MODERN TAYLOR SERIES METHOD

Jiří Kunovský

Department of Computer Science and Engineering,
Faculty of Electrical Engineering and Computer Science,
Technical University of Brno,
Božetěchova 2, 612 66 BRNO, The Czech Republic,
Telephone: +420-5-7275234 (+420-5-41212219),
Fax:+420-5-41211141
E-Mail kunovsky@dcse.fee.vutbr.cz

BRNO 1994

Contents

Chapter 1

MODERN TAYLOR SERIES METHOD

1.1 Introduction

By a numerical solution of an ordinary differential equation

$$y' = f(t, y), \qquad y(t_0) = y_0 \tag{1}$$

we understand the finding of a sequence:

$$[y(t_0) = y_0], \quad [y(t_1) = y_1], \quad [y(t_2) = y_2], ..., [y(t_n) = y_n].$$

The best-known and most accurate method of calculating a new value of a numerical solution of a differential equation (1) is to construct the Taylor series in the form

$$y_{n+1} = y_n + h * f(t_n, y_n) + \frac{h^2}{2!} * f^{[1]}(t_n, y_n) + \ldots + \frac{h^p}{p!} * f^{[p-1]}(t_n, y_n), \tag{2}$$

where h is the integration step.

Methods of numerical solutions of differential equations have been studied since the end of the last century. A large number of integration formulas have been published especially for solving special systems of differential equations. In general, it was not possible to choose the best method but for a subclass of tasks defined by similar properties the most suitable method could always be found.

The presented " Modern Taylor Series Method " has proved to be both very accurate and fast. It is based on a direct use of the Taylor series.

The main idea behind the Modern Taylor Series Method is an automatic integration method order setting, i.e. using as many Taylor series terms for computing as needed to achieve the required accuracy.

1.2 Positive Properties of the Taylor Series Method

To demonstrate the positive properties of the Taylor series method the following differential equation is solved

$$y' = aycost, \quad y(0) = 1, \quad a = 2. \tag{3}$$

The exact solution of (3) is

$$y = e^{asint} \tag{4}$$

so that the absolute error of the numerical solution can be determined as the difference between the numerical and exact solutions.

To numerically solve (3) by the Taylor series method (using (2)) we need to calculate the following formulas:

$$\begin{aligned}
f(t,y) &= aycost;\\
f^{[1]}(t,y) &= af(t,y)cost - aysint;\\
f^{[2]}(t,y) &= af^{[1]}(t,y)cost - aycost - 2af(t,y)sint;\\
f^{[3]}(t,y) &= af^{[2]}(t,y)cost - 3af(t,y)cost - 3af^{[1]}(t,y)sint + aysint;
\end{aligned}$$

$$\begin{aligned}
f^{[4]}(t,y) &= af^{[3]}(t,y)cost - 6af^{[1]}(t,y)cost + aycost - 4af^{[2]}(t,y)sint +\\
&\quad 4af(t,y)sint;
\end{aligned}$$

$$\begin{aligned}
f^{[5]}(t,y) &= af^{[4]}(t,y)cost - 10af^{[2]}(t,y)cost + 5af(t,y)cost -\\
&\quad 5af^{[3]}(t,y)sint + 10af^{[1]}(t,y)sint - aysint;
\end{aligned}$$

$$\begin{aligned}
f^{[6]}(t,y) &= af^{[5]}(t,y)cost - 15af^{[3]}(t,y)cost + 15af^{[1]}(t,y)cost - \\
&\quad aycost - 6af^{[4]}(t,y)sint + 20af^{[2]}(t,y)sint - \\
&\quad 6af(t,y)sint;
\end{aligned}$$

$$\begin{aligned}
f^{[7]}(t,y) &= af^{[6]}(t,y)cost - 21af^{[4]}(t,y)cost + 35af^{[2]}(t,y)cost - \\
&\quad 7af(t,y)cost - 7af^{[5]}(t,y)sint + 35af^{[3]}(t,y)sint - \\
&\quad 21af^{[1]}(t,y)sint + aysint;
\end{aligned}$$

$$\begin{aligned}
f^{[8]}(t,y) &= af^{[7]}(t,y)cost - 28af^{[5]}(t,y)cost + 70af^{[3]}(t,y)cost - \\
&\quad 28af^{[1]}(t,y)cost + aycost - 8af^{[6]}(t,y)sint + \\
&\quad 56af^{[4]}(t,y)sint - 56af^{[2]}(t,y)sint + 8af(t,y)sint;
\end{aligned}$$

$$\vdots \tag{5}$$

Note: One way to calculate the formulas (5) is to use the DERIVE system (a special program for symbolic differentiation).

Methods of different orders can be used in a computation. For instance the 1st order method (ORD=1) means that when computing the new value y_{n+1} only the first Taylor series term is taken into account

$$y_{n+1} = y_n + h * f(t_n, y_n), \tag{6}$$

the 2nd order method (ORD=2) uses Taylor series terms up to the second power of the step h

$$y_{n+1} = y_n + h * f(t_n, y_n) + \frac{h^2}{2!} * f^{[1]}(t_n, y_n), \tag{7}$$

etc.

The quality of the computation can be judged by the tables Tab.1 and Tab.2 (integration step h=0.1s and printing interval 1s were parameters of the computations). Tab.1 and Tab.2 present the numerical solution of (3) using formulas (5).

t(s)	ORD4(%)	ORD6(%)	ORD8(%)
1.0	5.5e-04	3.7e-06	1.0e-08
2.0	2.7e-04	9.2e-07	4.5e-09
3.0	7.9e-04	2.5e-07	7.1e-09
4.0	9.4e-04	1.7e-05	1.9e-08
5.0	8.0e-05	2.6e-05	2.3e-08
6.0	7.7e-04	3.7e-06	9.4e-09
7.0	1.3e-03	4.2e-06	3.9e-09
8.0	1.1e-03	2.8e-06	8.3e-09
9.0	1.5e-03	2.6e-06	4.8e-09
10.0	1.2e-03	9.6e-06	1.6e-08

Tab.1

The absolute value of the relative error of the computation at chosen times (1,2,..10 s) is the main criterion for evaluating the computation (Tab.1). In the column "ORD4" the absolute values of the relative error of the computation are shown; the 4th order method (ORD=4) was used. Similarly, the 6th order method (ORD=6) and the 8th order method (ORD=8) were used for the computation of "ORD6" and "ORD8".

In this paper we also define, as an important criterion, the tallying of the valid figures of a numerical computation with the analytical solution - for clarity in table Tab.2 only those digits of the numerical solution of (3) tallying with the analytical solution are shown.

t(s)	NUM4	NUM6	NUM8	MTSM
1.0	5.3813	5.381364	5.38136451	5.38136451648877
2.0	6.1631	6.1631921	6.163192175	6.16319217563612
3.0	1.326	1.32609696	1.326096966	1.3260969664414
4.0	0.22011	0.22011	0.2201150333	0.220115033306814
5.0	0.146922	0.146922	0.1469227193	0.146922719324015
6.0	0.57187	0.5718771	0.571877199	0.571877199752585
7.0	3.7209	3.720928	3.720928364	3.72092836426954
8.0	7.233	7.23345	7.233452838	7.23345283888572
9.0	2.2801	2.2801402	2.28014028	2.28014028705485
10.0	0.33687	0.3368753	0.336875375	0.336875375797793

Tab.2

The 4th order method (ORD=4) was used for the computation of results shown in the column "NUM4". Similarly, the 6th order method (ORD=6) and

8th order method (ORD=8) were used for the computation of "NUM6" and "NUM8". Results in the column "MTSM" were obtained by the Modern Taylor Series Method.

The Modern Taylor Series Method used in the computations increases the method order ORD automatically, i.e. the values of the terms

$$\frac{h^p}{p!} * f^{[p-1]}(t_n, y_n)$$

are computed for increasing integer values of p until adding the next term does not improve the accuracy of the solution.

Even though calculating (5) by the DERIVE program requires a substantial amount of time, it stresses the important fact that the numerical calculation is most exact when the method order ORD is accordingly high for the integration step h given.

For completeness' sake, the numerical solution of the differential equation (3) (for a=2) as a function of time is shown in Fig.1.1.

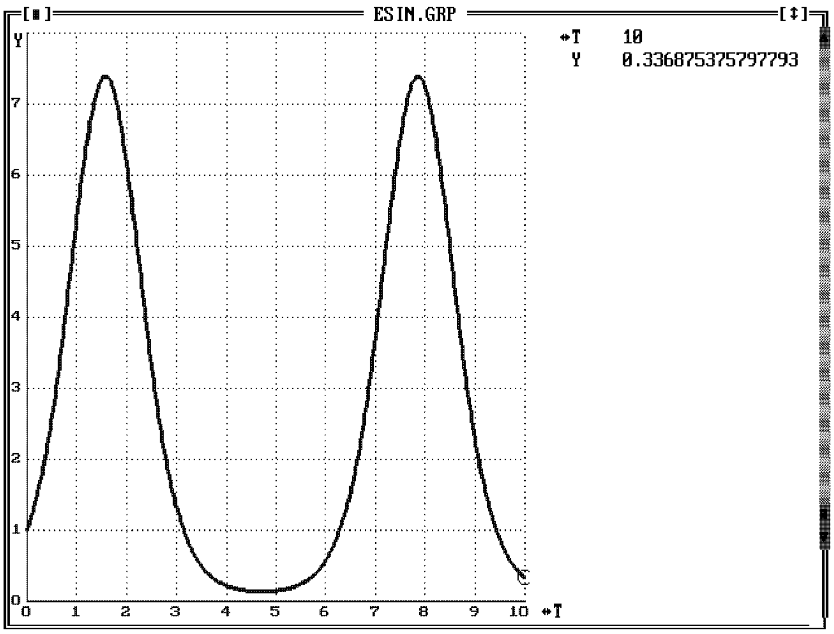

Figure 1.1:

The Modern Taylor Series Method was used for the solution. In the right-hand part of Fig.1.1 a particular time value T and the corresponding value of Y are shown.

1.3 Technical Initial Problems

The main problem connected with using Taylor series (in the form of (2)) is the need to generate higher derivatives$f^{[1]}, f^{[2]}, \dots$. This is in fact the reason why Runge-Kutta formulas of various orders have been used.

If we succeed, however, to obtain the terms with higher derivatives, the accuracy of calculations by Taylor series method is extreme (it is in fact only limited by the type of the arithmetic unit used). This is typical, in particular, of the solution of the technical initial problems.

Technical initial problems are defined as initial problems where the right-hand side functions of the system are those occurring in the technical practice, that is functions generated by adding, multiplying and superposing elementary functions. Such systems can be expanded into systems with polynomials on the right-hand sides of the equations. In such a case the Taylor series terms can easily be calculated.

To demonstrate this, the equation (3)

$$y' = aycost, \qquad y(0) = y_0$$

is analyzed again. A simple computation scheme based on equation (1) follows:

$$f(t,y) = aycost.$$

Let $v = cost.$

Then

$$f(t,y) = ayv,$$

$$f^{[1]}(t,y) = a(f(t,y)v + yv'),$$

$$f^{[2]}(t,y) = a(f^{[1]}(t,y)v + 2f(t,y)v' + yv''),$$

.

.

$$f^{[p-1]}(t,y) = a(\sum_{i=0}^{p-2} f^{[p-2-i]}(t,y)\ v^{[i]}\ \binom{p-1}{i} + yv^{[p-1]}), \qquad (\qquad p \geq 2),$$

where

$$v' = -sint,$$

$$
\begin{aligned}
u &= sint, \\
v^{[p]} &= -v^{[p-2]} \qquad (\quad p \geq 2), \\
u^{[p-1]} &= \quad v^{[p-2]} \qquad (\quad p \geq 2).
\end{aligned}
$$

Similar constructions can be created for all elementary functions, such as exp, sin, cos, tg, cotg, ln, sinh, ...

The idea above requires software capable of automatically performing the decomposition of the right-hand sides of ordinary differential equations.

This new approach has been implemented in a simulation language TKSL/386 (an implementation of the Taylor Kunovsky Simulation Language on an Intel 80386 based personal computer).

In fact, the well-known rules of differential and integral calculus have been used.

Chapter 2

SIMULATION LANGUAGE TKSL/386

Theoretical work on the numerical solution of ordinary differential equations by the Taylor series method has been going on for a number of years. The simulation language TKSL/386 has been created to test the properties of the technical initial problems and to test an algorithm for Taylor series method.

TKSL/386 has the following features

- user friendly environment (Turbo Vision),
- adjustable computation accuracy,
- adjustable method order,
- computation with variable integration step h,
- exact detection of discountinuities.

TKSL offers the most practical features available today and it also presents an entirely new approach to solving continuous systems:

- on analysing a particular continuous system only the required precision of the solution is defined,
- the implicitly built-in integration method of order 64 (with the integration step h automatically defined/fixed) is distinctly superior in the computation quality to other integration algorithms currently used,
- the order of the implicitly built-in integration method can interactively be increased (yielding a new computation quality),
- stiff systems are viewed as systems of ordinary differential equations (and as such are automatically solved with the required precision).

2.1 Programming in TKSL/386

Programming in TKSL/386 is very easy. The menu system in TKSL/386 is very similar to that in TURBO PASCAL 6.0. Using TKSL/386 is demonstrated again on the equation (3)

$$y' = aycost, \quad y(0) = 1, \quad a = 2.$$

The corresponding source text in TKSL/386 is

```
var y;
const a=2,
      tmax=10,
      dt=0.2,
      eps=1e-20;
system
      y'= a*y*cos(t)    & 1;
sysend.
```

All variables that will be needed are declared in the line starting with **var**. All necessary constants are declared in the line starting with **const** (**a** is a constant in the equation, **tmax** is the maximal computation time , **dt** is the step size and **eps** is the required accuracy).

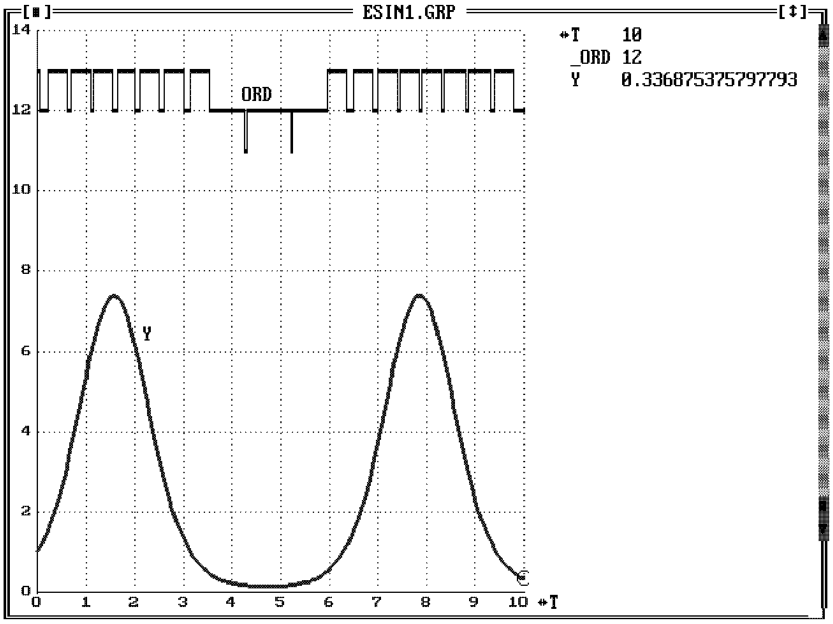

Figure 2.1:

The equation (3) is declared in block mode (between **system** and **sysend**). The initial condition for differential equation (3) is written in the form & *initial condition*.

The two typical windows which are displayed on the screen immediately after the program is started can be seen in Fig.2.1. In the right-hand part of Fig.2.1 the symbol ↔ marks the variable which is plotted on the horizontal axis (in this case the time T).

The variable T is increased by the preset step size dt and at the same time the corresponding values of the variables ORD and y (ORD stands for the method order) are shown in the right-hand part of Fig.2.1. In the left-hand part of Fig.2.1 two functions of time are shown in the course of the computation. There are graphs of y and ORD . The last function has not yet been published in any paper on this subject.

In the above example (for a=2, integration step h=0.05s) the value of ORD ranges between the values 11 and 13.

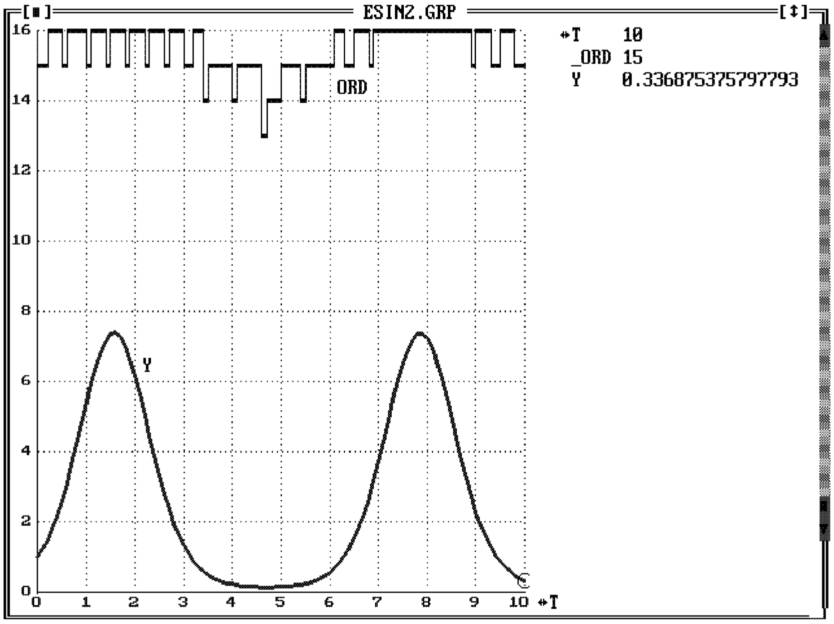

Figure 2.2:

If the value of integration step increases to h=0.1s , the corresponding resulting solution of the equation (3) (again for a=2) is in Fig.2.2 . The value Y of the numerical solution at time T=10s is of course the same, only the value of ORD, as expected, has increased (ORD ranges between 13 and 16).

The value of the coefficient **a** can be changed easily in the simulation language TKSL/386. The numerical solution of the equation (3) (for a=100, h=0.02) and the value of ORD as functions of time are shown in Fig.2.3 .

It is typical of the Modern Taylor Series Method that the value of ORD

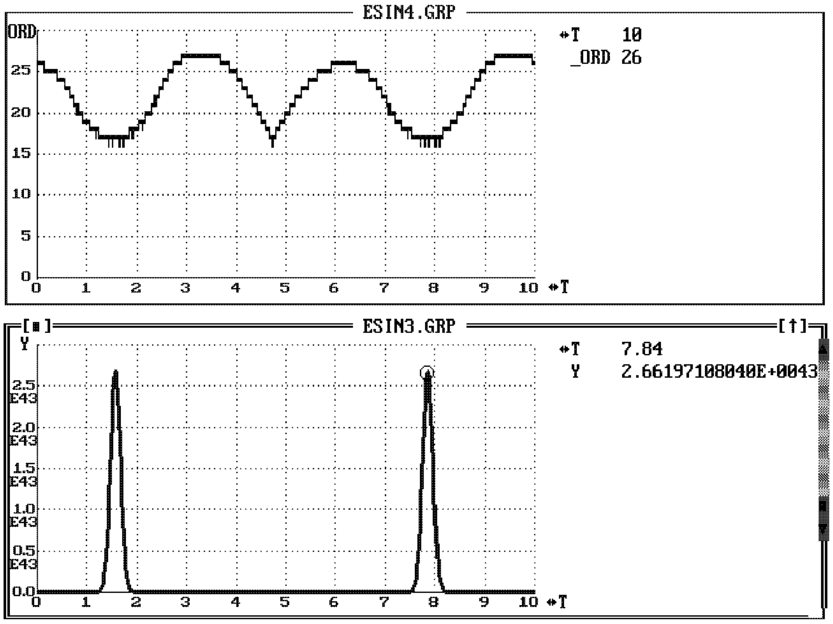

Figure 2.3:

changes during the computation.

The high accuracy of the TKSL/386 is demonstrated on the following system of equations

$$y' = aycost \quad y(0) = 1 \tag{8}$$
$$x' = -axcost \quad x(0) = 1 \tag{9}$$
$$z = xy \tag{10}$$

In the left-hand part of Fig.2.4, x, y, z and ORD as functions of time are shown in the course of the computation (for a=2, h=0.1s). In the right-hand part of Fig.2.4 the values of ORD, x, y, z are plotted (for T=10s).

The system of equations (8), (9), (10) was deliberately designed for the variable z to characterize the accuracy of the computation.

For the test function

$$z = x \cdot y \quad we \quad have \quad z = 1,$$

since the exact solution of (8) is

$$y = e^{asint}$$

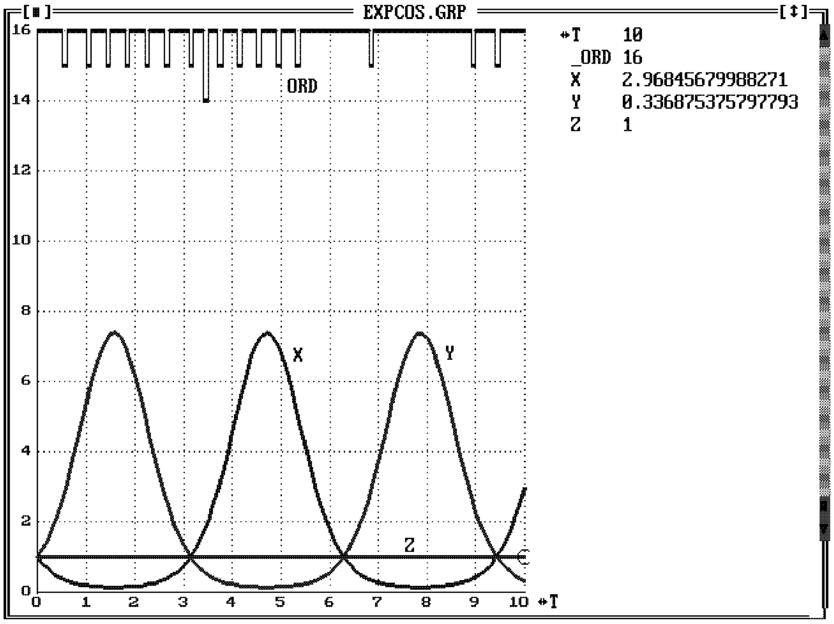

Figure 2.4:

and the exact solution of (9) is

$x = e^{-asint}$.

