library READ5;
      comment library READ6;
      READ6( 67, 70);
      drumplace := 40×67 + 39;
      from drum(DATA);
      NRAD := DATA[0];
      NAX := DATA[1];
      nradp := DATA[2];
      naxp :=  DATA[3];
      delx := DATA[4]/NAX;
      delr := DATA[5]/(2×NRAD);
      nont := DATA[6] = 1;
comment
```

```
;
      nont0 := DATA[7] = 1;
      nonz  := DATA[8] = 1;
      RATERM := DATA[9];
      ZTERM := DATA[10];
      K := DATA[11];
      explicite := DATA[12] = 1;
      transfer := DATA[13] = 1;
      climit := DATA[14];
      drumplace := 40×68 + 39;
      from drum(DATA);
      tau := DATA[0];
      beta := DATA[1];
      gamma := DATA[2];
      lambda1 := DATA[3];
      lambda2 := DATA[4];
      lambda3 := DATA[5];
      eps := DATA[6];
      sum := RATERM + ZTERM;
      begin comment input of operations;
         array OLD, NEW[1:sum];
         newterm := false;
         drumplace := 71×40 - 1 + sum;
         from drum(OLD);
         for m := 1 step 1 until sum do
         NEW[m] := OLD[m];
         for m := 1 step 1 until sum do
         READ5(NEW[m], m, L1);
         for m := inchar while char ≠ 53 do;
L1:      for m := 1 step 1 until sum do
         if NEW[m] ≠ OLD[m] then newterm := true;
         drumplace := 71×40 - 1 + sum;
         to drum(NEW)
      end input of operations;
      i := 0;
      if nont then i := 1;
      if nont0 then i := i + 1;
      if nonz then i := i + 1;
comment
```

```
;
        for j := 1 step 1 until i do
        begin
           array INLET[0:NRAD];
           drumplace := dtop - (NRAD+1)×(j-1);
           from drum(INLET);
           for m := 0 step 1 until NRAD do
           READ5(INLET[m], m, L1);
           for m := inchar while char ≠ 53 do;
L1:        drumplace := dtop - (NRAD+1)×(j-1);
           to drum(INLET)
        end for j
     end input;
     begin comment data printing;
        integer mcount, mfact;
        array DATA[0:39];
        comment library TABLE2;
        comment library T3;
        comment library EXPRES2;
        TABLE2(67, 8, NP1,
        ⟨<CALCULATION PARAMETERS⟩, 29);
        T3(⟨<Number of radial steps⟩, 33, ⟨-nddd⟩);
        T3(⟨<Number of axial steps⟩, 34, ⟨-nddd⟩);
        T3(⟨<Number of radial printing points⟩, 23, ⟨-nddd⟩);
        T3(⟨<Number of axial printing points⟩, 24, ⟨-nddd⟩);
        T3(⟨<Cylinder height (m)⟩, 37, ⟨-ndd.d00₁₀-dd⟩);
        T3(⟨<Cylinder diameter (m)⟩, 35, ⟨-ndd.d00₁₀-dd⟩);
        T3(⟨<Non-uniform t-inlet, 0: no, 1: yes⟩, 21, ⟨-nddd⟩);
        T3(⟨<Non-uniform t0-inlet, 0: no, 1: yes⟩, 20, ⟨-nddd⟩);
        T3(⟨<Non-uniform z-inlet, 0: no, 1: yes⟩, 21, ⟨-nddd⟩);
        T3(⟨<Number of RATE terms⟩, 35, ⟨-nddd⟩);
        T3(⟨<Number of ZFUNC terms⟩, 34, ⟨-nddd⟩);
        T3(⟨<Number of constants⟩, 36, ⟨-nddd⟩);
        T3(⟨<Integration method, 1: explicite, 2: implicite⟩, 9, ⟨-nddd⟩);
        T3(⟨<Transfer from last section, 0: no, 1: yes⟩, 14, ⟨-nddd⟩);
        T3(⟨<Maximum number of iterations⟩, 27, ⟨-nddd⟩);
        LINE(2);
NP1:    TABLE2(68, 8, NP2,
        ⟨<HEAT TRANSFER AND DIFFUSION COEFFICIENTS⟩, 20);
comment
```