The accuracy of the computation is preserved even if the variables reach values of 10^{43} and 10^{-44} by order of magnitude. The numerical solution of the system (8),(9),(10) reaches these values for a=100 (Fig.2.5).

The corresponding test function Z and the values of ORD as a function of time (for h=0.01s, a =100) are shown in Fig.2.6 - in the part labelled EXPCOS5. In the part labelled EXPCOS6 time functions of Z and ORD (for h= 0.1s, a=100) are shown. The value of the test function z is constantly at z=1, only the values of the function ORD increase as h grows, as expected.

The achieved high accuracy of the computation of the expression

$$z = x \cdot y = 1$$

is based on the fact that the new value of y_{n+1} is calculated in each step automatically until the adding of the next Taylor series terms has no effect.

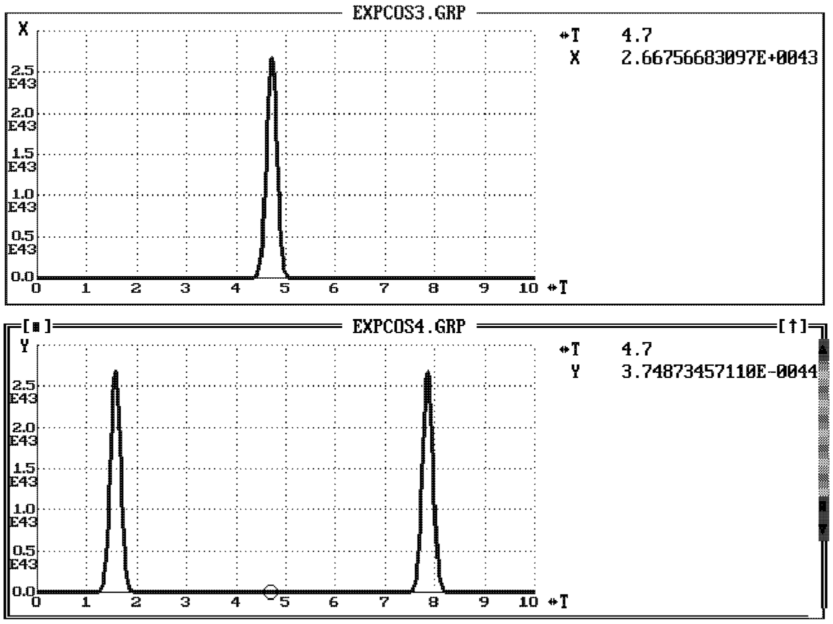

Figure 2.5:

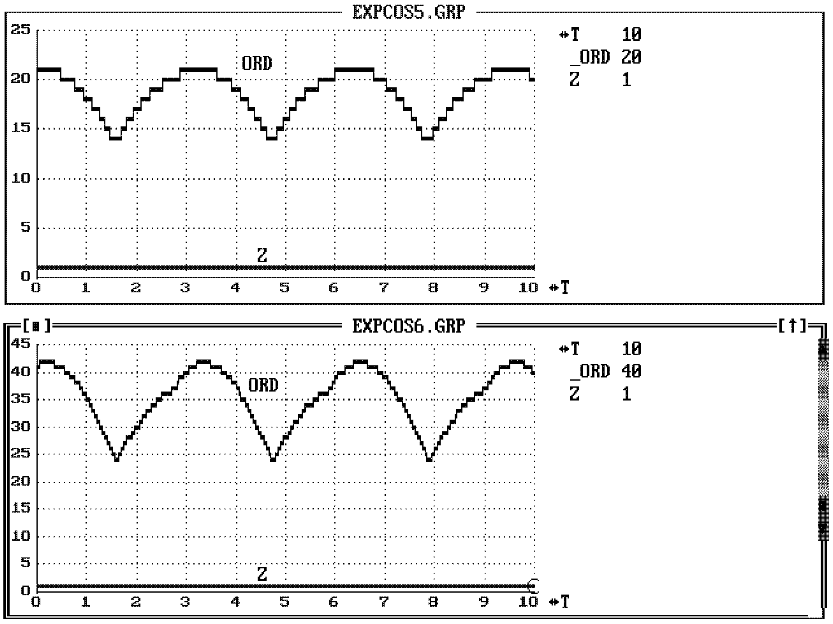

Figure 2.6:

Chapter 3

POLYNOMIAL FUNCTIONS

3.1 An Accurate Solution

If the right-hand side of a first order differential equation

$$y' = a * t^k, \quad y(0) = y_0 \tag{11}$$

has the form of a polynomial of a degree k, then the integration method of order ORD = k+1 will ensure the absolute accuracy of calculations with an arbitrary integration step.

The numerical solution of (11) by the Taylor series method is

$$y_{n+1} = y_n + h * a * t_n^k + \frac{h^2}{2} * a * k * t_n^{k-1} + ... + \frac{h^{k+1}}{k+1} * a \tag{12}$$

The integration step h can be chosen arbitrarily; the numerical solution of (11) will be absolutely accurate.

Note: The Modern Taylor Series Method used in the computations automatically sets the value

$$ORD = k + 1.$$

To demonstrate the effect of a calculation with an arbitrary integration step two simple examples will be shown.

Example 1:
$y' = a, \quad y(0) = 0.$ (13)

The numerical solution of (13) by the Taylor series method (using (2)) is

$$y_{n+1} = y_n + h * y' \tag{14}$$

$(y'' = y''' = ... = y^{(n)} = 0),$

or
$$y_{n+1} = y_n + h * a. \tag{15}$$

The integration step h can be chosen arbitrarily for the calculation; the numerical solution of (13) is absolutely accurate due to (15). (In Fig.3.1 the solution of (13) is shown for a=1, h=0.2s, tmax=10s).

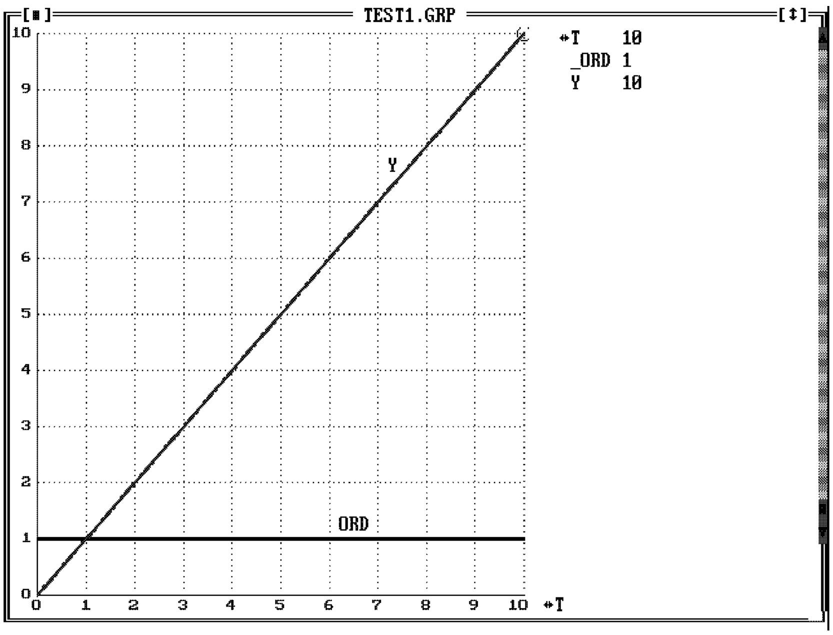

Figure 3.1:

Note: The Modern Taylor Series Method used in the computations automatically sets the value

$$ORD = 1.$$

Example 2:

$$y' = a * t, \quad y(0) = 0. \tag{16}$$

The numerical solution of (16) by the Taylor series method is

$$y_{n+1} = y_n + h * y' + \frac{h^2}{2} * y'' \tag{17}$$

$$(y''' = y^{(4)} = ... = y^{(n)} = 0),$$

or

$$y_{n+1} = y_n + h * a * t_n + \frac{h^2}{2} * a. \tag{18}$$

In Fig.3.2 in the part labelled TEST2 the solution of (16) is shown for a=2, h=0.2s, tmax=10s.

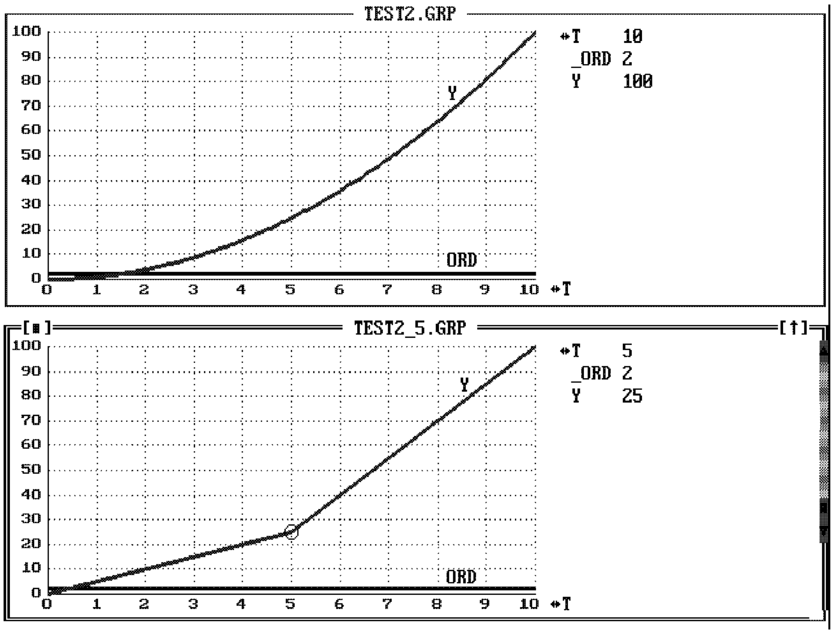

Figure 3.2:

The calculation of the equation (16) can be carried out with an arbitrary integration step h. For illustration, a calculation for h=5s is shown in Fig.3.2. (in the part labelled Test2_5). The graphical software used connects the calculated points of the numerical solution with line segments.

Note: The Modern Taylor Series Method used in the computations automatically sets the value

$$ORD = 2.$$

The analytical solution of (16) is

$$y = a\frac{t^2}{2}.$$

3.2 An Approximate Solution

Let us suppose again that the right-hand side of a first order differential equation has the form of a polynomial of a degree k.

If the order of the integration method is less than k+1, we can get the numerical solution of the differential equation (11) only approximately. The accuracy, in such a case, is dependent on the integration step.

Example 3: Let us determine the value of the solution y of the following equation (19) at time t=10s

$$y' = 6 * t^5 \quad y(0) = 0. \tag{19}$$

In the example we have k=5. If we use the 4th order numerical integration method, i.e. if only the Taylor series terms up to the fourth power of h are only taken into account, we obtain an approximate solution of equation (19). The approximate solution y(10) of equation (19), is shown in Tab.3.

h	y(10)	rel.error(%)	NRS
10	000000.000000000000	1.00E+0002	1
1.0	999720.000000000000	2.80E-0002	10
0.1	999999.970179020196	2.98E-0006	100
0.01	1000000.000020105120	3.34E-0008	1000
0.001	999999.994871600780	1.95E-0007	10000
0.0001	999999.977767592815	2.86E-0006	100000

Tab.3

It can be seen in Tab.3 that we can get close to the exact solution of y(10) by decreasing the integration step h (the exact solution is $y = t^6$; for $t = 10s$ we have $y(10) = 1000000$).

In the column "NRS" the number of computation steps is given after which, with the integration step h given, the point $t_n = 10s$ has been reached.

Note: If we used the method of the corresponding order according to the exact numerical solution by the Taylor series method

$$y_{n+1} = y_n + h * 6 * t_n^5 + h^2 * 15 * t_n^4 + h^3 * 20 * t_n^3 + h^4 * 15 * t_n^2 + +h^5 * 6 * t_n + h^6$$

we would obtain the exact solution y(10) for h=10s in one computation step.

Example 3 was chosen to stress

- the necessity of decreasing the integration step h (if the method order is insufficient),
- the inconvenient fact of the increasing number of computation steps NRS for y(10) (in comparison to the exact numerical solution using the method of the corresponding order),
- the inconvenient influence of the accumulated error when decreasing the integration step h.

Similarly, if we use the 5th order integration method, again, we will obtain an approximate solution y(10) of equation (19) at t = 10s. Results with a higher accuracy (in comparison to Tab.3) are shown in Tab.4.

h	y(10)	rel.error(%)	NRS
10.0	000000.000000000000	1.00E+0002	1
1.0	999990.000000000000	1.00E-0003	10
0.1	999999.999879020196	1.38E-0008	100
0.01	1000000.000020105120	3.37E-0008	1000
0.001	999999.994871600780	1.95E-0007	10000
0.0001	999999.977767592815	2.86E-0006	100000

Tab.4

Chapter 4

HOMOGENOUS DIFFERENTIAL EQUATIONS

Homogenous differential equations with constant coefficients are further typical examples of technical initial problems.

Two examples of homogenous differential equations and their solutions by the Modern Taylor Series Method will be analyzed in this chapter. The rather detailed analysis can be applied to further types of problems.

4.1 Homogenous Equation - Example 1

Let us solve the differential equation

$$y' = y, \quad y(0) = 1. \tag{20}$$

A numerical solution of (20) by the Taylor series method (using (2)) is

$$y_{n+1} = y_n + h * y_n + \frac{h^2}{2!} * y_n + ... + \frac{h^p}{p!} * y_n + ... \tag{21}$$

$$(y = y' = y'' = ... = y^{(p)}),$$

or

$$y_{n+1} = y_n * (1 + h + \frac{h^2}{2!} + ... + \frac{h^p}{p!} + ...). \tag{22}$$

The numerical solution of (20) (using (22) or (23)) will depend on the number of Taylor series terms used.

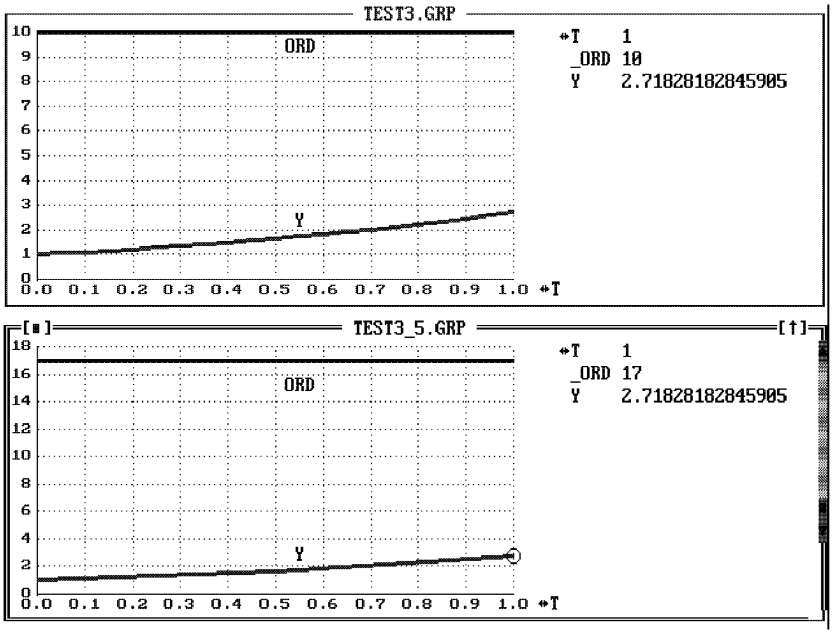

Figure 4.1:

Note : The analytical solution of (20) is

$$y = e^t. \tag{23}$$

In Fig.4.1 the solution of (20) is shown. Fig.4.1 consists of two parts. The part labelled TEST3 shows a calculation of the differential equation (20) for integration step h=0.1s.

The part labelled TEST5 shows a computation with an integration step h=0.5s. As expected, the value of ORD has automatically increased as h grows. The value of y at time T=1s is of course the same in both parts of Fig.4.1.

Results illustrating the use of the Taylor series for applying a numerical integration method are shown in Tab.5. Tab.5 demonstrates the results of a numerical solution of the differential equation (20) after one computation step (with the integration step h=1s).

In each line of Tab.5 the "Reduced value y(1)" of the numerical solution of the differential equation (20) and the corresponding value of the "Absolute error" - for the method order ORD used are printed.

In the column "Absolute error" we can read the difference between the exact and numerical solutions. For the exact solution of the equation (20) for $t_1 = h = 1s$, in view of the equation (23), we have

$$y(1) = e^1 = 2.718281828459045235.....$$

Reduced value y(1)	ORD	Absolute error
2.	1	-0.718281828459045235
2.	2	-0.218281828459045235
2.	3	-0.051615161792378683
2.7	4	-0.009948495125712053
2.71	5	-0.001615161792378750
2.718	6	-0.000226272903489866
2.7182	7	-0.000027860205077168
2.7182	8	-0.000003058417775609
2.718281	9	-0.000000302885853176
2.7182818	10	-0.000000027312660911
2.71828182	11	-0.000000002260552523
2.718281828	12	-0.000000000172876824
2.7182818284	13	-0.000000000012286394
2.71828182845	14	-0.000000000000815681
2.71828182845	15	-0.000000000000050959
2.71828182845904	16	-0.000000000000003220
2.71828182845904	17	-0.000000000000000444
2.71828182845904	18	-0.000000000000000333
2.71828182845904	19	-0.000000000000000333
2.71828182845904	20	-0.000000000000000333

Tab.5: Long real arithmetic

In order to make the results of the numerical solution more clear and illustrative only the digits tallying the digits of the exact solution are shown.

It follows from Tab.5 that the requirement of a higher method order is justifiable - with the same integration step h a higher method order (i. e. with more terms of the Taylor series) can yield a higher accuracy (it approximates better the exact solution).

However, the increase in the accuracy of the result is not unlimited. In Tab.5 the accuracy stops increasing when the ORD reaches the value of 18. This is caused by an underflow during the computation of the higher order Taylor series terms. The addition of these terms changes neither the resulting value of y_{n+1} nor the absolute error - the absolute error has reached its saturated value ESAT (the value of ESAT depends on the word width of the arithmetic of the computer used).

4.1.1 Accuracy and Word Width

The accuracy of the result can be influenced in a considerable way by the word width. For instance when the word width is 32 bits and the integration step h=1s, only the accuracy of computation of the equation (20) shown in Tab.6 can be reached.

Reduced value y(1)	ORD	Absolute error
2.	1	-7.183E-0001
2.	2	-2.183E-0001
2.7	3	-5.162E-0002
2.71	4	-9.948E-0003
2.718	5	-1.615E-0003
2.7182	6	-2.263E-0004
2.7182	7	-2.785E-0005
2.718281	8	-3.053E-0006
2.718281	9	-2.980E-0007
2.7182818	10	-2.421E-0008
2.71828182	11	1.863E-0009
2.71828182	12	1.863E-0009
2.71828182	13	1.863E-0009

Tab.6: 32 bit arithmetic

With a specially constructed 128-bit arithmetic a very high computation accuracy for the equation (20) can be reached even with the integration step h=1s (Tab.7).

Given an integration step h, it is very important

- for the absolute error to decrease with an increasing word width,

- for the absolute error of the solution to decrease with an increasing method order ORD.

Reduced value y(1)	ORD	Abs err
2.	1	-7.183E-0001
2.	2	-2.183E-0001
2.7	3	-5.162E-0002
2.71	4	-9.948E-0003
2.718	5	-1.615E-0003
2.7182	6	-2.663E-0004
2.7182	7	-2.786E-0005
2.718281	8	-3.059E-0006
2.7182818	9	-3.029E-0007
2.71828182	10	-2.731E-0008
2.718281828	11	-2.261E-0009
2.7182818284	12	-1.729E-0010
2.71828182845	13	-1.229E-0011
2.71828182845	14	-8.155E-0013
2.71828182845904	15	-5.077E-0014
2.718281828459045	16	-2.976E-0015
2.7182818284590452	17	-1.648E-0016
2.71828182845904523	18	-8.652E-0018
2.7182818284590452353	19	-4.315E-0019
2.7182818284590452353	20	-2.050E-0020
2.7182818284590452353602	21	-9.300E-0022
2.71828182845904523536028	22	-4.036E-0023
2.7182818284590452353602874	23	-1.679E-0024
2.7182818284590452353602874	24	-6.704E-0026
2.718281828459045235360287471	25	-2.575E-0027
2.7182818284590452353602874713	26	-9.523E-0029
2.718281828459045235360287471352	27	-3.397E-0030
2.7182818284590452353602874713526	28	-1.170E-0031
2.718281828459045235360287471352662	29	-3.896E-0033
2.71828182845904523536028747135266249	30	-1.255E-0034
2.718281828459045235360287471352662497	31	-3.910E-0036
2.7182818284590452353602874713526624977	32	-1.102E-0037
2.7182818284590452353602874713526624977	33	4.408E-0039
2.7182818284590452353602874713526624977	34	4.408E-0039
2.7182818284590452353602874713526624977	35	4.408E-0039

Tab.7: 128 bit arithmetic

4.1.2 Evaluation of the Computation Speed

Further important results can be achieved by experimental calculations in solving numerically the differential equation (20) with an explicit use of the Taylor series.

The exact solution $y(1) = e^1$ can also be approximated by solving the equation (20) with the integration step $h = 0.5s$ (Tab.8 - two computation steps are necessary in this case) or with the integration step $h = 0.1s$ (Tab.9 - ten computation steps are necessary).

Only those digits (in the result of the numerical solution of (20)) are printed which tally the exact value.

Reduced value y(1)	ORD	Absolute error
2.	1	-0.468281828459045310
2.	2	-0.077656828459045312
2.71	3	-0.009514467347934263
2.7182	4	-0.000935637052795313
2.7182	5	-0.000077008038038451
2.718281	6	-0.000005449497788135
2.7182818	7	-0.000000338137442601
2.71828182	8	-0.000000018677261404
2.7182818284	9	-0.000000000929473165
2.71828182845	10	-0.000000000042083559
2.71828182845	11	-0.000000000001747935
2.71828182845904	12	-0.000000000000067057
2.718281828459045	13	-0.000000000000002442
2.718281828459045	14	-0.000000000000000222
2.718281828459045	15	-0.000000000000000222
2.718281828459045	16	-0.000000000000000222

Tab.8: Long real arithmetic

Reduced value y(1)	ORD	Absolute error
2.	1	-0.124539368359045440
2.71	2	-0.004200981850820962
2.718	3	-0.000104565977435356
2.7182	4	-0.000002084323879603
2.718281	5	-0.000000034655339265
2.71828182	6	-0.000000000494185248
2.71828182845	7	-0.000000000006168954
2.71828182845	8	-0.000000000000068612
2.71828182845904	9	-0.000000000000000555
2.718281828459045	10	0.000000000000000222
2.718281828459045	11	0.000000000000000222
2.718281828459045	12	0.000000000000000222

Tab.9: Long real arithmetic

The results in Tab.8 and Tab.9 are again characterized by the saturated absolute errors ESAT (the computation is terminated when three consecutive absolute error values have not changed). However, the saturated absolute error ESAT is reached using a lower ORD (as compared with Tab.5). This is due to the fact that with the same type of arithmetic higher order Taylor series terms will have no effect when the integration step h is shortened. Up from a certain method order a shorter integration step h causes an underflow so that the adding of further Taylor series terms does not change the result.

Similarly, the results for the integration step h= 0.01s are in Tab.10 and the results for the integration step h= 0.001s are shown in Tab.11.

Reduced value y(1)	ORD	Absolute error
2.7	1	-0.013467999037519162
2.7182	2	-0.000044965899087201
2.718281	3	-0.000000112359411108
2.718281828	4	-0.000000000224644081
2.71828182845	5	-0.000000000000374145
2.71828182845904	6	-0.000000000000001332
2.71828182845904	7	-0.000000000000001332
2.71828182845904	8	-0.000000000000001332

Tab.10: Long real arithmetic

Reduced value y(1)	ORD	Absolute error
2.71	1	-0.001357896223154187
2.718281	2	-0.000000452707280774
2.718281828	3	-0.000000000113170251
2.7182818284590	4	-0.000000000000022982
2.7182818284590	5	-0.000000000000022982
2.7182818284590	6	-0.000000000000022982

Tab.11: Long real arithmetic

It follows from Tab.5, Tab.8 to Tab.11 that the saturated absolute error ESAT is reached with different numbers of Taylor series terms.

A different question is, of course, with what speed can the result be obtained if a shortened integration step h is used (with the corresponding saturated absolute error ESAT).

This speed evidently depends on the technical construction and on the number of operations required. For evaluation of the computation speed the equation (22) was rewritten into the form

$$y_{n+1} = y_n + DY1_n + DY2_n + ... + DYP_n, \tag{24}$$

where

$DY1_n = h * y_n$

$DY2_n = \frac{h}{2} * DY1_n$

.

.

$DYP_n = \frac{h}{P} * DY(P-1)_n.$

The relation between the number of operations (required for reaching the saturated absolute error ESAT) and the integration step h is given in Tab.12.

h	ORD	NRCS	Addition	Multipl.	DIV
1	18	1	18	18	17
0.5	14	2	28	28	26
0.1	10	10	100	100	90
0.01	6	100	600	600	500
0.001	4	1000	4000	4000	3000

Tab.12

In the column "DIV" ve have the number of division operations, in the column "Multipl." we have the number of product operations and in the column "Addition" the number of addition operations necessary for computing the results according to Tab.5, Tab.8, Tab.9, Tab.10 and Tab.11. In the column "ORD" in Tab.12 there is always the value of the method order with which the absolute error (according to Tab.5, Tab.8, Tab.9, Tab.10, Tab.11) reached its saturated value ESAT.

In the column "NRCS" , to complete the picture, the number of computation steps is shown after which, given the integration step h, the point $t_n = 1s$ has been reached (i. e. the point at which the exact solution is $y_n = e^1$).

Regardless of the practical construction of addition, multiplication and division it is clear from Tab.12 that the number of operations (required for reaching the highest accuracy - with the corresponding integration step h) increases as the integration step h shortens.

Shortening the integration step h does not mean, however, only an increase in the number of operations. It is also characterized by an existence of the accumulated error - the error from one step is carried to the following steps.

This fact can be clearly demonstrated by studying the absolute errors in Tab.5, Tab.8 to Tab.11. It is obvious that the optimal integration step h producing the least saturated absolute error ESAT of the computation exists.

It is certainly very advantageous to do the computation with the optimal integration step h if we want to reach a high accuracy. Using the same arithmetic, the accuracy reached with the integration step h = 1s is by two orders better than with the integration step h = 0.001s.

Futhermore, a very important conclusion can be drawn from Tab.12 - the number of operations (addition, division, and multiplication) with the integration step h = 1s is less than the corresponding number of operations with the integration step h = 0.001s.

Briefly, it means that the computation is done most precisely and at the same time most quickly with the optimal integration step h (and the corresponding optimal method order). A computation done with other than optimal integration step h is always slower and less accurate.

4.1.3 Experimental Time Evaluations

The following tables (Tab.13,Tab.14,Tab.15) are of great importance.

Tab.13 is the same as Tab.5 (with "Reduced value y(1)" and "ORD") but brings time evaluation of the computation. For instance, using the 17th method order requires 0.983 ms.

Reduced value y(1)	ORD	Time (ms)
2.	1	0.084
2.	2	0.140
2.	3	0.195
2.7	4	0.248
2.71	5	0.307
2.718	6	0.365
2.7182	7	0.422
2.7182	8	0.468
2.718281	9	0.531
2.7182818	10	0.589
2.71828182	11	0.649
2.718281828	12	0.693
2.7182818284	13	0.757
2.71828182845	14	0.828
2.71828182845	15	0.861
2.71828182845904	16	0.911
2.71828182845904	17	0.983
2.71828182845904	18	1.033

Tab.13

If we wanted to reach the same accuracy by the 4th order Runge-Kutta method, we would have to use a substantially shorter integration step and the computation time would be 271.229 ms (Tab.14).

h(s)	Reduced value y(1)	Time (ms)
1	2.7	0.299
0.1	2.7182	2.691
0.01	2.718281828	27.500
0.001	2.71828182845904	271.229

Tab.14

Tab.15 shows that it is possible to calculate the differential equation (20) with the integration step as great as 400s, which requires the use of 575 Taylor series terms. The computation takes 31.999 ms.

h(s)	ORD	Time (ms)
1	18	1.033
10	43	2.504
20	64	3.560
30	82	4.642
40	99	5.551
50	115	6.487
100	200	11.282
400	575	31.999

Tab.15

Note: All time evaluations were obtained on the ACA 32000 computer (based on National Semiconductor 32000 processor).

4.2 Homogenous Equation - Example 2

Let us solve the differential equation

$$x' = -x, \qquad x(0) = 1. \tag{25}$$

The numerical solution of (25) by the Taylor series method is

$$x_{n+1} = x_n * (1 - h + \frac{h^2}{2!} - \frac{h^3}{3!} + \ldots). \tag{26}$$

Note : The exact solution of (25) is

$$z = e^{-t}. \tag{27}$$

In combination with (20) the equation (25) can be called a "check function". The reason for this is that for the product of the analytical solution of equations (20) and (25) we have

$$z = y * x = e^t * e^{-t} = 1 \tag{28}$$

and thus we can use (28) for testing the accuracy of the numerical solution. The system of equations (20),(25),(28) has been selected in such a way that the high accuracy of the computation is again shown. The function z is constantly equal to 1 (Fig.4.2).

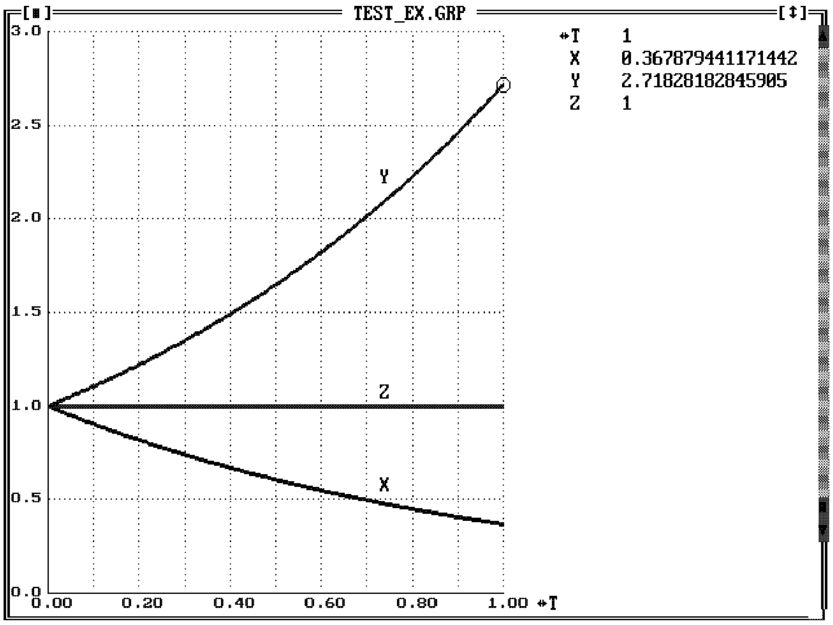

Figure 4.2:

The computation in Fig.4.2 starts at time point zero and terminates at time t=1s. Corresponding values of the variables at time point t=1s are shown in the upper right-hand corner of the Fig.4.2.

Note:

Similarly, a check function of a homogenous equation

$y'' + y = 0 \quad y(0) = 0, \quad y'(0) = 1$

(or equivalent system

$y' = x \quad y(0) = 0,$

$x' = -y \quad x(0) = 1)$

can be written in a form

$$z = x^2 + y^2 = 1.$$

Time functions of x, y and ERR ($ERR = z - 1$) are in Fig.4.3.

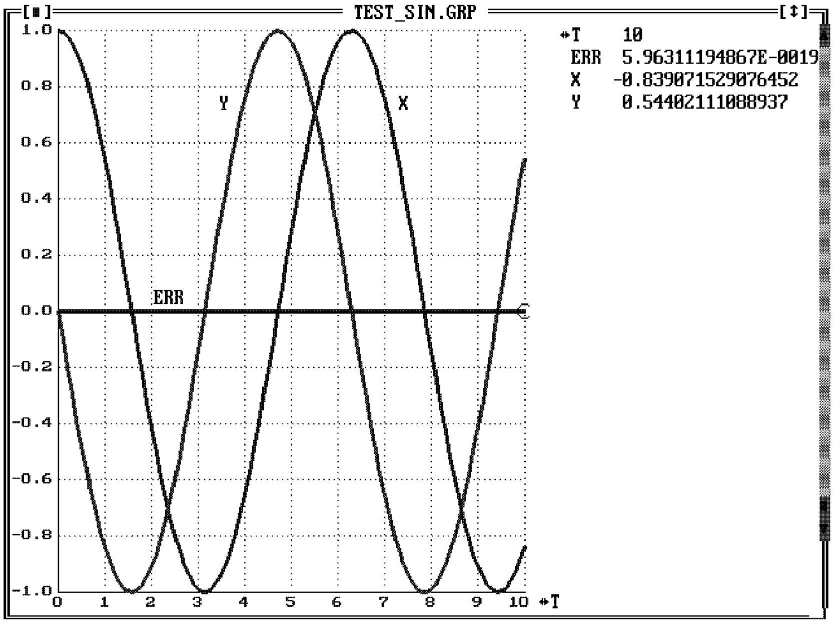

Figure 4.3:

Conclusion: The solution of homogenous differential equations is one of the most important applications of the Modern Taylor Series Method.

Chapter 5

VAN-DER-POL'S EQUATION

As an application of the Modern Taylor Series Method the solution of the well known Van-der-Pol's equation

$$\frac{d^2y}{dt^2} + \mu(y^2 - 1)\frac{dy}{dt} + y = 0$$

is described.

Let

$$\frac{dy}{dt} = y_1.$$

Then

$$\frac{dy_1}{dt} = -y - \mu(y^2 - 1)y_1.$$

The corresponding source text in TKSL/386 is

```
var
  y,y1;
const
  mi = 3, p1 = 0, p2 = 1,
  tmax = 10, eps = 1E-20;
system
  y' = y1                          & p1;
  y1'= mi*y1*(1-y*y)-y             & p2;
sysend.
```

Time functions ORD, y and y1 are in Fig.5.1. The aim of Fig.5.1 is to point out two things. First, the values of ORD are high and second, these values vary considerably during the computation.

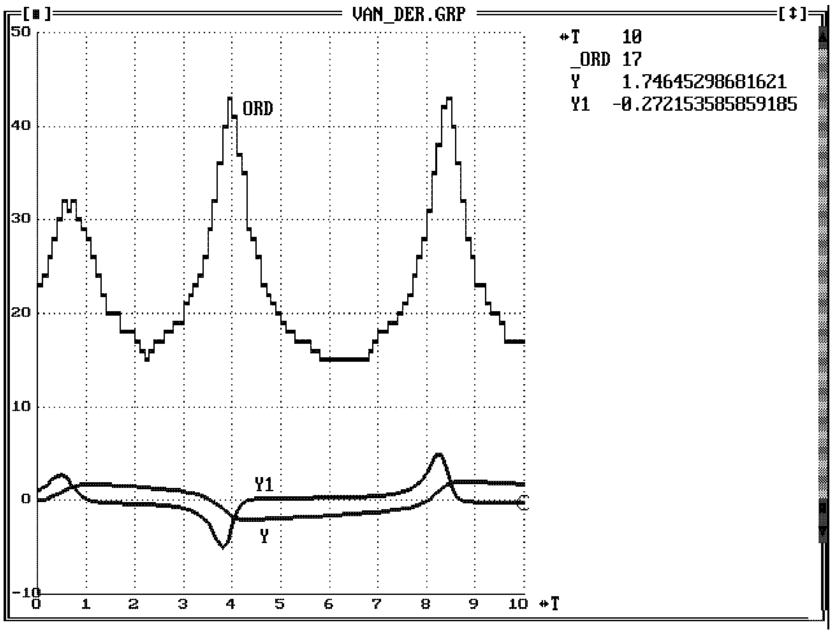

Figure 5.1:

Fig.5.2 shows the solution in a better scale. The solution of the Van-der-Pol equation is shown using phase-plain in Fig.5.3.

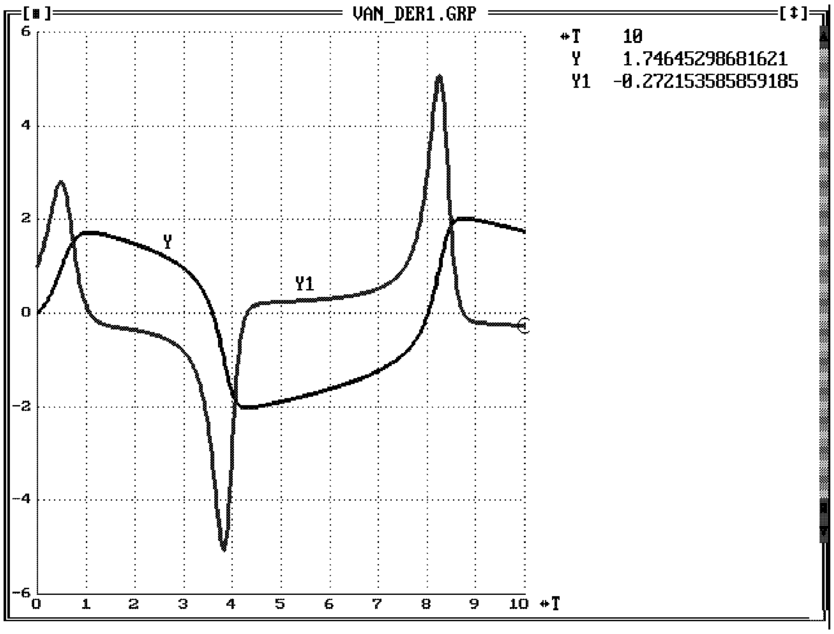

Figure 5.2:

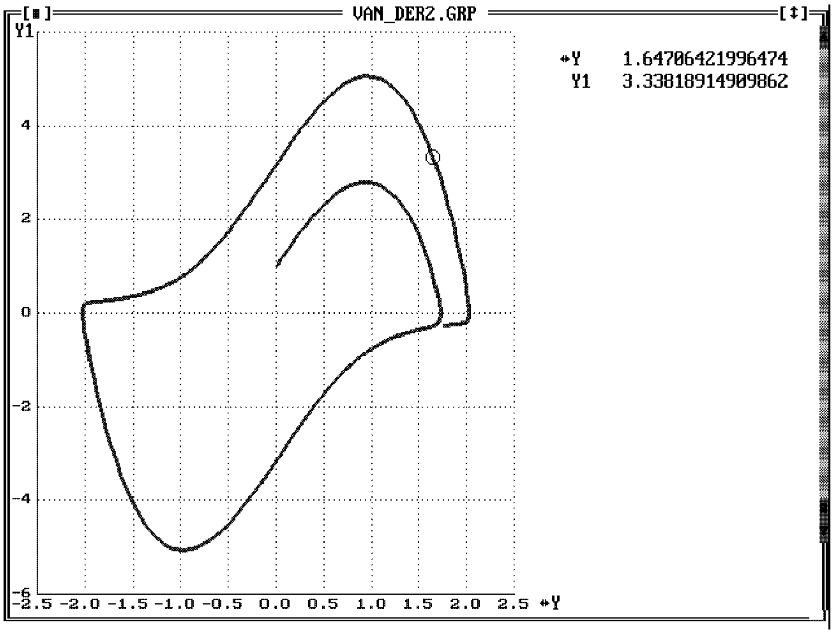

Figure 5.3:

Chapter 6

TRAJECTORY OF AN ELECTRON

When analyzing the trajectory of an electron we use the equation for the force $\vec{F}$ acting on an electron in an electromagnetic and electrostatic field

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}).$$

By substituting

$$\vec{F} = m\vec{a},$$

defining a particular electromagnetic and electrostatic field (and the direction of the velocity vector)

$$B_x = B_y = 0, \qquad E_z = 0, \qquad v_z = 0,$$

decomposing into the directions of the x and y axes

$$F_x = q(E_x + v_y B_z)$$

$$F_y = q(E_y - v_x B_z),$$

or

$$m\frac{d^2x}{dt^2} = q(E_x + \frac{dy}{dt}B_z)$$

$$m\frac{d^2y}{dt^2} = q(E_y - \frac{dx}{dt}B_z)$$

and supposing

$\frac{q}{m} = -1.76 * 10^{11}[m^2V^{-1}s^{-2}]$

we get the corresponding source text in TKSL/386 :

```
var
  x,y,vx,vy,ex,ey;
const
  tmax=8e-11,
  dt=1e-12,
  eps=1e-20,
  bz=0.5;
system
  ex=0;
  ey=0;
  x'=vx &0;
  y'=vy &0;
  vx'=-1.76e11*(ex-vy*bz) &-8e7;
  vy'=-1.76e11*(ey+vx*bz)  &0;
sysend.
```

Fig.6.1 supports the well-known fact that an electron moves in a circle in a magnetic field (B_z=0.5T, E_x=E_y=0, $v_x(0)$= -8 $* 10^7 ms^{-1}$).

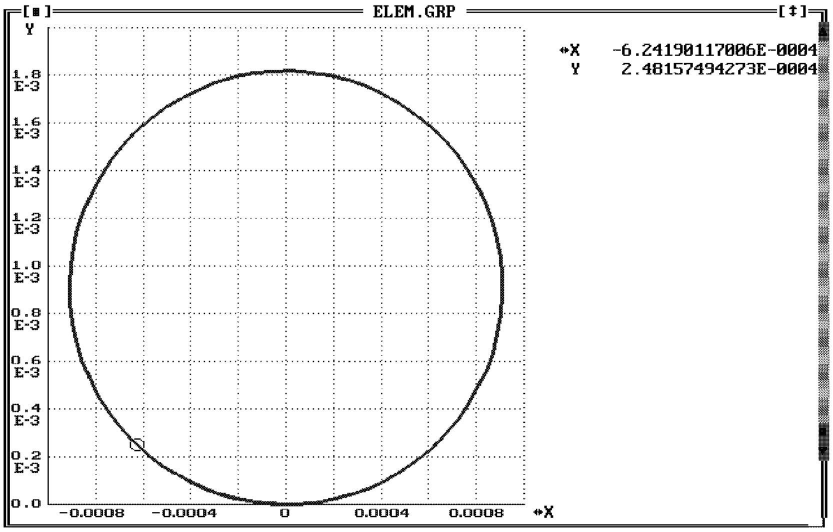

Figure 6.1:

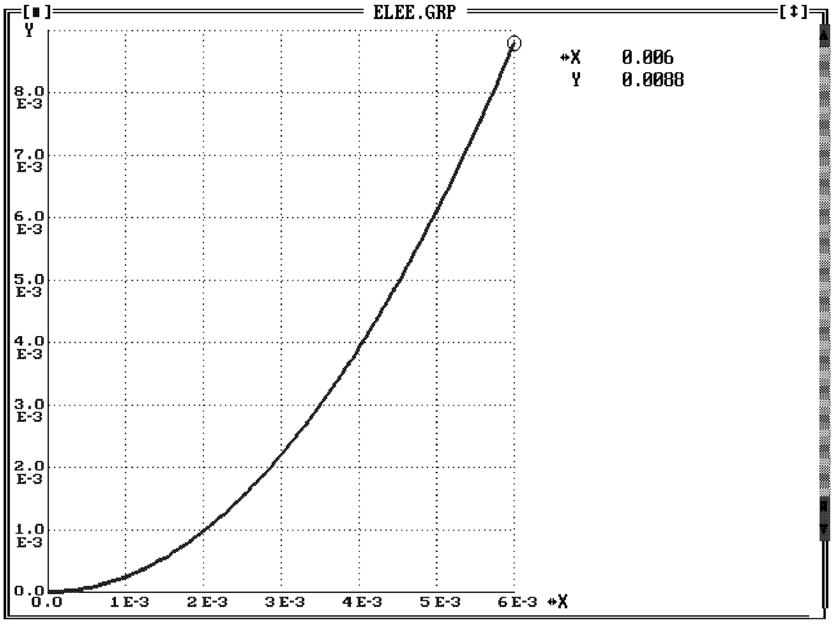

Figure 6.2:

Fig.6.2, in turn, shows that in an electrostatic field an electron moves along a parabola ($E_y = -10^7 Vm^{-1}, E_x = 0, B_z = 0, v_x(0) = 6 * 10^7 ms^{-1}$).

In a constant electrostatic ($E_x = 0, E_y = 10^7 Vm^{-1}$) and at the same time in a constant electromagnetic field ($B_z = 0.5T$), an electron follows the trajectory shown in Fig.6.3.