```
;
     T3(⟨<Adiabatic temperature increase, tau (deg.C)⟩, 12, ⟨-nddd.dd⟩);
     T3(⟨<Thermal diffusivity, beta (m)⟩, 27, ⟨-ndd.dd00₁₀-dd⟩);
     T3(⟨<Mass diffusivity, gamma (m)⟩, 29, ⟨-ndd.dd00₁₀-dd⟩);
     T3(⟨<Bed cooling constant, lambda1 (/m)⟩, 22, ⟨-ndd.dd00₁₀-dd⟩);
     T3(⟨<Wall cooling constant, lambda2 (/m)⟩, 21, ⟨-ndd.dd00₁₀-dd⟩);
     T3(⟨<Wall ratio, lambda3 = 2U/K (/m)⟩, 25, ⟨-ndd.dd00₁₀-dd⟩);
     T3(⟨<Permissible error in integration (deg.C)⟩, 15, ⟨-nddd.dddd⟩);
     LINE(2);
NP2: TABLE2(69, 8, NP3,
     ⟨<INLET TEMPERATURES AND COMPOSITION⟩, 23);
     T3(⟨<Catalyst temperature (deg.C)⟩, 27, ⟨-nddd.dd⟩);
     T3(⟨<Cooling tube temperature (deg.C)⟩, 23, ⟨-nddd.dd⟩);
     T3(⟨<Annulus temperature (deg.C)⟩, 28, ⟨-nddd.dd⟩);
     T3(⟨<Mole per cent of key component⟩, 25, ⟨-nddd.dddd⟩);
     LINE(2);
NP3: TABLE2(70, 8, NP4,
     ⟨<LIST OF CONSTANTS⟩, 31);
     for i := 1 step 1 until K do
     begin
        outtext(⟨<Constant no.⟩);
        output(⟨-nd⟩, i);
        outsp(41);
        output(⟨-ndd.dd00₁₀-dd⟩, DATA[i-1]);
        LINE(1)
     end for i;
     LINE(2);
NP4: if newterm then
     begin
        integer array OP1[1:RATERM], OP2[1:ZTERM];
        procedure START;
        if j = 1 then outtext(if i = 1
        then ⟨<RATE     ⟩
        else ⟨<ZFUNC    ⟩)
        else outsp(8);
        drumplace := 71×40 - 1 + RATERM + ZTERM;
        from drum(OP2);
        from drum(OP1);
        PSHIFT(12);
        outsp(31);
comment
```

```
;
         outtext(⟨<LIST OF OPERATIONS⟩);
         LINE(2);
         outtext(⟨<Function    Operation⟩);
         LINE(2);
         for i := 1, 2 do
         begin
            EXPRES2(if i = 1 then RATERM else ZTERM, j, if i = 1 then
            OP1[j] else OP2[j], ⟨<x⟩, ⟨<CONST⟩, ⟨<FUNCT⟩, START);
            LINE(1)
         end for i;
         LINE(2)
      end if newterm
   end data printing;
   begin comment integration;
      integer count, cmax;
      real tcat, zcat, x, xstart, ttub, tann, mean;
      integer array OP1[1:RATERM], OP2[1:ZTERM];
      array t, t0, z[0:NRAD], func[1:2], CONST[1:K];
      comment library EXPRES1;
      comment library PAPADEQ1;
      comment library PAPADEQ2;
      real procedure RATE;
      begin
         func[1] := tcat;
         func[2] := zcat;
         RATE := EXPRES1(RATERM, j, OP1[j], x, CONST, func, failure)
      end RATE;
      real procedure ZFUNC;
      ZFUNC := EXPRES1(ZTERM, j, OP2[j], x, CONST, func, failure);
      procedure failure;
      go to EX;
      procedure CHECK;
      begin
         if count>cmax then cmax := count;
         if i = 0 ∨ i÷naxp×naxp = i then
         begin
            procedure PLINE(fact, data, text);
            value fact;
comment
```