In the part denoted by ELEME the initial velocity of the electron is $v_x(0) = 6*10^7 ms^{-1}$, in the part denoted by ELEME1 the velocity is $v_x(0) = -6*10^7 ms^{-1}$ (the velocity vector has the opposite direction).

In a constant magnetic field ($B_z = 0.5T$) where the electrostatic field is at the same time defined by the expressions

$$E_x = -10^7 \cdot sin 5 \cdot 10^9 t [Vm^{-1}]$$

$$E_y = -10^7 \cdot cos 5 \cdot 10^9 t [Vm^{-1}]$$

the obtained trajectory is shown in Fig.6.4 ($v_x(0) = -3 * 10^7 ms^{-1}$).

For completeness' sake, the divergencies in the directions of x and y axis from Fig.6.4 as functions of time are shown in Fig.6.5.

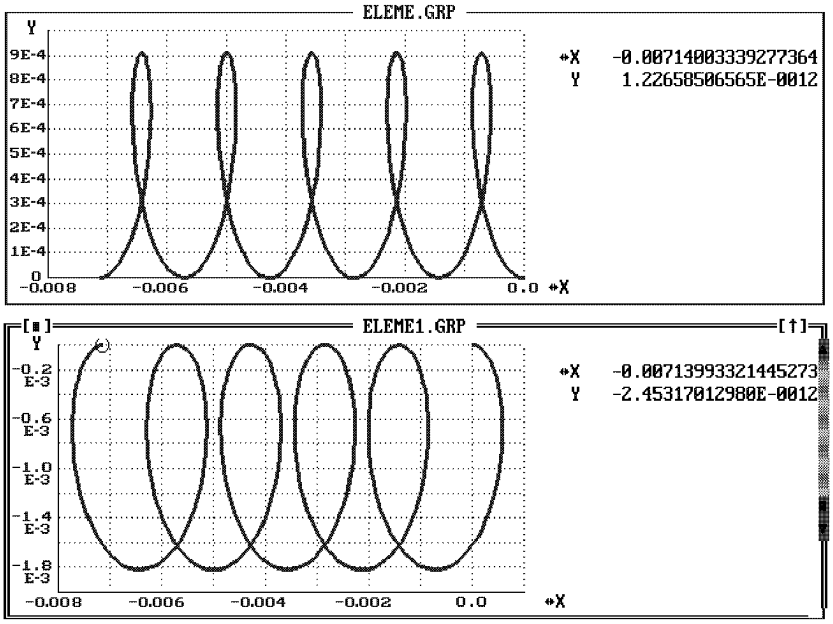

Figure 6.3:

The aim of this chapter is to point out that physical problems can be transformed to solving differential equations. As in the previous chapters, the Modern Taylor Series Method can also be used.

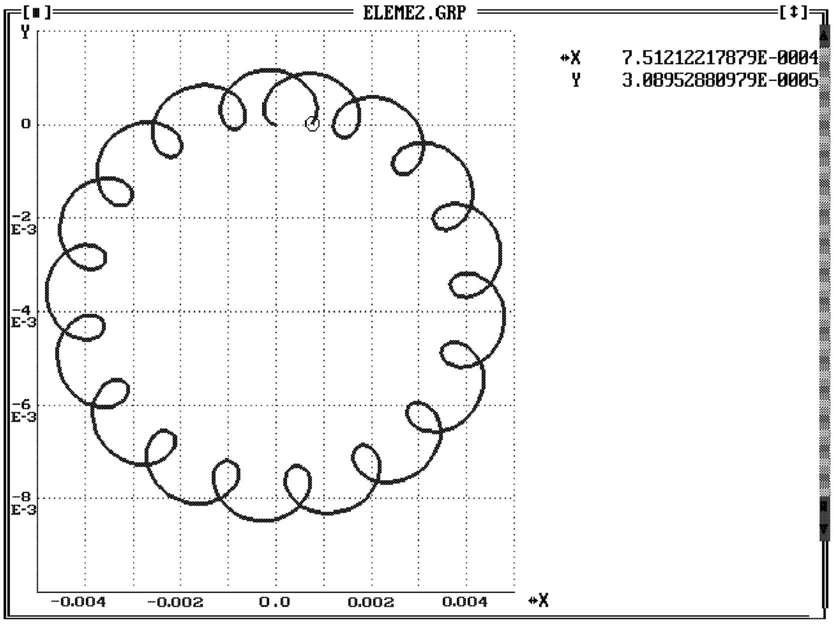

Figure 6.4:

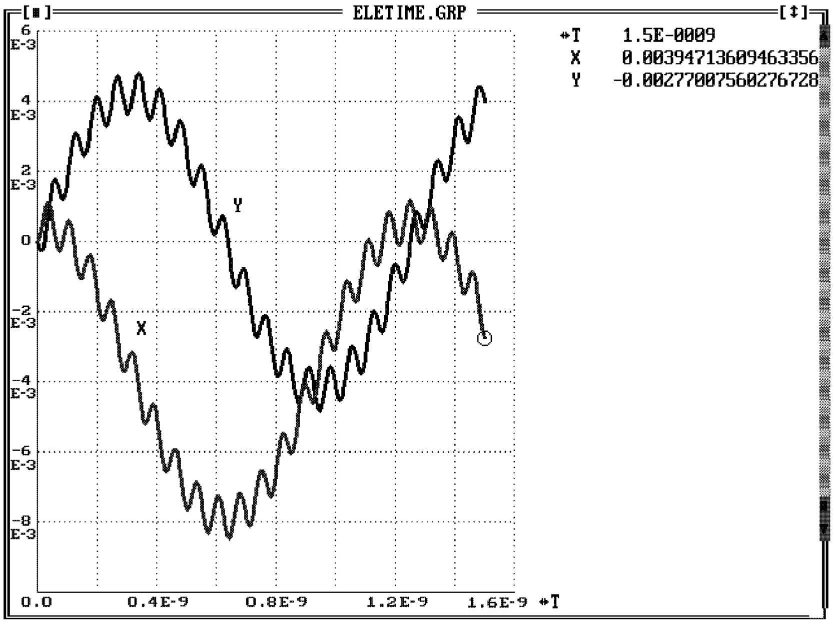

Figure 6.5:

Chapter 7

ELECTRICAL CIRCUITS

Solving electrical circuits is a typical problem leading to solving differential equations and the Modern Taylor Series Method can again be used. We start from Kirchhoff's laws. A simple example follows.

For a serial circuit RLC connected to a voltage source u we have

$u_L + u_R + u_C = u$

or

$L\frac{di}{dt} + Ri + \frac{1}{C}\int idt = u.$

By substituting

$y' = i$

we have

$i' = \frac{1}{L}(u - Ri - \frac{1}{C}y).$

Time functions u_R, u_L, u_C for $u = 1V$, $R = 1000\Omega$, $L = 1H$, $C = 10^{-6}F$ are shown in Fig.7.1.

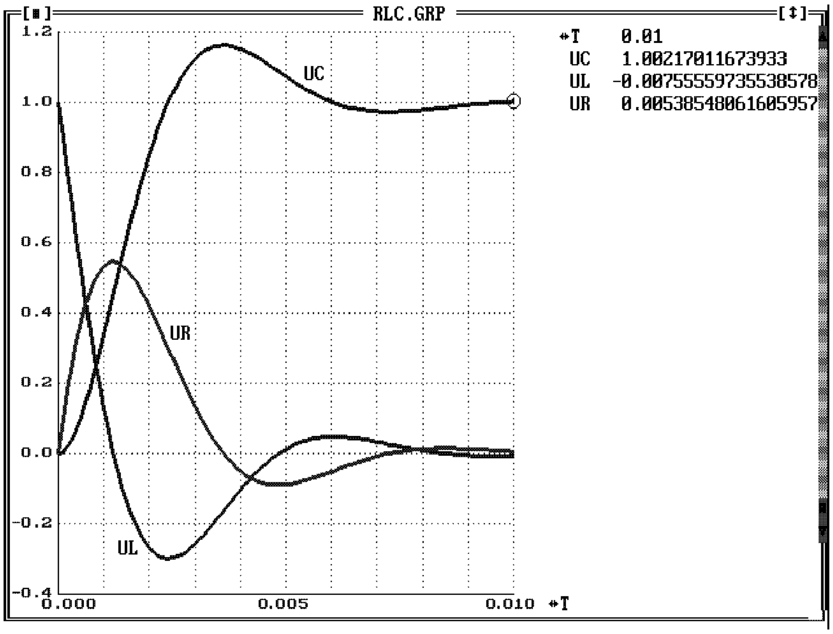

Figure 7.1:

The corresponding source text in TKSL/386 is

```
var i,y,u,UR,UL,UC;
const  R=1000,L=1,C=1E-6,tmax=0.01,dt=0.0001;
system
u=1;
i'=1/L*(u-R*i-1/C*y)    &0;
y'=i                    &0;
UR=R*i;
UC=1/C*y;
UL=u-R*i-1/C*y;
sysend.
```

Fig.7.2 (the part labelled LVAR) shows the current I1=i and voltage $V1 = u_C$ of the serial circuit RLC ($R = 48.5\Omega, L_0 = 0.054H, C = 200*10^{-6}F$) as functions of time (for u=0, $V1(0) = u_C(0) = 500V$) if the time function of inductivity is defined as

$$L = L_0 + 1000t^2[H].$$

For completness' sake the part labelled LVAR1 shows time functions I1N=i and $V1N = u_c$ for $L = L_0$.

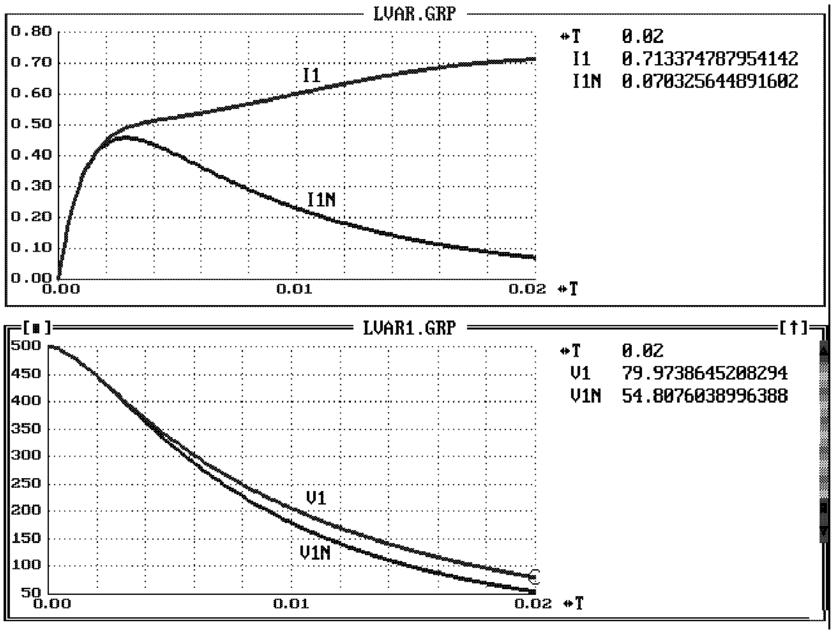

Figure 7.2:

Chapter 8

MECHANICAL SYSTEMS

Analyzing mechanical systems is another typical application of a numerical method of solution of differential equations. This time we start from Newton's kinetic laws. Similarly as in the electron case the force $\vec{F}$ is substituted by the expression

$$\vec{F} = m\vec{a}.$$

As an example, the coupled mass-spring-damper can be described by

$$m_1 \frac{d^2 y_1}{dt^2} = -k_1 y_1 + k_2(y_2 - y_1) + C_2(\frac{dy_2}{dt} - \frac{dy_1}{dt} - C_1 \frac{dx_1}{dt})$$

$$m_2 \frac{d^2 y_2}{dt^2} = -k_2(y_2 - y_1) - C_2(\frac{dy_2}{dt} - \frac{dy_1}{dt}).$$

Let $\quad \frac{dy_1}{dt} = v_1, \qquad \frac{dy_2}{dt} = v_2.$

For particular parameters we have the following source text in TKSL/386

```
var v1,v2,y1,y2;
const tmax=2, dt=0.01;
system
v1'=-10*v1+5*v2-1250*y1+1000*y2  &0;
v2'=5*v1-5*v2+1000*y1-1000*y2    &0;
y1'=v1   &0;
y2'=v2   &-0.45;
sysend.
```

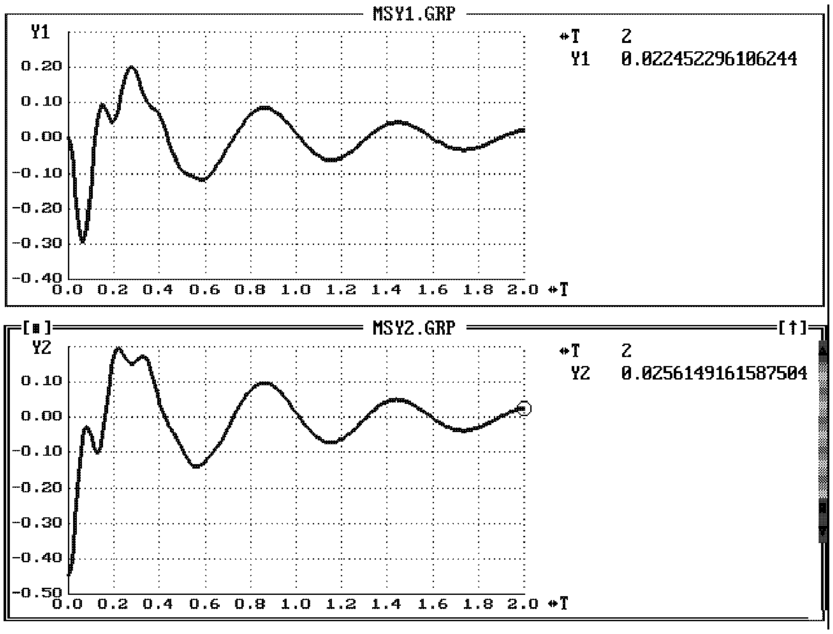

Figure 8.1:

Fig.8.1 shows time functions of deviations Y1 and Y2 of the system. The dead condition is Y1=0, Y2=0. Here we analyse a case when the mass m_2 was deflected in the initial state to a position Y2(0)=-0.45m.

Chapter 9

PHYSICAL PENDULUM

A simple physical pendulum whose analysis is a typical demonstration example in world simulation languages (ACSL, EASY5x, HYBSIS,...) obeys the equation

$$z_2'' = -z_2' + k_1 sin z_2.$$

The physical pendulum can be described (for particular parameters) by the following source text (in TKSL/386):

```
var
  z1,z2;
const
  k1 = -31.415;
const
  tmax = 5, dt=0.01;
system
  z1'=-z1+k1*sin(z2)          & 13;
  z2'=z1                      & 0;
sysend.
```

Time functions of the angular velocity Z2 and the deviation Z1 of a physical pendulum (for the case when the pendulum in its motion has not reached the top dead point) are in Fig.9.1 (in the part labelled PEND). In the part labelled PEND1 the corresponding x-y plot is shown.

In Fig.9.2 (in the part PEND2) the time functions are plotted (for the case when the pendulum in its motion has passed the top dead point). In the part labelled PEND3 the corresponding x-y plot is shown.

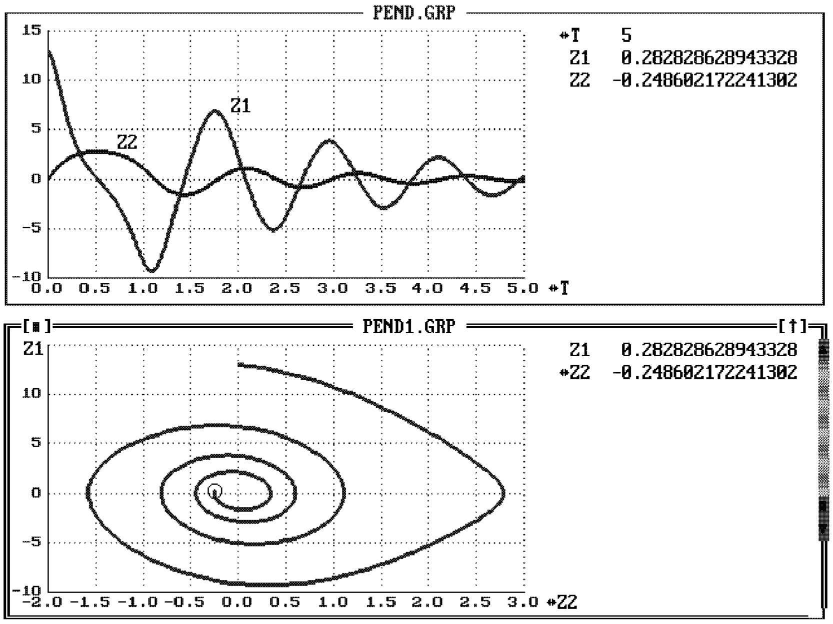

Figure 9.1:

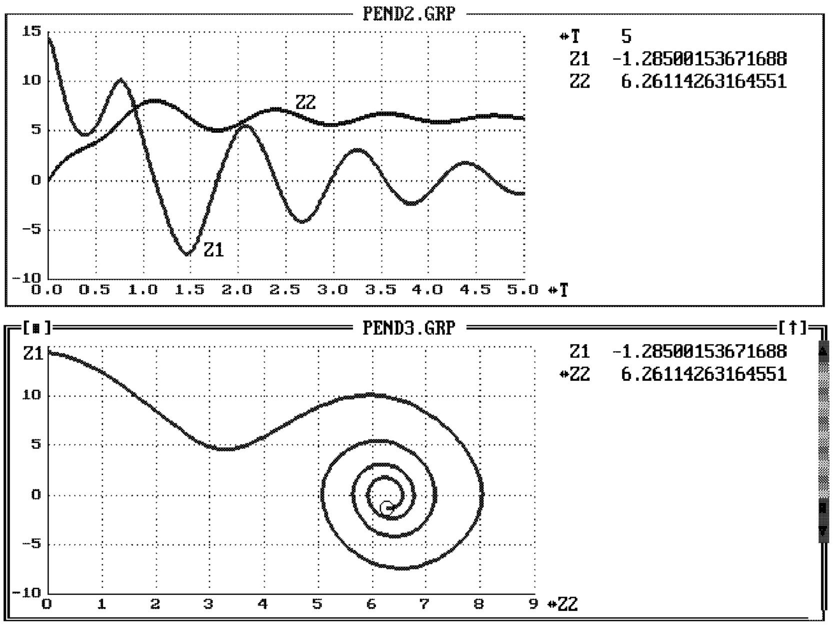

Figure 9.2:

Chapter 10

DISCONTINUITIES

A demonstration of applying the Modern Taylor Series Method to solving problems with discontinuities is in Fig.10.1.

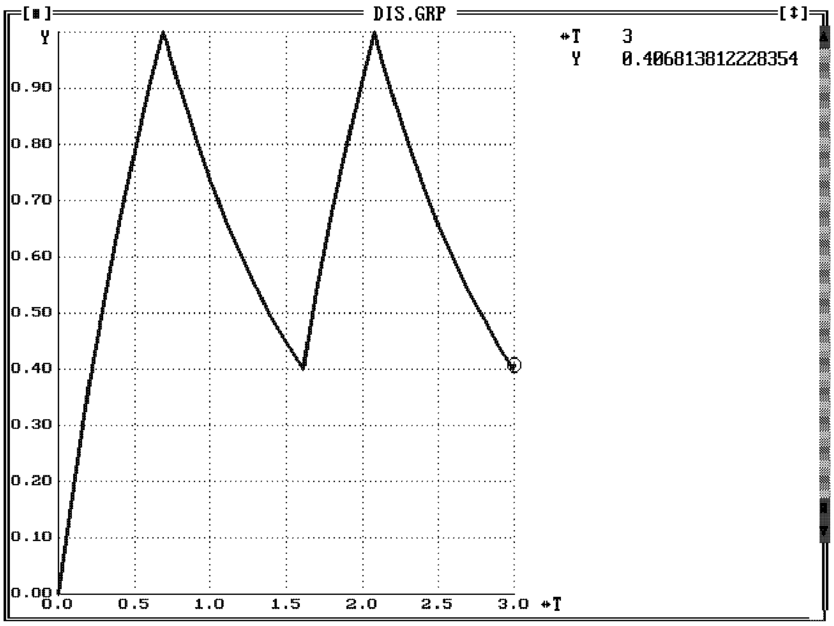

Figure 10.1:

Here two separated differential equations

$$y' = 2 - y \tag{29}$$
$$y' = -y \tag{30}$$

are in fact solved according to which condition is fulfilled. The calculation starts at the initial condition y(0)=0. In the initial state the equation (29) is solved. After the level=1 is passed, the system is switched to a new state and the equation (30) is solved. The point of discontinuity is determined by bisecting the integration step.

The corresponding source text in TKSL/386 is:

```
var y;
const level=1,tmax=3,eps=1e-18;
system
y'=2-y  &0;
case y of
>level:level=0.4;  y'=-y    &1;
else   level=1;    y'=2-y   &0.4;
esac;
sysend.
```

To solve differential equations with discontinuties the TKSL/386 uses the CASE construction.

Thus the absolute value is solved in the following way:

```
system
x=sin(t);
case x of
>0: y=sin(t);
else y=-sin(t);
esac;
sysend.
```

Dead zone uses the CASE construction:

```
x=sin(t);
case x of
>0.5: y=x-0.5;
>-0.5:y=0;
else y=x+0.5;
esac;
```

Step function uses the CASE construction:

```
x=sin(t);
case x of
>0.5: y=1;
>-0.5:y=0;
else y=-1;
esac;
```

Relay uses the CASE construction:

```
x=2*sin(t);
case x of
<level: y=0; level=1;
else y=1; level=0;
esac;
```

Saturation uses the CASE construction:

```
x=sin(t);
case x of
>0.5: y=0.5;
>-0.5:y=x;
else y=-0.5;
esac;
```

Signum uses the CASE construction:

```
x=sin(t);
case x of
>0: y=1;
else y=-1;
esac;
```

Fig.10.2 and Fig.10.3 show a test of typical nonlinear functions. The part labelled ABS shows the absolute value , the part labelled DZON shows the dead zone , the part labelled SAT shows the saturation, the part labelled Q shows the step function, the part labelled REL shows a relay with hysteresis and the part labelled SIG1 shows the signum function.

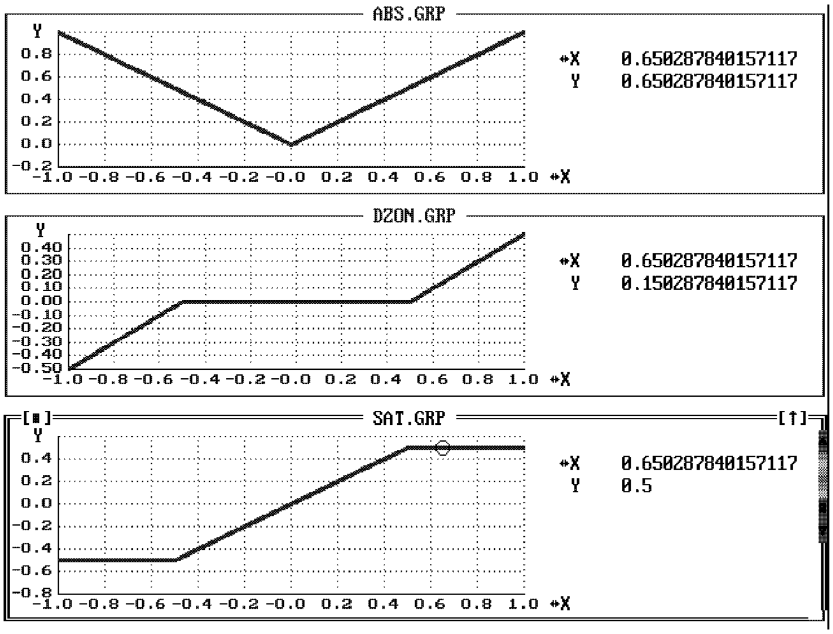

Figure 10.2:

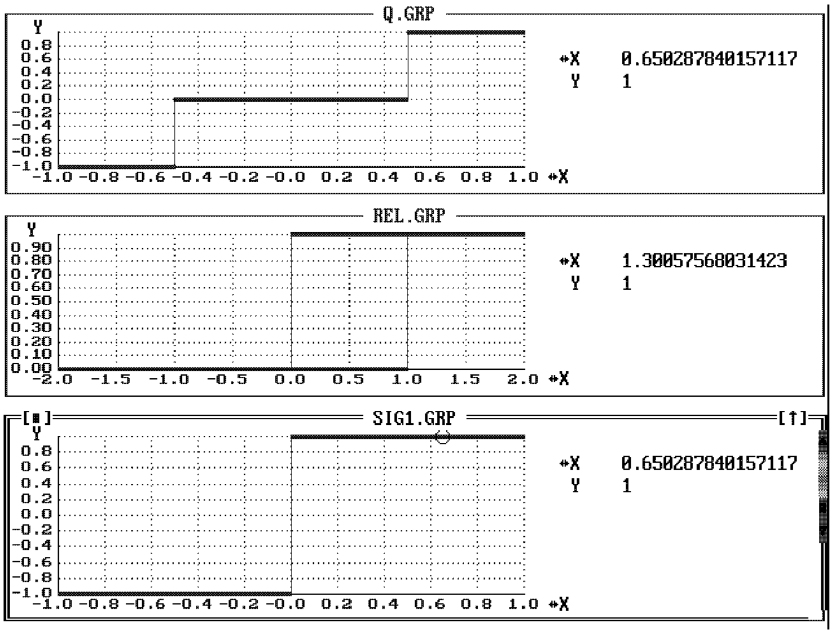

Figure 10.3:

Chapter 11

STIFF SYSTEMS

In many areas of science and technology the problems of the so called stiff systems are encountered. By a stiff system of differential equations we understand a system with considerably different time constants (i.e. a system with considerably different eigenvalues of the corresponding Jacobi matrix). A stiff and stable system has the following properties

$$Re\lambda_j < 0,$$

$$max \mid Re\lambda_j \mid >> min \mid Re\lambda_i \mid,$$

where λ_j denotes an eigenvalue of the Jacobi matrix of the system and $Re\lambda_j$ its real part.

The stability and exactness are two basic problems connected with numerical solutions of stiff systems. A number of one-step, multiple-step, explicit and implicit methods have been developed that are more or less suitable for solving stiff systems. Mostly they are modifications of well-known numerical methods. It is the aim of this chapter to evaluate the possibilities of the stiff systems being solved by the Modern Taylor Series Method.

11.0.1 Example 1

Let us focus our attention on particular problems with the integration of stiff problems. Let us consider a system of linear differential equations

```
y1'=y2,               y1(0)= 1                                    (31)

y2'=-1000y1-1001y2,   y2(0)=-1                                    (32)
```

with the exact solution

$y_1 = e^{-t}, \qquad y_2 = -e^{-t}.$

The general solution of the system (31),(32) is in the form

$y_i = A_i e^{-t} + B_i e^{-1000t}$

$i = 1, 2 \qquad A_i, B_i \qquad$ are constants

and the eigenvalues of the matrix of the system are

$\lambda_1 = -1, \qquad \lambda_2 = -1000.$

The corresponding time functions Y1 and Y2 of the system (31),(32) are in Fig.11.1.

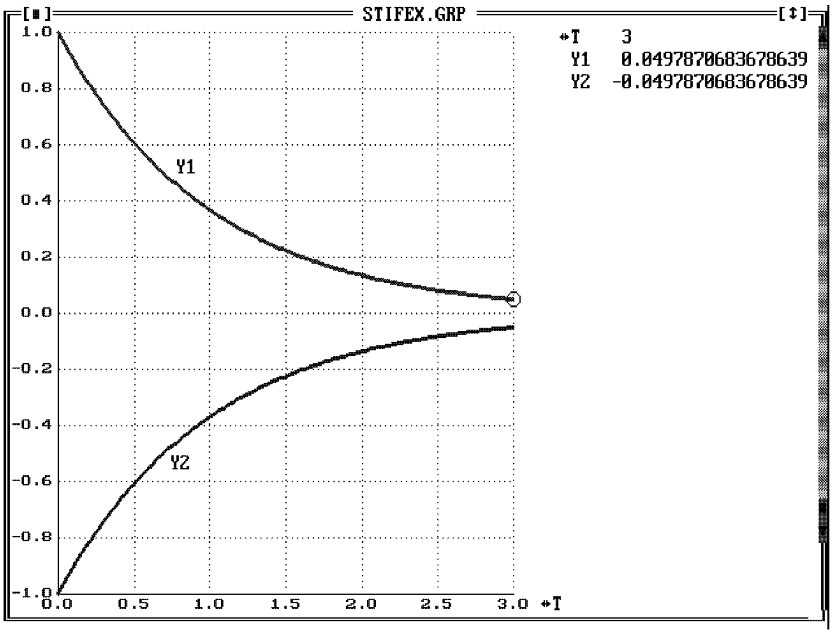

Figure 11.1:

Fig.11.2 illustrates problems of stiff systems. The part labelled STIFEX1 shows the solution and the function ORD for the system

```
y1'=y2,                  y1(0)= 1                    (33)

y2'=-100y1-101y2,        y2(0)=-1.                   (34)
```

This is a case when the difference of eigenvalues is small.

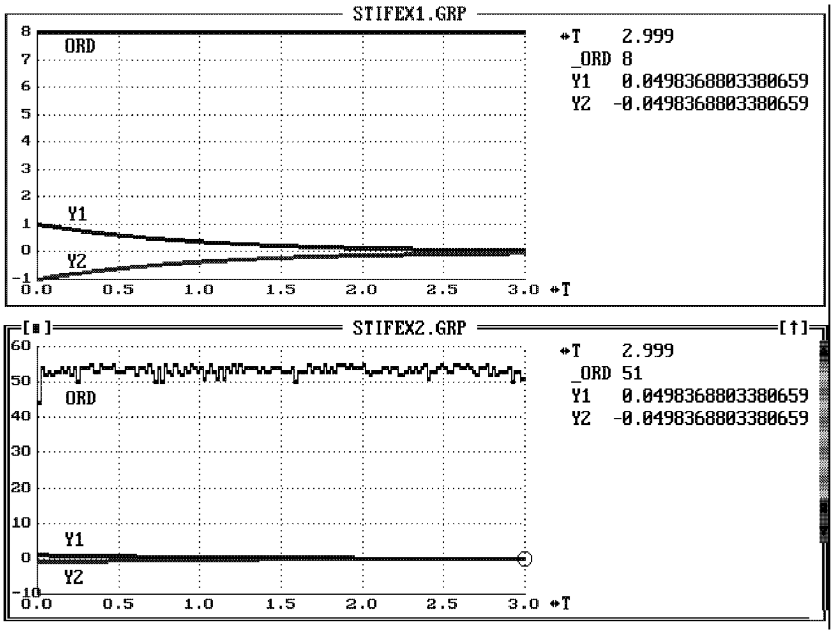

Figure 11.2:

If the difference of eigenvalues of the system is large (system (31),(32)), there is a significant increase in the value of ORD (in the part labelled STIFEX2).

11.0.2 Example 2

The situation is similar when solving the next system:

```
y1'=-0.0001*y1-499.9999*y2,           y1(0)=2         (35)
y2'=          -500*y2,                y2(0)=1         (36)
y3'=          19500*y2    -20000*y3,  y3(0)=2         (37)
```

It is typical of stiff systems that their solutions have to be found on long intervals.

Even though the solution components Y2,Y3 at time T=0.099999s are negligible, the computation must still be carried out with a small integration step and a high method order ORD (Fig.11.3). A detail of the origin of a solution of the system (35),(36),(37) is illustrated in the Fig.11.4 (Tmax=0.001s for the part labelled STIF_L1 and Tmax=0.1s for the part labelled STIF_L2).

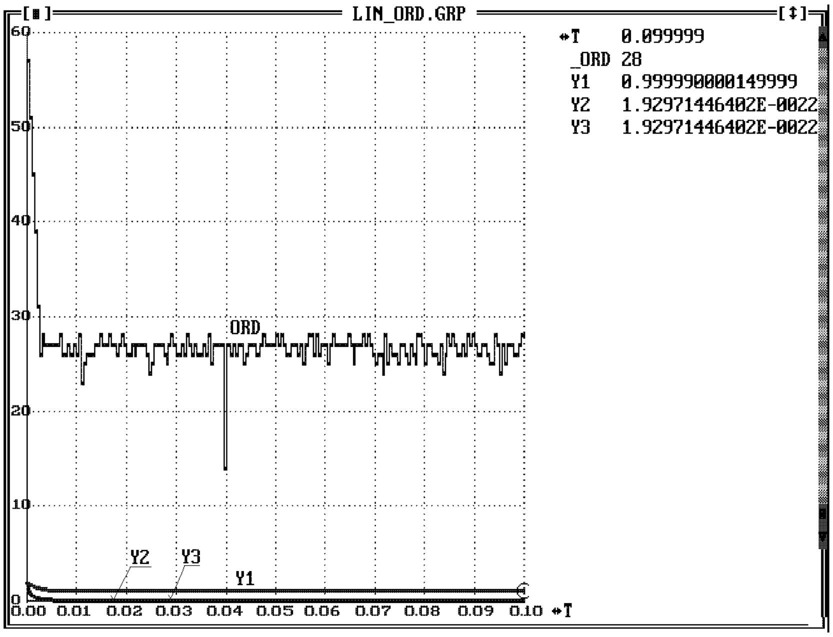

Figure 11.3:

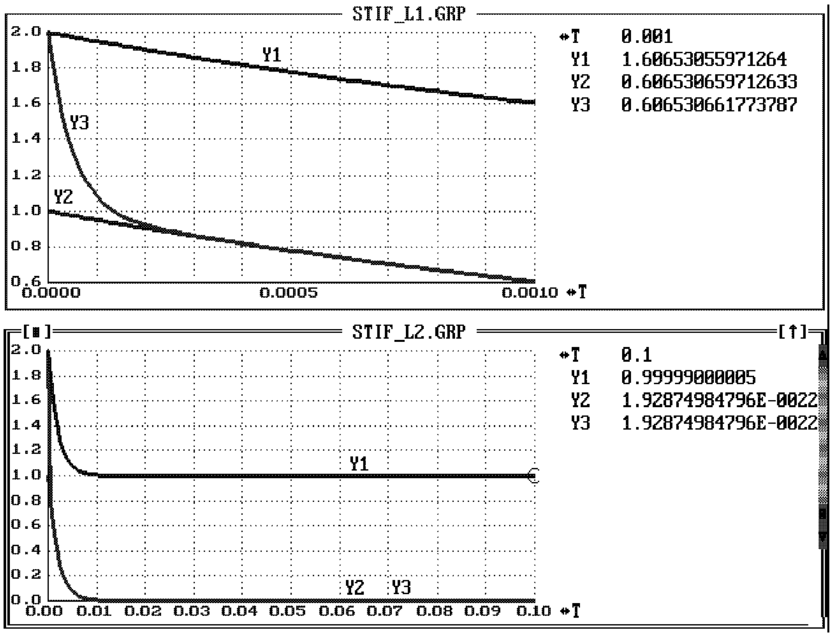

Figure 11.4:

11.0.3 Example 3

The following example of a stiff system tests the ability of the numerical method to handle discontinuities. The problem is as follows

$$y_1' = c_1 * (y_2 + c_2 - y_1)$$

$$y_2' = c_3 * (c_4 - y_2).$$

The task is to find the position of discontinuities for $t \epsilon < 0,\ 5s >$.
Parameters c_1 and c_3 remain unchanged during the simulation:
$c_1 = 2.7 * 10^6, c_3 = 3.5651205$.

The system operates in two states:

- $c_2 = 0.4$ and $c_4 = 5.5$ when the system is in state 1 ($y_1(0) = 4.2$ and $y_2(0) = 0.3$). The system remains in state 1 as long as $y_1 < 5.8$.
- When the system switches to state 2, parameters c_2 and c_4 change to $c_2 = -0.3$ and $c_4 = 2.73$. The system remains in state 2 as long as $y_1 > 2.5$. When passing this instance the system switches back to state 1.

The corresponding source text in TKSL/386 is:

```
var    y1,y2;
const  level=5.8,tmax=5,eps=1e-18,
       c2=0.4,c4=5.5;
system y1'=2.7e6*(y2+c2-y1) &4.2;
       y2'=3.5651205*(c4-y2) &0.3;
       case y1 of
       >level: level=2.5;
               c2=-0.3;c4=2.73; { first task}
        { c2=1.3;c4=4.33;..second task ) }
       else level=5.8;
               c2=0.4;c4=5.5;
       esac;
sysend.
```

The time function of the solution Y1 (first task) is plotted in Fig.11.5 - in the part labelled DIS_ST1.

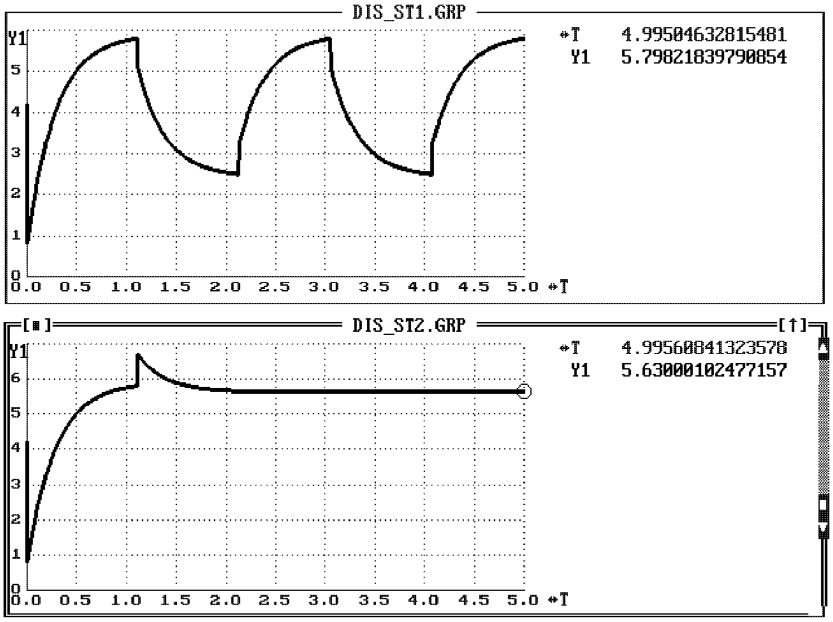

Figure 11.5:

The positions of each of the five discontinuities obtained from solving the first task are in Table 16. The results of the computation shown in the table are of course produced with the full accuracy, as it is characteristic of TKSL.

1.1083061677721695
2.12968535515772928
3.0541529070076332
4.07553209439319275
4,99999964623928239

Tab.16

The time function of the solution Y1 (second task - when we change the state 2 parameter values to $c_2 = 1.3, c_4 = 4.33$) is plotted in Fig.11.5 - in the part labelled DIS_ST2.

11.0.4 Example 4

The following nonlinear stiff system has also been solved by the TKSL/386:

$$y_1' = -0.04 * y_1 + 10000 * y_2 * y_3 \qquad y_1(0) = 1 \tag{38}$$
$$y_2' = 0.04 * y_1 - 10000 * y_2 * y_3 - 30000000 * y_2 \qquad y_2(0) = 0 \tag{39}$$
$$y_3' = 30000000 * y_2 * y_2 \qquad y_3(0) = 0 \tag{40}$$

For a nonlinear system of differential equations the eigenvalues are given by the Jacobi matrix J . In nonlinear systems of differential equations the eigenvalues of the matrix J depend on the time t and they change during the integration. The system above describes fast chemical reactions.

The results are in Fig.11.6. It is typical of the system (38),(39),(40) that $SUMA = 1$ in the entire time interval ($SUMA = y_1 + y_2 + y_3$).

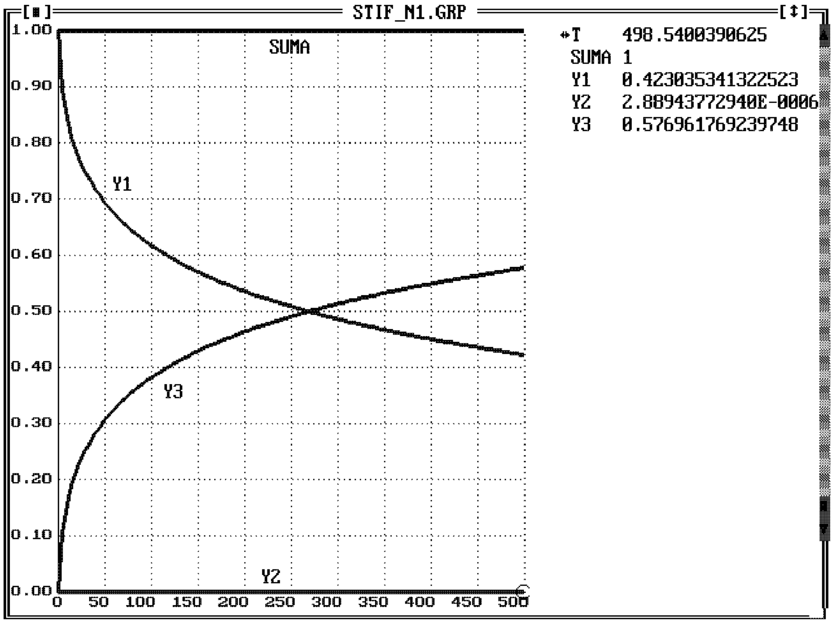

Figure 11.6:

11.0.5 Example 5

The following example shows that a special class of stiff systems can suitably be approximated by a time delay.

Let us analyze the system

$$y' = -ay + az, \qquad y(0) = 0;$$
$$z = sint.$$

The corresponding source text in TKSL/386 (for particular values of parameters) is

```
var y,z,w;
const a=5,tmax=10,eps=1e-20;
system
y'=-a*y+a*z  &0;
z=sin(t);
sysend.
```

The time delay of the solution Y with respect to the input function z can be seen clearly in Fig.11.7 for $a = 5$ and in Fig.11.8 (in the part labelled STIFSIN2 for $a = 8$ and in the part labelled STIFSIN2 for $a = 100$).

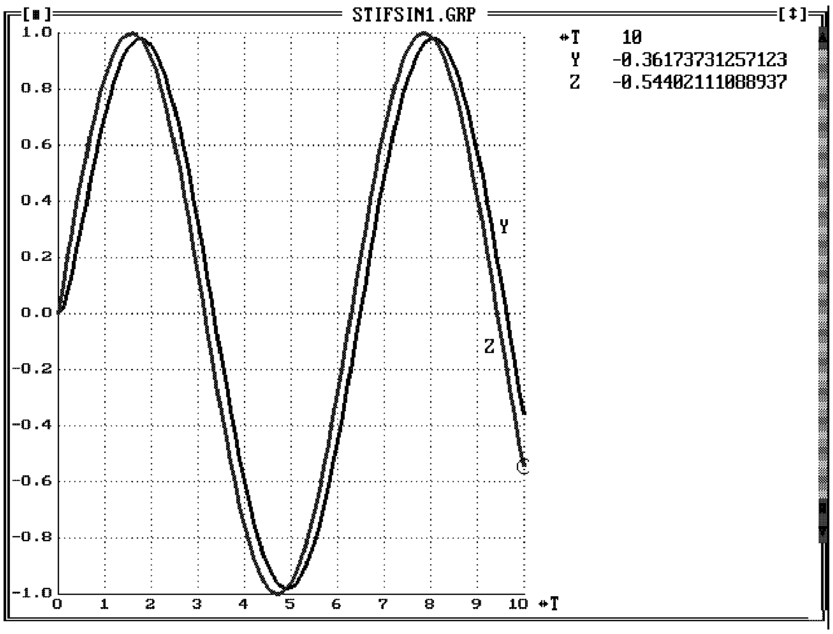

Figure 11.7:

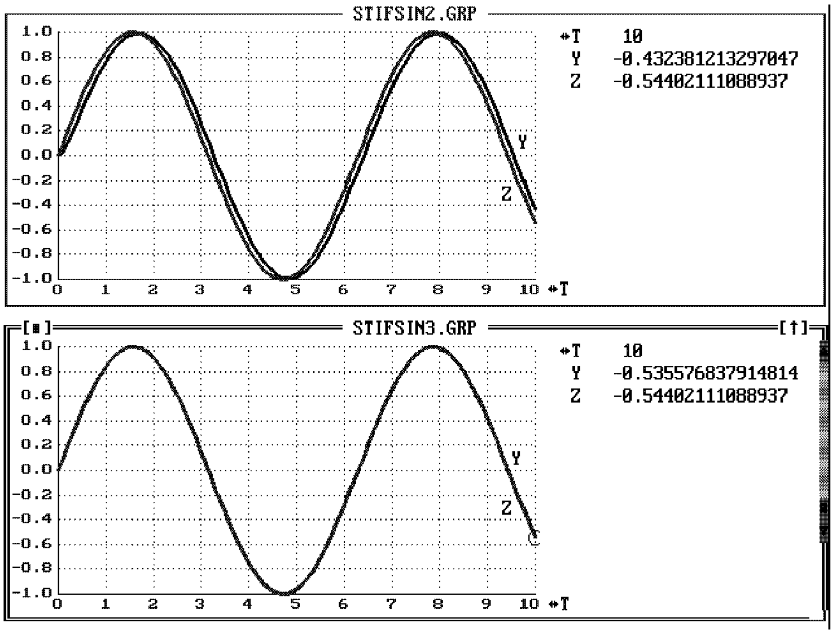

Figure 11.8:

Let us define the error ERR as the difference between the numerical solution y and the input function with time delay

$$ERR = y - sin(t - \frac{1}{a}).$$

The time function of the absolute error ERR (resulting from approximating the solution by the time delay) is shown in Fig.11.9. The part labelled APPSI1 shows the function ERR for a=100, the part labelled APPSI2 for a=1000.
It is clear that for $a \to \infty$ the absolute value of ERR in a stable state converges to 0.