```
;
              real fact;
              array data;
              string text;
              begin
                 outtext(text);
                 for r := 0 step 1 until nradp do
                 begin
                    output(⟨-nddd.dd⟩, fact×data[r×NRAD:nradp]);
                    if (r+1):8×8 = r + 1 ∧ r ≠ 0 ∧ r ≠ nradp then
                    begin
                       LINE(1);
                       outsp(8)
                    end if
                 end for r
              end PLINE;
              LINE(1);
              output(⟨nd.dd⟩, x);
              PLINE(1, t, ⟨< t:⟩);
              if lambda1 ≠ 0 then
              begin
                 LINE(1);
                 PLINE(1, t0, ⟨<     t0:⟩)
              end if;
              if gamma ≠ 0 then
              begin
                 LINE(1);
                 PLINE(100, z, ⟨<      z:⟩)
              end if;
              LINE(1);
              outtext(⟨<   tann:⟩);
              output(⟨-nddd.dd⟩, tann);
              if ¬ explicite then
              begin
                 LINE(1);
                 outtext(⟨<  count:⟩);
                 output(⟨-nddd⟩, sign(count)×(count-1))
              end if implicite
           end if print
        end CHECK;
     comment
```

```
;
      naxp := NAX:naxp;
       drumplace := 71×40 - 1 + RATERM + ZTERM;
       from drum(OP2);
       from drum(OP1);
       drumplace := 70×40 - 1 + K;
       from drum(CONST);
       xstart := x := 0;
       begin comment inlet data;
          array D[0:3];
          drumplace := 69×40 + 3;
          from drum(D);
          tcat := D[0];
          ttub := D[1];
          tann := D[2];
          zcat := D[3]/100;
          for r := 0 step 1 until NRAD do
          begin
             t[r] := tcat;
             t0[r] := ttub;
             z[r] := zcat
          end for r;
          drumplace := dtop;
          if nont then from drum(t);
          if nont0 then from drum(t0);
          if nonz then
          begin
             from drum(z);
             for r := 0 step 1 until NRAD do
             z[r] := z[r]/100
          end if nonz;
          if transfer then
          begin
             drumplace := dtop - 3×(NRAD+1);
             from drum(t);
             from drum(t0);
             from drum(z);
             from drum(D);

comment
```

```
;
        tann := D[3];
        x := xstart := D[2]
      end if transfer
    end inlet data;
    PSHIFT(20);
    outsp(30);
    outtext(⦃<INTEGRATION RESULTS⦄);
    LINE(2);
    outtext(⦃<         Radial direction —>⦄);
    LINE(1);
    outtext(⦃<x(m)⦄);
    LINE(1);
    i := 0;
    cmax := count := 0;
    CHECK;
    if explicite then
    PAPADEQ1(r, NRAD, i, NAX, tann, delr, x, xstart, delx, tcat, ttub, zcat,
          RATE, ZFUNC, beta, gamma, lambda1, tau, lambda2, lambda3, t, t0,
          z, CHECK)
    else
    PAPADEQ2(r, NRAD, i, NAX, count, climit, tann, delr, x, xstart, delx,
          tcat, ttub, zcat, RATE, ZFUNC, beta, gamma, lambda1, tau, lambda2,
          lambda3, eps, t, t0, z, CHECK);
    LINE(3);
    PSHIFT(10);
    begin comment printing of mean values;
      procedure MEAN(fact, data, text);
      value fact;
      real fact;
      array data;
      string text;
      begin
        outtext(text);
        mean := 0;
        for r := 0 step 1 until NRAD do
        mean := mean + r×data[r];
        output(⦃-nddd.dd⦄, fact×mean/NRAD/(1+NRAD)×2)
      end MEAN;
comment
```

```
;
           outtext(⟨⟨Radial mean values:⟩);
           LINE(2);
           MEAN(1, t, ⟨⟨t:⟩);
           if lambda1 ≠ 0 then
           MEAN(1, t0, ⟨⟨     t0:⟩);
           if gamma ≠ 0 then
           MEAN(100, z, ⟨⟨      z:⟩);
           LINE(3)
        end printing of mean values;
        if ¬ explicite then
        begin
           outtext(⟨⟨Maximum number of iterations:⟩);
           output(⟨-nddd⟩, cmax - 1);
           LINE(3)
        end if implicite;
        go to OUT;
EX:     LINE(1);
        outtext(⟨⟨Calculation trouble⟩);
        LINE(2);
OUT:     drumplace := dtop - 3×(NRAD+1);
        to drum(t);
        to drum(t0);
        to drum(z);
        func[2] := tann;
        func[1] := x;
        to drum(func)
     end integration;
     sectno := sectno + 1;
     go to AA;
ZZ: end of program;
```

1097

27.10.1965