Note:

The time delay of the input function z (Fig.11.10) can be produced by the Taylor series expansion or by the so called Pade expansion.

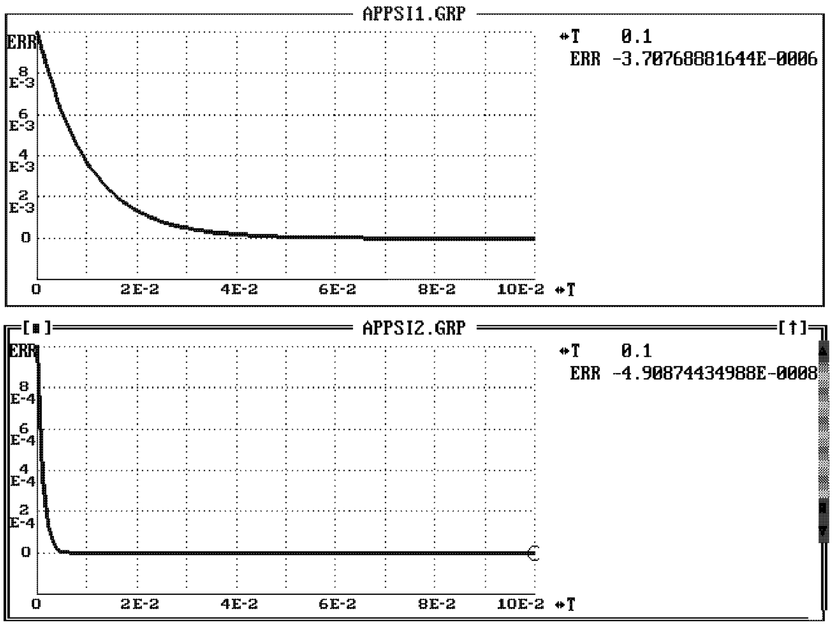

Figure 11.9:

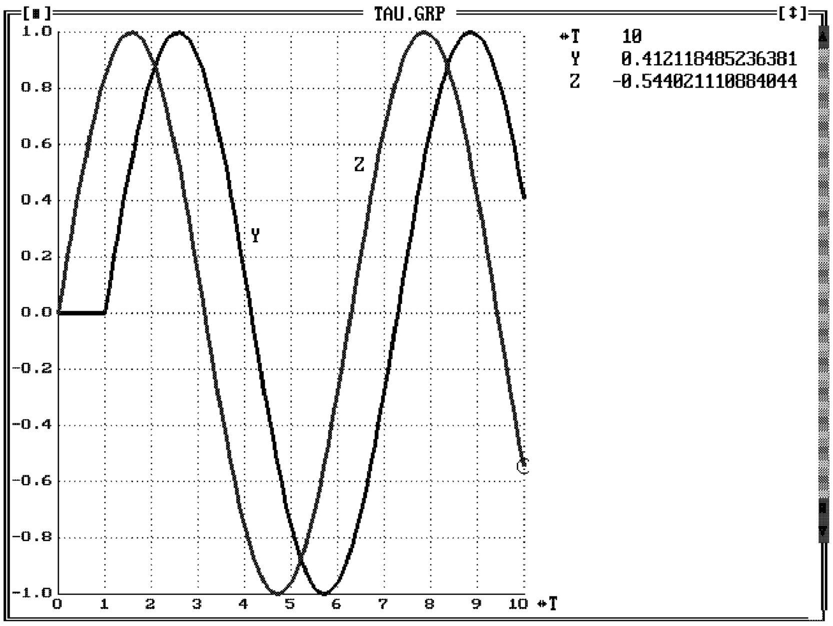

Figure 11.10:

Chapter 12

DEFINITE INTEGRALS AND INTEGRAL EQUATIONS

Definite integrals and integral equations, due to the number of applications, are very important mathematical tools. Their solution using the Modern Taylor Series Method is the subject of this chapter.

The problem of solving a definite one-dimensional integral taken as a function of the upper boundary can be transformed to solving a system of differential equations.

Let a definite integral

$$y = \int_{0}^{9.853} e^{2sint} dt \tag{41}$$

be given.

The definite integral can be rewritten in the form:

$$y' = z$$

$$z = e^{2sint}.$$

The initial conditions can be obtained by substituting the value of the lower boundary for the variable t of the function z.

The numerical solution of the integral is obtained at the point corresponding to the upper boundary of the integral ($t_{max} = 9.853s$).

The corresponding source text in TKSL/386 is

```
var y,z;
const    tmax=9.853,
          eps=1e-20;
system
y'=z   &0;
z=exp(2*sin(t));
sysend.
```

The time functions of the value of the integral y and the function z being integrated are in Fig.12.1. A particular calculation of the integral for T=9.853s is in the right-hand part of Fig.12.1.

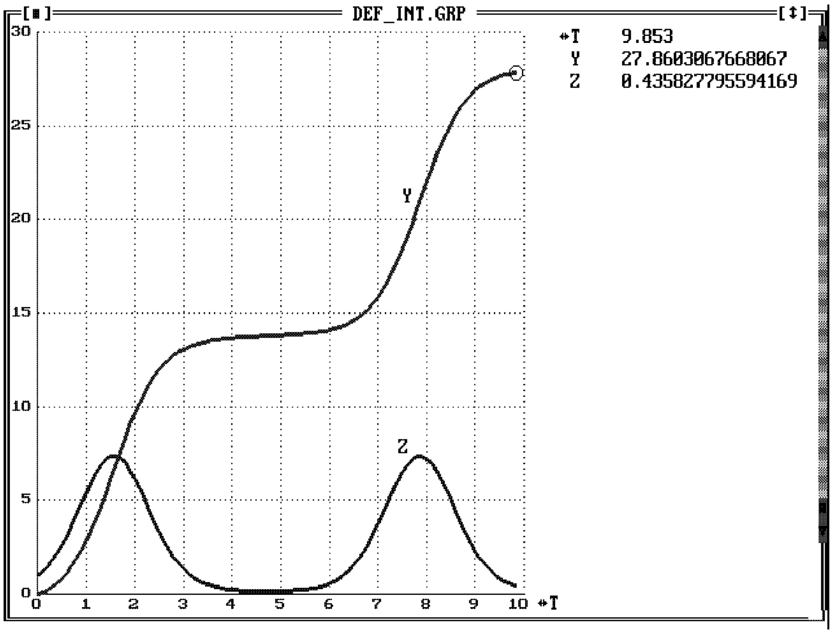

Figure 12.1:

Applying the Modern Taylor Series Method to solving the Volterra integral equation, we get a following computation scheme.

Let an integral equation

$$y(t) = sint + \int_0^t y(s)ds \tag{42}$$

be given.

The integral equation can be rewritten in the form

$$y' = cost + y. \tag{43}$$

Note:

The exact solution of (43) is

$$y = \frac{(e^t + sint - cost)}{2}.$$

The source text in TKSL/386 is

```
var y,x;
const tmax=10,eps=1e-20,dt=0.1;
system
y'=cos(t)+y    &0;
x=(exp(t)+sin(t)-cos(t))*1/2;
sysend.
```

The corresponding time functions of X and Y are shown in Fig.12.2. X is the exact solution. Y is the numerical solution of (43). It is very illustrative to compare both results.

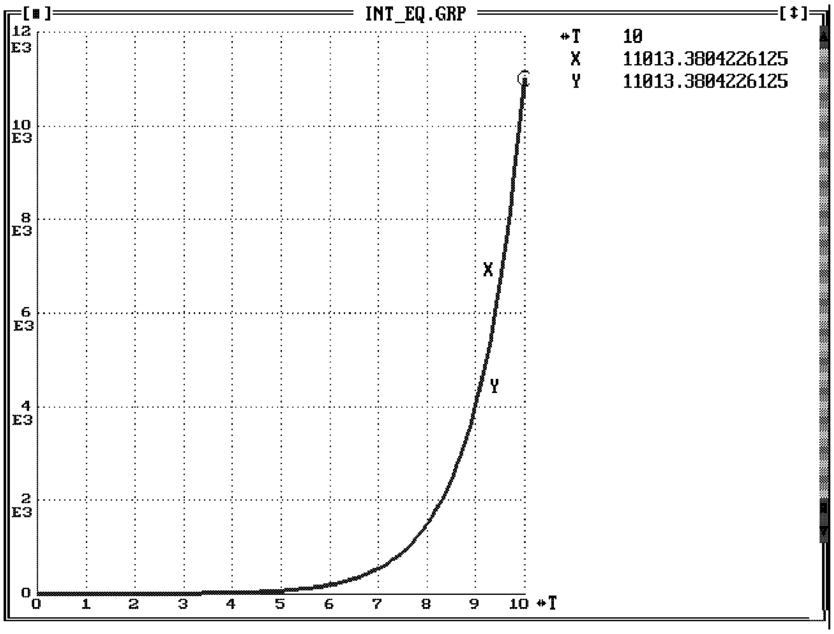

Figure 12.2:

Chapter 13

SYSTEMS OF LINEAR ALGEBRAIC EQUATIONS

Systems of linear algebraic equations (SLAE) can also be solved by the Modern Taylor Series Method. In this case the system of linear algebraic equations

$$\mathbf{Ax} - \mathbf{b} = \mathbf{0} \tag{44}$$

must be transformed to a system of differential equations (SDE)

$$\mathbf{Ax} - \mathbf{b} = -\mathbf{x}'. \tag{45}$$

Supposing that the real parts of roots of the characteristic equation

$$\mid \mathbf{A} + \lambda\mathbf{I} \mid = 0 \tag{46}$$

where $\mathbf{I}$ is the unit matrix, are negative, the derivatives on the right-hand side of the system (45) will be equal to zero in a stable state and the solution of the SLAE will be identical with the solution of the system of differential equations (SDE).

Since not every matrix $\mathbf{A}$ satisfies the condition (46), sometimes the system of differential equations (44) has to be transformed to a stable system. One of the ways to do this is to multiply the whole system of algebraic equations by transposed matrix $\mathbf{A}^T$ from left, so that the actual system to be solved is

$$\mathbf{A}^T\mathbf{Ax} - \mathbf{A}^T\mathbf{b} = -\mathbf{x}'. \tag{47}$$

If the matrix $\mathbf{A}$ is non-singular, which is a general condition for a SLAE to have a solution, the resulting matrix $\mathbf{A}^T\mathbf{A}$ is positively definite.

The matrix **A** is real, thus $\mathbf{A}^T\mathbf{A}$ is positively stable and so is the system (47). A transformation performed by multiplying **A** by the transposed matrix $\mathbf{A}^T$ has a special property - the resulting functions resemble a strong attenuation.

The time function of the solution of SLAE can only take on one of the forms shown in Fig.13.1.

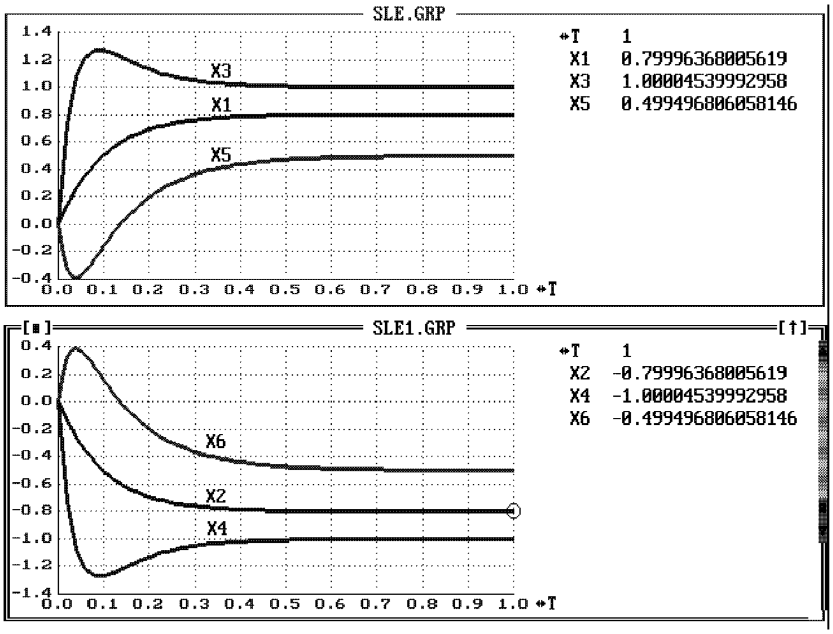

Figure 13.1:

As a demonstration Fig.13.2 shows a numerical solution of the Example SLAE1 (including ORD - in the part labelled SLAE_ORD) and, in a better scale, in the part labelled SLAE.

Example SLAE1:

System of linear algebraic equations

```
  x1 - 2*x2 = -1
-5*x1 + 3*x2 =  4
```

has been transformed to

$x1' = -26 * x1 + 17 * x2 - 21$

$x2' = 17 * x1 - 13 * x2 + 14.$

Note: The computation process finds automatically the roots of the given system of linear algebraic equations. The roots are found when

$x_1' = 0,$

$x_2' = 0.$

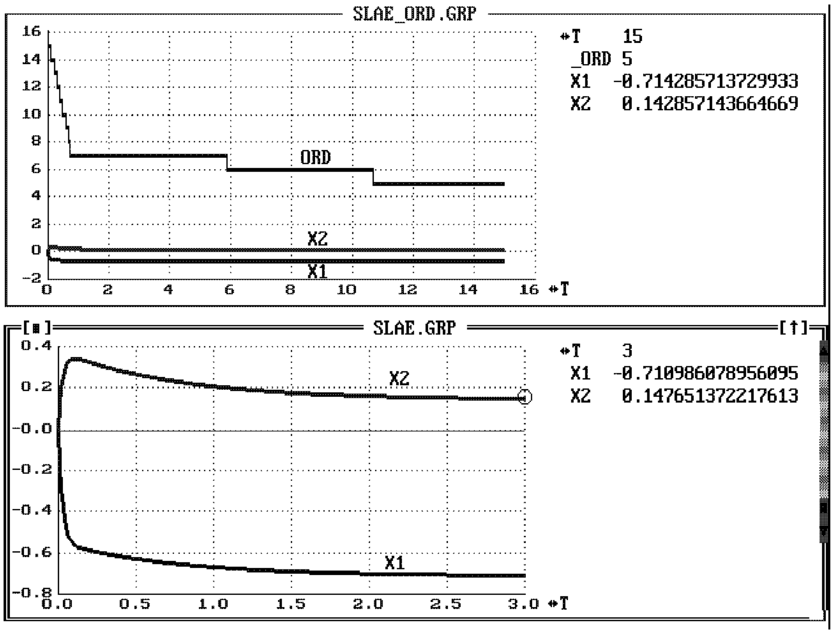

Figure 13.2:

Chapter 14

PARTIAL DIFFERENTIAL EQUATIONS

In practical applications, the second order partial differential equations (PDE) and in particular the linear ones (elliptic, hyperbolic and parabolic) are of special importance. In second order linear partial differential equations with constant coefficients, it is quite easy to obtain the exact solution for some special initial and boundary problems, which makes it easier to compare the methods used and asses their practical value for solving problems with other initial and boundary values, problems with non-constant coefficients or nonlinear problems.

Numerical methods of solving PDE's based on approximations of the derivatives by differences are among basic methods. Let us mention the method of grids and the method of lines. If we cover the space of independent variables with a grid of a finite number of nodes and replace the derivatives by differences using only values in chosen nodes, we will get the method of grids and the solution of a PDE is transformed into the solution of a system of algebraic equations. If we leave the derivatives of one variable continuous and replace the derivatives of other independent variables by differences, we will get the method of lines. Thus the solution of a PDE is transformed into the solution of a system of ordinary differential equations and the system can be solved by means of the Modern Taylor Series Method.

After expressing the second derivative, it is possible, in the simplest case, to use the symmetric formula for the three-point approximation

$$\frac{\partial^2 U(x,t)}{\partial x^2} = \frac{y_{k-1} - 2y_k + y_{k+1}}{(\Delta x)^2}.$$

14.1 Parabolic PDE

A typical parabolic equation is the equation for heat conductivity. It is usually written in the form

$$\frac{\partial^2 U(x,t)}{\partial x^2} = b\frac{\partial U(x,t)}{\partial t}. \tag{48}$$

The above equation is closely connected with the diffusion of gases and therefore it is called a diffusion equation.

The solution of the equation (48)

with initial conditions

$U(x,0) = sin(2\pi x) \qquad x\epsilon < 0,1 >$

at points x_5, x_{15} selected out of the points dividing the interval into 20 segments together with the ORD function is plotted in Fig.14.1 (three-point approximation was used).

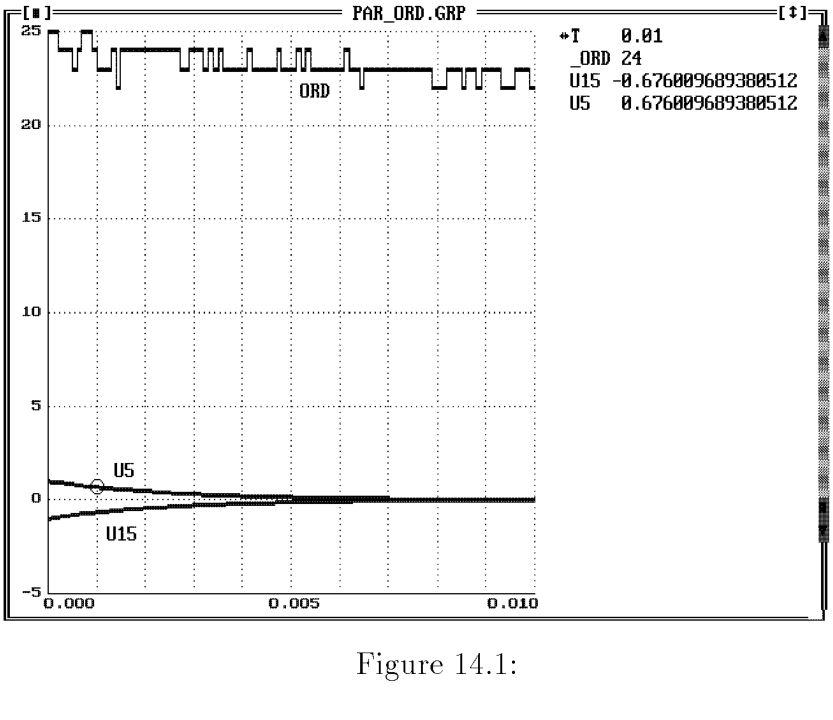

Figure 14.1:

Fig.14.2 shows the solutions (U1, U2, . . . , U19) in a better scale.

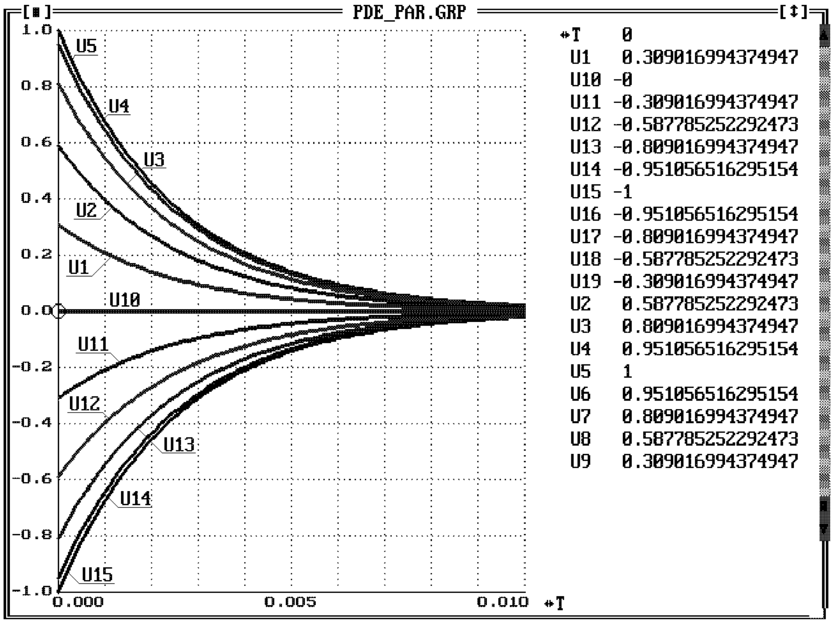

Figure 14.2:

The corresponding source text in TKSL/386 for particular values of parameters is

```
var
  u0,u1,u2,u3,u4,u5,u6,u7,u8,u9,u10,
  u11,u12,u13,u14,u15,u16,u17,u18,u19,u20;
const
  a=400,  k=20,  tmax=0.1,eps=1e-20,  c=2;
system
u1'=a*(u0-2*u1+u2)     &sin(c*3.14159*1/k);
u2'=a*(u1-2*u2+u3)     &sin(c*3.14159*2/k);
u3'=a*(u2-2*u3+u4)     &sin(c*3.14159*3/k);
u4'=a*(u3-2*u4+u5)     &sin(c*3.14159*4/k);
u5'=a*(u4-2*u5+u6)     &sin(c*3.14159*5/k);
u6'=a*(u5-2*u6+u7)     &sin(c*3.14159*6/k);
u7'=a*(u6-2*u7+u8)     &sin(c*3.14159*7/k);
u8'=a*(u7-2*u8+u9)     &sin(c*3.14159*8/k);
u9'=a*(u8-2*u9+u10)    &sin(c*3.14159*9/k);
u10'=a*(u9-2*u10+u11)  &sin(c*3.14159*10/k);
u11'=a*(u10-2*u11+u12) &sin(c*3.14159*11/k);
u12'=a*(u11-2*u12+u13) &sin(c*3.14159*12/k);
u13'=a*(u12-2*u13+u14) &sin(c*3.14159*13/k);
u14'=a*(u13-2*u14+u15) &sin(c*3.14159*14/k);
u15'=a*(u14-2*u15+u16) &sin(c*3.14159*15/k);
```

```
u16'=a*(u15-2*u16+u17) &sin(c*3.14159*16/k);
u17'=a*(u16-2*u17+u18) &sin(c*3.14159*17/k);
u18'=a*(u17-2*u18+u19) &sin(c*3.14159*18/k);
u19'=a*(u18-2*u19+u20) &sin(c*3.14159*19/k);
sysend.
```

14.2 Hyperbolic PDE

One of the commonest hyperbolic PDE is the wave equation, which can be expressed in the basic form

$$b\frac{\partial^2 V(x,t)}{\partial x^2} = \frac{\partial^2 V(x,t)}{\partial t^2} \tag{49}$$

with initial conditions

$$V(x,0) = sin(\pi x) \qquad x\epsilon < 0,1 >$$

$$\frac{\partial V(x,0)}{\partial t} = 0.$$

The equation (49) may describe the swings of an ideal string of unit length, fixed at the extremal points to the x-axis which satisfies the boundary values and is released at time zero (thus having a zero velocity at time zero).

Fig.14.3 shows the solutions V1, V2, ..., V5 of the equation(49) (swing is divided into 10 segments).

Fig.14.4 shows the function V5 (at the midpoint of the string) together with the function ORD.

Note: The analytical solution (at the midpoint of the string) is

$$V_5 = cos\pi t.$$

The corresponding source text in TKSL/386 for particular values of parameters is

```
Var  err,z,
U1, U2, U3, U4, U5, U6, U7, U8, U9,
V0, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10;
Const
  PR= 10,eps=1e-20,
  A= PR*PR,tmax=10,dt=0.1,
```

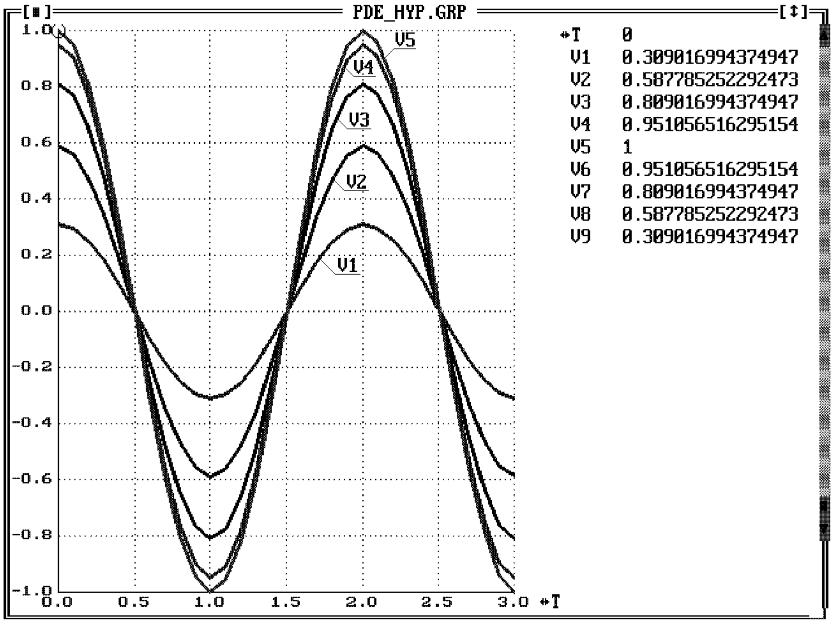

Figure 14.3:

```
  PI= 3.14159265358979323 85;
System
  V0= 0;
  V10= 0;
  U1'= A* (V0-2*V1+V2 )  &0;
  U2'= A* (V1-2*V2+V3 )  &0;
  U3'= A* (V2-2*V3+V4 )  &0;
  U4'= A* (V3-2*V4+V5 )  &0;
  U5'= A* (V4-2*V5+V6 )  &0;
  U6'= A* (V5-2*V6+V7 )  &0;
  U7'= A* (V6-2*V7+V8 )  &0;
  U8'= A* (V7-2*V8+V9 )  &0;
  U9'= A* (V8-2*V9+V10)  &0;
  V1'= U1                &Sin(PI*1/PR);
  V2'= U2                &Sin(PI*2/PR);
  V3'= U3                &Sin(PI*3/PR);
  V4'= U4                &Sin(PI*4/PR);
  V5'= U5                &Sin(PI*5/PR);
  V6'= U6                &Sin(PI*6/PR);
  V7'= U7                &Sin(PI*7/PR);
  V8'= U8                &Sin(PI*8/PR);
  V9'= U9                &Sin(PI*9/PR);
SysEnd.
```

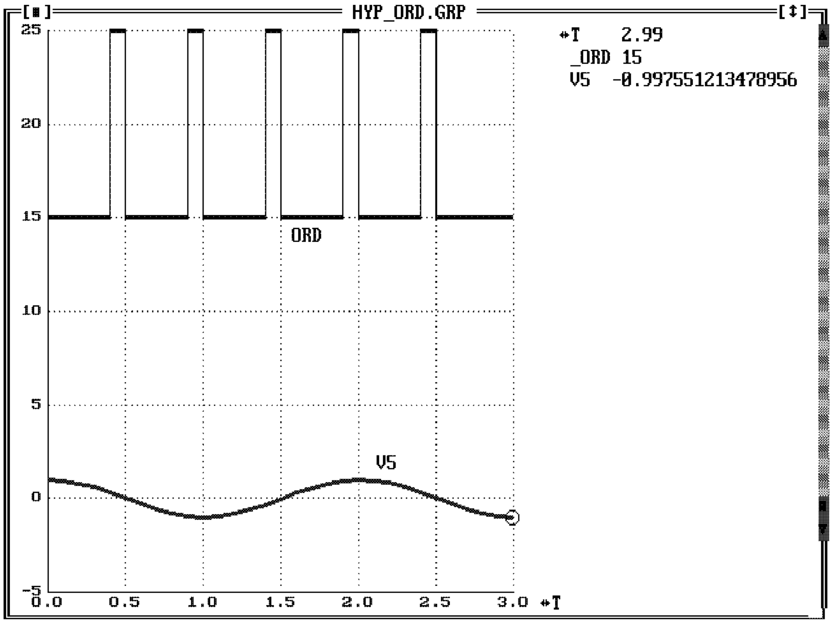

Figure 14.4:

The ERR function (the difference between numerical and analytical solution) of the solution V5 (at the midpoint of the string which is divided into 10 segments) using a three-point approximation is plotted in Fig.14.5 - in the part labelled ER_10_3. The part labelled ER_100_3 plots the ERR function of the deviation (again at the midpoint of the string divided in this case into 100 segments). It follows from Fig.14.5 that the ERR can be decreased by an increase in the number of segments.

The main part of the corresponding source text of ER_100_3 in TKSL/386 is

```
System
  z=cos(PI*t);
  V0  = 0;
  V100= 0;
  U1' = A* (V0-2*V1+V2)       &0;
  U2' = A* (V1-2*V2+V3)       &0;
  .
  .
  U50'= A* (V49-2*V50+V51)    &0;
  .
  .
  U98'= A* (V97-2*V98+V99)    &0;
  U99'= A* (V98-2*V99+V100)   &0;
  V1'= U1                     &Sin(PI*1/PR);
  V2'= U2                     &Sin(PI*2/PR);
```

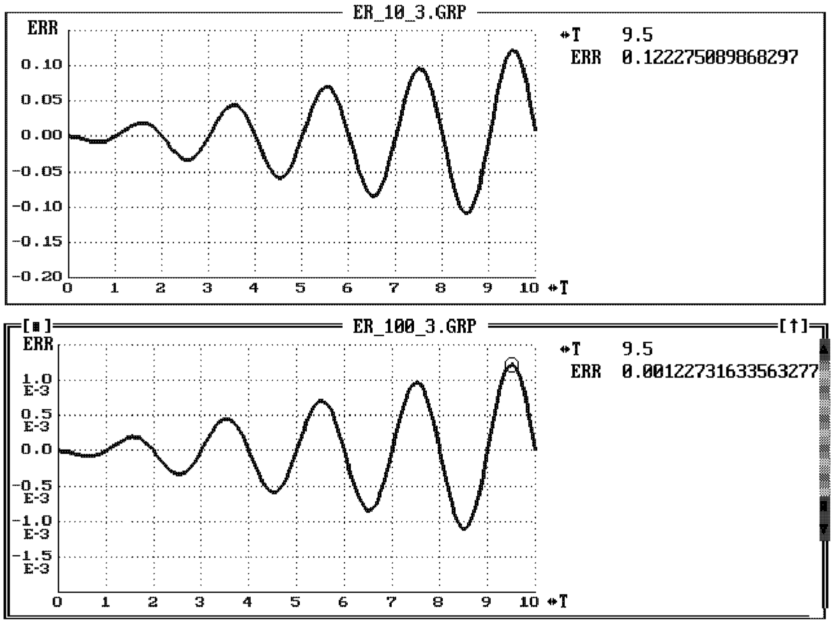

Figure 14.5:

```
  .
  .
  V50'= U50                    &Sin(PI*50/PR);
  .
  .
  V98'= U98                    &Sin(PI*98/PR);
  V99'= U99                    &Sin(PI*99/PR);
  err=z-V50;
SysEnd.
```

The ERR can more effectively be decreased by an increase in the order of the difference formula. The part labelled ER_12_5 of Fig.14.6 plots the ERR function (again at the midpoint of the string divided into 12 segments supposing that a five-point approximation has been used). Further decrease in the ERR (at the midpoint of the string divided into 100 segments supposing that a five-point approximation has been used) is shown in the part labelled ERR_100_5.

The main part of the corresponding source text of ER_100_5 in TKSL/386 is

```
System
z=cos(PI*t);
  V0= 0;
  V100= 0;
  U1'= A* (11*V0-20*V1+6*V2+4*V3-V4)          &0;
  U2'= A* (-V0+16*V1-30*V2+16*V3-V4)          &0;
```

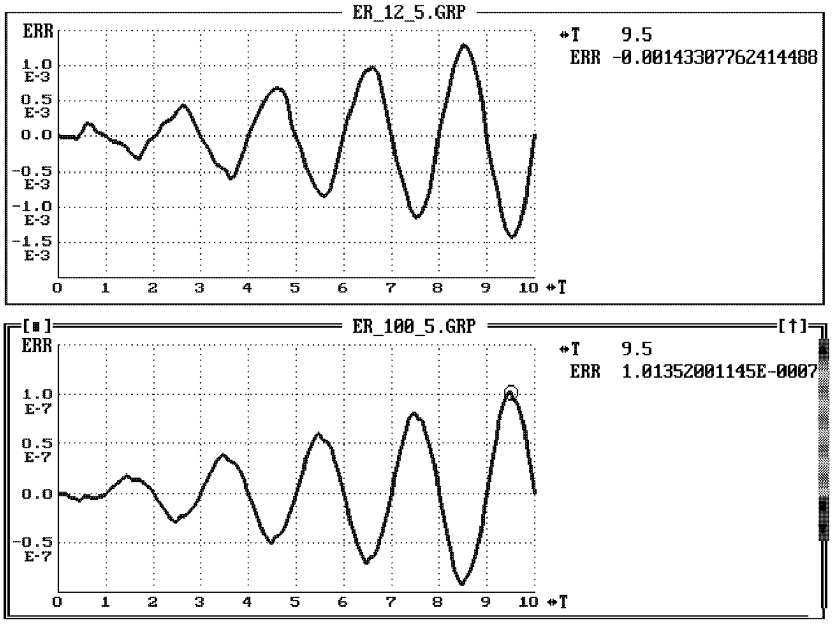

Figure 14.6:

```
  U3'= A* (-V1+16*V2-30*V3+16*V4-V5)          &0;
  .
  .
  U50'= A* (-V48+16*V49-30*V50+16*V51-V52)    &0;
  .
  .
  U97'= A* (-V95+16*V96-30*V97+16*V98-V99)    &0;
  U98'= A* (-V96+16*V97-30*V98+16*V99-V100)   &0;
  U99'= A* (-V96+4*V97+6*V98-20*V99+11*V100)  &0;
  V1'= U1  &0;                                &Sin(PI*1/PR);
  V2'= U2  &0;                                &Sin(PI*2/PR);
  V3'= U3  &0;                                &Sin(PI*3/PR);
  .
  .
  V50'= U50  &0;                              &Sin(PI*50/PR);
  .
  .
  V97'= U97  &0;                              &Sin(PI*97/PR);
  V98'= U98  &0;                              &Sin(PI*98/PR);
  V99'= U99  &0;                              &Sin(PI*99/PR);
err=z-V50;
SysEnd.
```

Note: The error of solution is given only by the method of lines.

Chapter 15

ALGEBRAIC AND TRANSCENDENTAL EQUATIONS

In this chapter a special method to find the real roots of an explicit set of algebraic (or transcendental) equations is described.

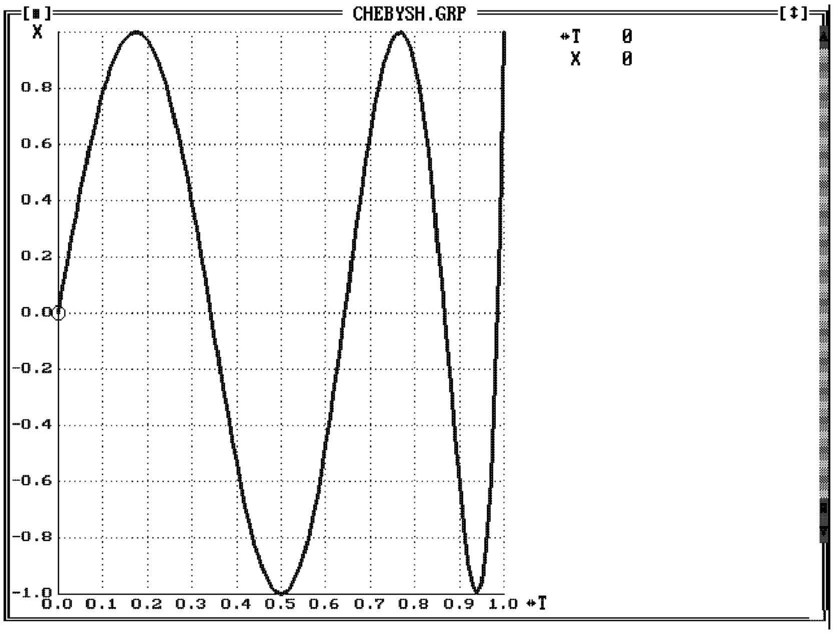

Figure 15.1:

When applicable, the simplest method to obtain solutions to $f(x) = 0$ is to draw a graph of f(x) and read the roots.

As an example the Chebyshev polynom

$f(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x$

is analyzed for $x\epsilon < 0, 1 > (Fig.15.1)$.

The first equation root is obviously x_1=0.

Computation with an automatic stop at finding the root can be used with advantage. The following nonlinear equation of movement pertaining to the solution x is defined

$$\frac{dx}{dt} = \lambda f(x). \tag{50}$$

λ is a suitable positive or negative number. The root is found when $\frac{dx}{dt} = 0$.

The folowing computation scheme is used:

- λ is set at $\lambda = +\ 1$. The system (50) is displaced from the initial stable state (from the first equation root $x_1 = 0$) to a new "non-stable" state such as x(0)=0.005. Fig.15.2 plots the substitute solution x (for x(0)=0.005). The new stable state $x_2 = 0.342020143325669$ is found.

- λ is set at $\lambda =$ - 1. The system (77) is displaced from the previous stable state (from the equation root $x_2 = 0.342020143325669$) to a new "non-stable" state such as x(0)=0.36. Fig.15.3 labelled CHEB2 plots the substitute solution x (for x(0)=0.36). The new stable state $x_3 = 0.642787609686539$ is found.

The substitute solution x (for x(0)=0.005) and the ORD function is plotted in Fig.15.3 - in the part labelled CHEB1. The stable state is characterized by the value of ORD=1. Similarly, the part labelled CHEB2 of Fig.15.3 plots the substitute solution x (for x(0)=0.36, λ =-1) together with ORD and the root $x_3 = 0.642787609686539$.

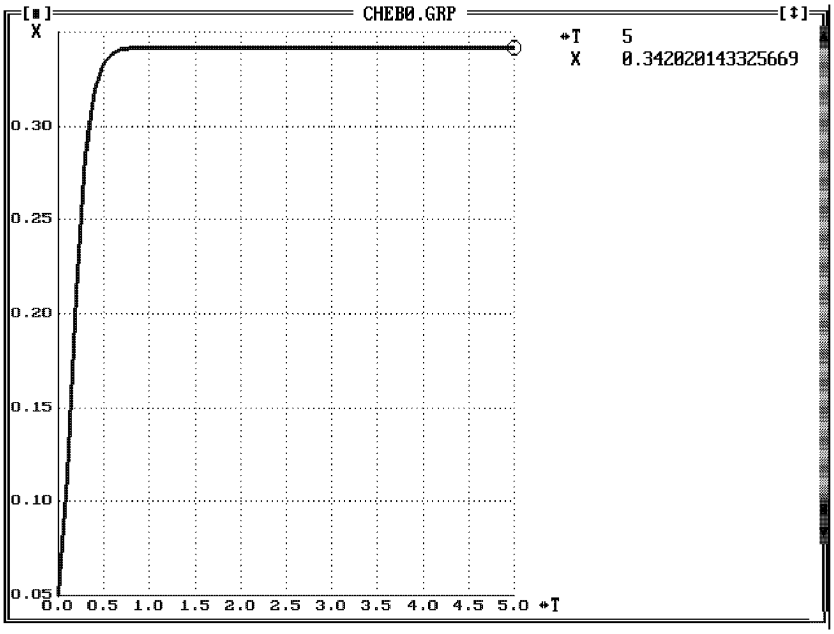

Figure 15.2:

Similarly x_4=0.866025403784439 and x_5=0.984807753012208 were obtained.

Note:
Similarly also one of the real roots of the quadratic equation

$$x^2 - 1 = 0$$

can be found (Fig.15.4).

Conclusion:
The computation process finds automatically one root of the given nonlinear algebraic equation. The root is found when

$\frac{dx}{dt} = 0.$

λ controls the computation process.

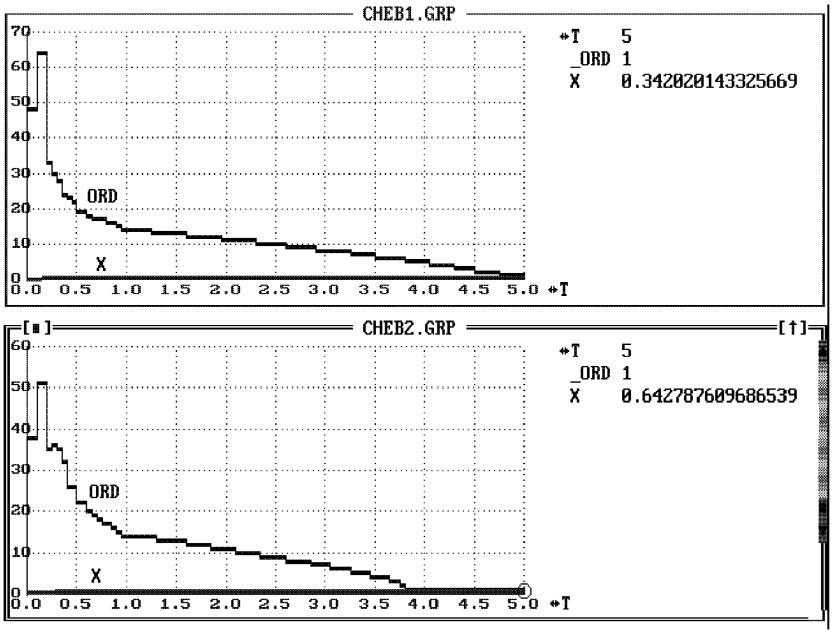

Figure 15.3:

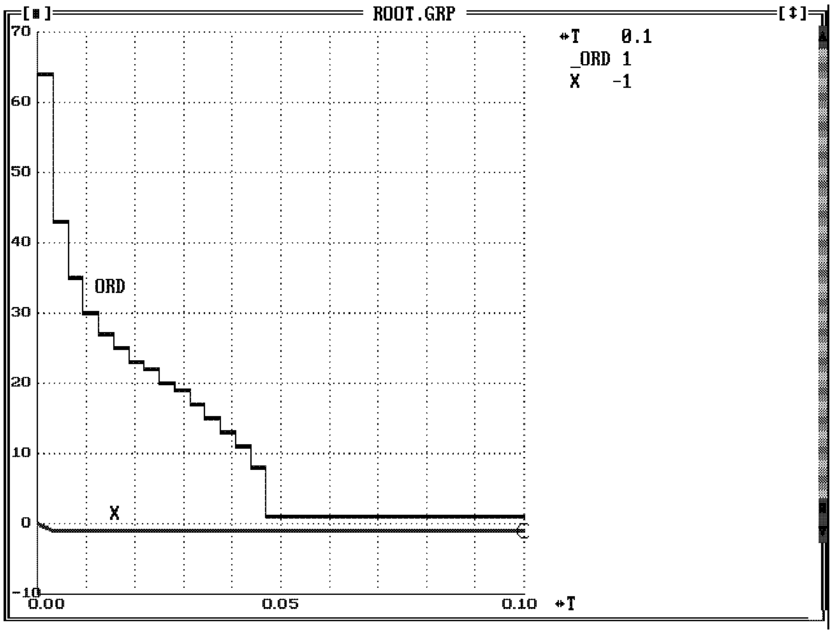

Figure 15.4:

Chapter 16

REPEATED COMPUTATIONS

The repeated computations for different values of the parameter have all been plotted in a common graph, which is very illustrative.

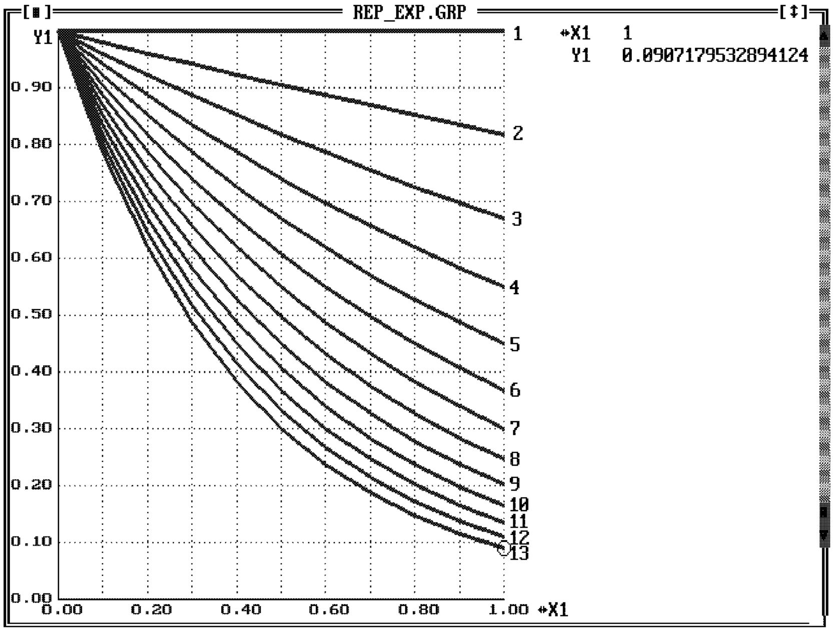

Figure 16.1:

Fig.16.1 shows a numerical solution of equation (51)

$$y_1' = -ay_1 \qquad y_1(0) = 1 \tag{51}$$

for a=0 (curve 1), a=0.2 (curve 2), a=0.4 (curve 3), ... a=2.4 (curve 13).

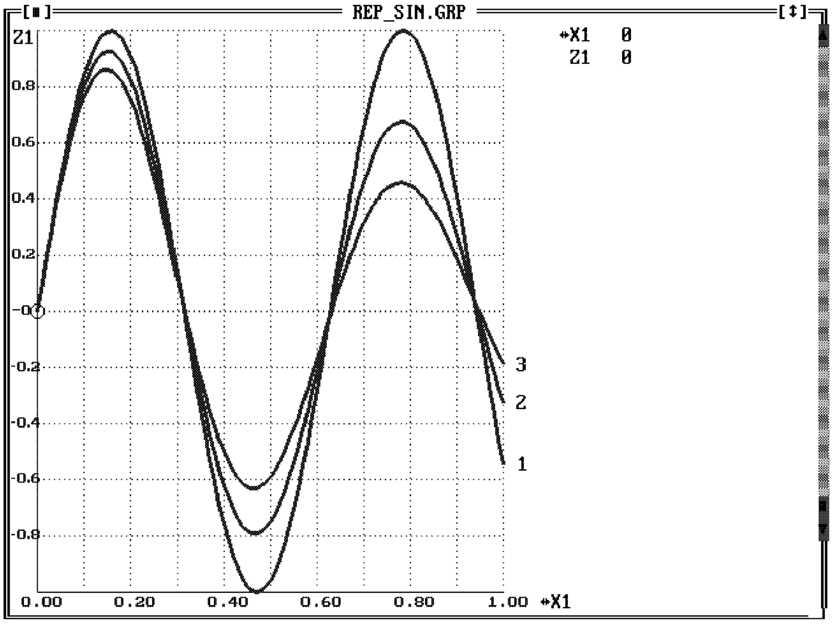

Figure 16.2:

Analogously Fig.16.2 shows a numerical solution of the system
$y' = -a * y - 100 * z_1 \qquad y(0) = 1$
$z_1' = y \qquad z_1(0) = 0$
for a=0 (curve 1), a=0.5 (curve 2), a=1 (curve 3).

The method of obtaining the numerical solution plotted in Fig.16.2 is shown in Fig.16.3.

The plotting of two solutions of a physical pendulum in the phase plain for two values of the initial conditions for the angular velocity is also very illustrative - Fig.16.4 ($z_1(0)$=13 rad s^{-1}... curve 1; $z_1(0)$=14 rad s^{-1}... curve 2).

The same is true for the plotting of the solution of Van-der-Pol equation for $\mu = 0$ (curve 1), $\mu = 0.5$ (curve 2), $\mu = 1$ (curve 3) - Fig.16.5.

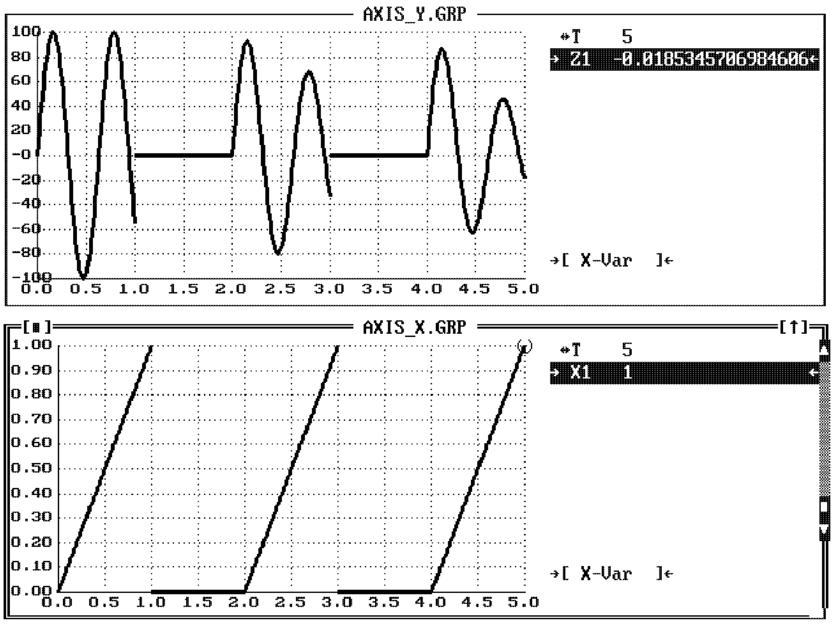

Figure 16.3:

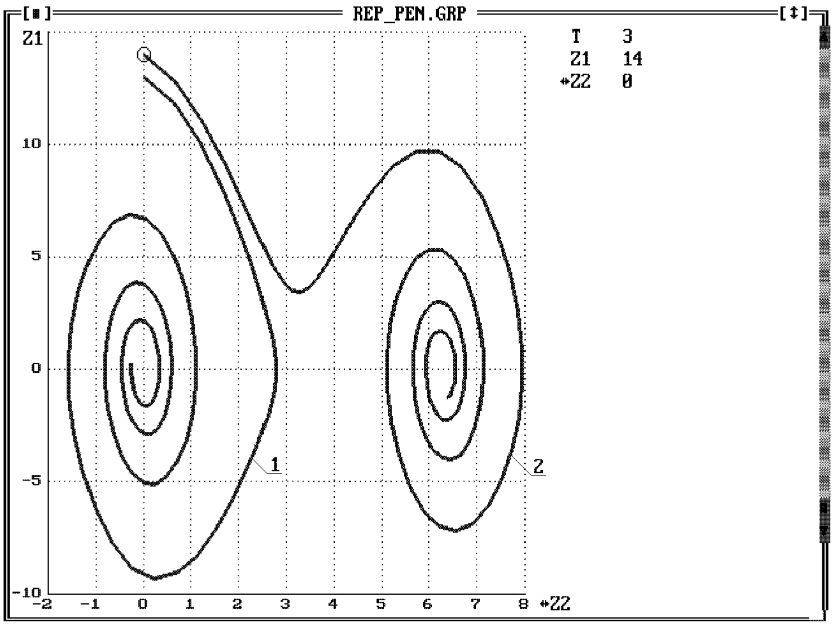

Figure 16.4:

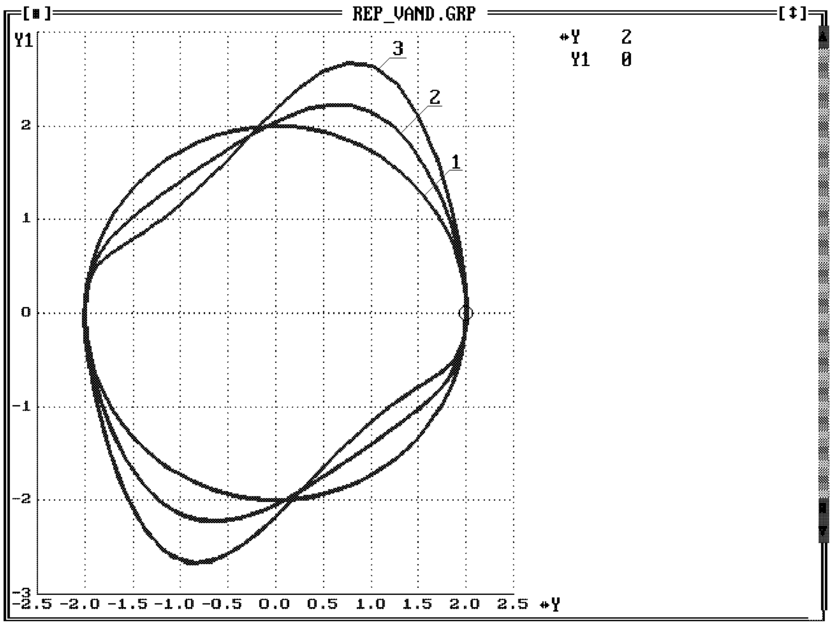

Figure 16.5:

Chapter 17

ITERATIVE COMPUTATIONS

As an example of an iterative computation, the computation of an equivalent initial condition of the differential equation (52)

$$y_1' = -y_1 \qquad y_1(1) = 0.273 \tag{52}$$

has been chosen.

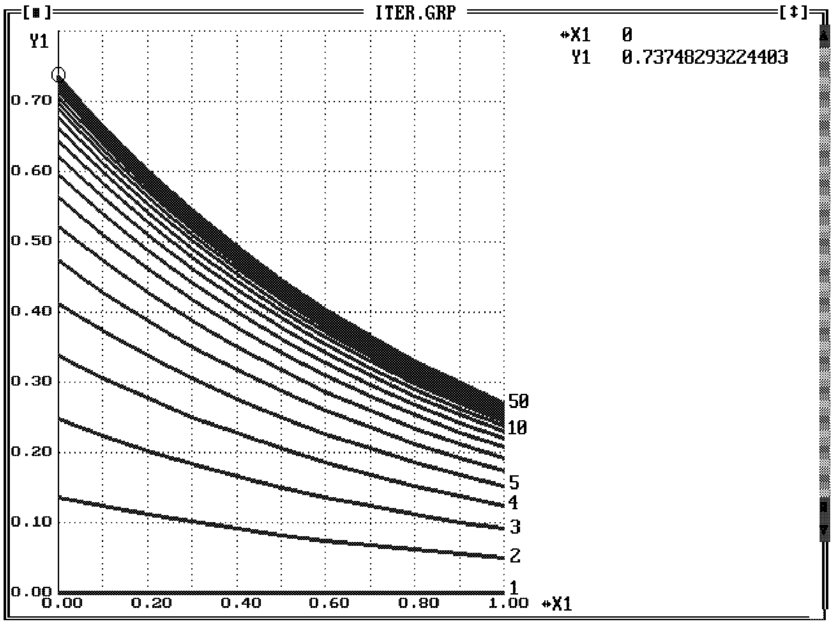

Figure 17.1:

A new value of the equivalent initial condition IC (in the cycle (i+1)) is determined by the following formula

$$IC^{i+1} = IC^i + \beta(y_1(1)^i - 0.273).$$

In each cycle of the computation the initial condition $y_1(0)$ is determined, the corresponding calculation is carried out and the deviation at the end-point is evaluated (Fig.17.1). Curve 1 is for $y_1(0) = 0$, curves $2, 3, \ldots 50$ show the step-wise improving of the accuracy of the solution ($\beta = 0.5$).

The time functions of the equivalent initial condition IC to be found and corresponding deviation ER (at the end-point) during the iteration process are plotted in Fig.17.2 (for $\beta = 0.5$).

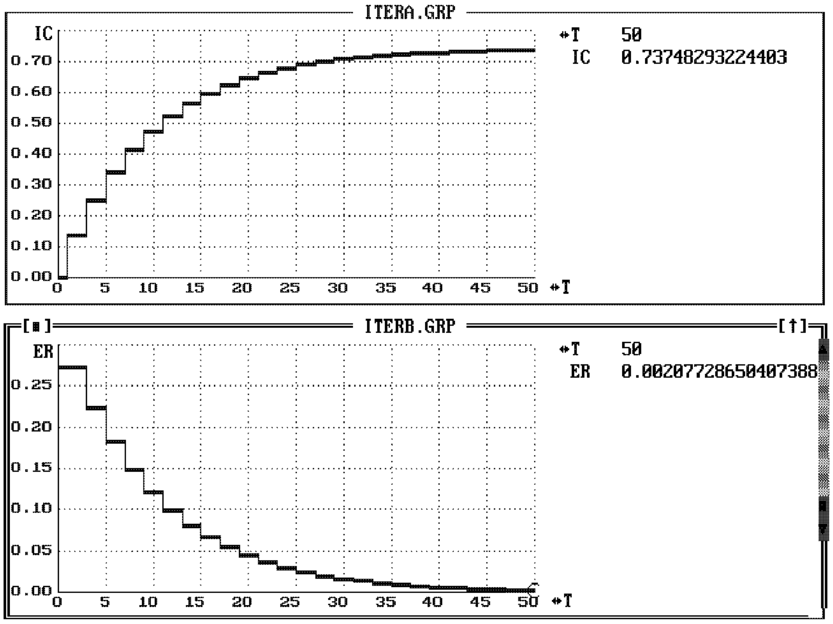

Figure 17.2:

The influence of β on the solution of the given iteration scheme is shown in Fig.17.3 - for $\beta = 3$ (diagram labelled ITERC), for $\beta = 4.8$ (diagram labelled ITERD) and for $\beta = 5.5$ (diagram labelled ITERE).

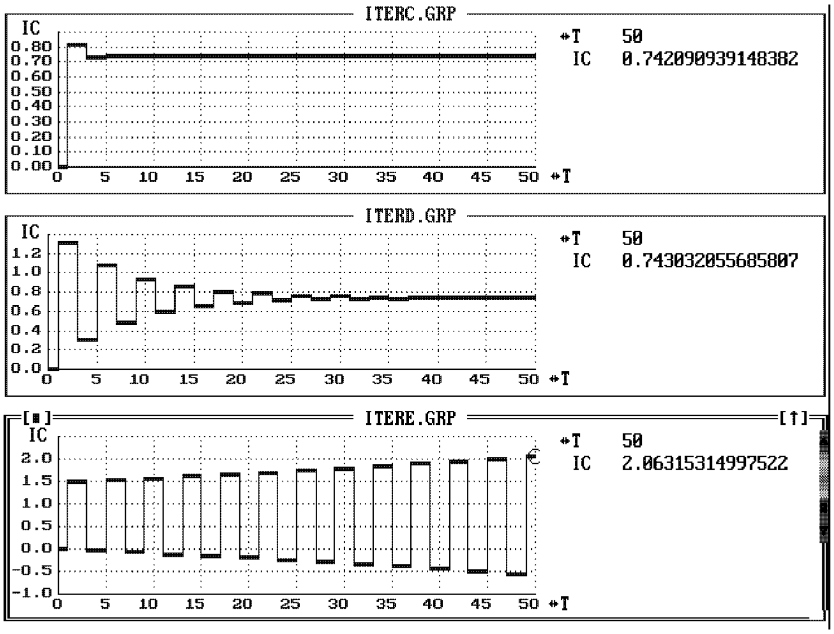
ITERC.GRP
IC
T 50
IC 0.742090939148382
ITERD.GRP
IC
T 50
IC 0.743032055685807
ITERE.GRP
IC
T 50
IC 2.06315314997522
T

Figure 17.3:

Chapter 18

SIMULATION LANGUAGE TKSL/ORCAD

The created simulation language TKSL/386 is a very powerful computing tool for finding accurate numerical solutions of differential equations. The next version of the simulation language TKSL/ORCAD is based on a graphic interface. The graphic interface transforms a graphic representation (graphic symbols for describing real-existing systems) into an equivalent textual representation. The graphic editor that has been chosen is a part of a system ORCAD .

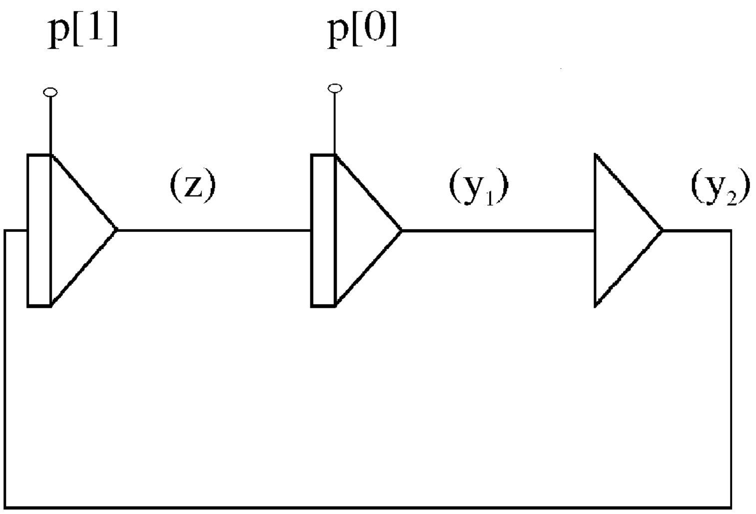

Figure 18.1: Analogue diagram

The analogue diagram (Fig.18.1) solving the system of equations (53),(54),(55) is well-known in the theory of analogue and hybrid computers.

```
y1'= z              y1(0)=0                                     (53)
z' = y2              z(0)=2                                     (54)
y2 =-y1.                                                        (55)
```

The analogue diagram (Fig.18.1) consists of two integrators (with predefined initial conditions) and one invertor. The integrators and invertor are among the basic elements of a specially created library of the simulation language TKSL/ORCAD (the library consists of integrators, invertors, summers, function generators, multipliers, dividers, blocks of transfer functions, nonlinear blocks and blocks defined by users). When the simulation language TKSL/ORCAD is used, the diagram (Fig.18.1) must be plotted on the screen (according to the rules of the system ORCAD). After the program is started, a source code identical with the source code of the language TKSL/386 is automatically generated.

Chapter 19

SIMULATION LANGUAGE TKSL/TRANSP

The block notation and block diagram (used in analogue and hybrid computers and analyzed in TKSL/ORCAD) is the underlying feature of the newly completed simulation language TKSL/TRANSP. It is known that the Taylor series method is a parallel one and therefore attempts at a parallel interpretation in a transputer network can be expected.

Transputers are high performance microprocessors that support parallel processing. They can be connected together in any configuration and they can form a building block for complex parallel processing systems.

19.1 OCCAM Programs and Procedures

Analogue diagrams represent a very convenient tool for describing parallel tasks. To show this the system of equations (53),(54),(55) and the analogue diagram (Fig.18.1) are analyzed again in this chapter.

The software products of INMOS are described by means of the so called folds. They are program parts preceded by three dots (... fold identification) which cannot be seen while editting but when the program is processed, these parts expand into the source text. We will use this method for the subsequent description of programs solving model problems.

OCCAM programs solving a system of differential equations expressed in the form of an analogue diagram, implemented on one transputer have the following structure (in the fold notation):

```
... program header
... block of declarations of variables and constants
...  procedures for  inputting parametres  and printing  out
     values
SEQ  -- main program
```

```
  ... initialization
  SEQ k=1 FOR n -- for n solution steps
    ... declarations of procedures of analogue elements
    ... declarations of communication channels
    PAR
      ... interconnecting analogue elements
    ... printing out values
: -- end of program
```

The program header defines the SP protocol - a communication protocol between the host computer IBM PC and a transputer: further it defines the name of the program, communication channels between the program process and the environment and the input/output of the library.

In the declaration fold the declarations of method order, integration step, solution time, model time, computing time, initial conditions etc. are placed.

The procedure inputting the parameters of the computation is inserted in the program using the directive

```
#INCLUDE,
```

in the same way as the procedure for the print-out and procedures describing the elements of the analougue diagram.

In the initialization fold parameters are inputted, number of computation steps is calculated, model time is initialized, initial conditions are set, result table header and initial conditions are printed out etc.

The fold describing the interconnection of the elements of the analogue diagram defines the interconnection of the elements according to the analogue diagram using communication channels.

In the following text the most important part of the OCCAM program - the interconnection of the elements (for the example given in Fig.18.1) is described.

```
-- interconnecting analogue elements
integrator.1(p[0],z,y1)
integrator.1(p[1],y2,z)
invertor(y1,y2)
```

Note: p[0] means the initial condition, z means the input and y1 means the output of the first integrator. Similarly, p[1] is the initial condition, y2 is the input and z is the output of the second integrator; y1 is the input and y2 is the output of the invertor.

The integrators and the invertor are taken as procedures, the corresponding calculations are viewed as processes. The integrators and invertor are interconnected by channels (for the definition of the channels the symbols (y1), (y2), (z) from Fig.18.1 are used).

The procedure for an invertor in OCCAM is in the form:

```
--invertor.occ
PROC invertor (CHAN OF REAL64 vstup, vystup)
  REAL64 a,b:
  SEQ
    vstup?a
    b:=-a
    vystup!b
: -- end
```

The procedure for an integrator in OCCAM is in the form:

```
--integr1.occ
PROC integrator.1 (REAL64 p,CHAN OF REAL64 vstup,
    CHAN OF REAL64  vystup)
  REAL64 a,b:
  SEQ
    a:=p
    SEQ
      PAR
        vystup!a
        vstup ?b
      a:=h * b
      p:=p+a
:  --end
```

Note: For clearness' sake, the numerical method shown in the above procedure uses only the first term of the Taylor series (the method order is defined as ORD=1). Obviously, the quality of the computation is significantly enhanced by using a higher order method.

The complete program for calculating the system of equations (53),(54),(55) follows:

```
--program sin
#INCLUDE "hostio.inc" ---- contains SP protocol
PROC sin (CHAN OF SP fs, ts, []INT memory)
  #USE "hostio.lib"  -- iserver libraries
```

```
-- declarations of variables, constants and initial conditions
INT n,t,tb,te:
REAL64 T,h,tp:
VAL [] REAL64 y0 IS [2.0(REAL64),0.0(REAL64)]:
[2] REAL64 p:

-- procedures for inputting parameters and printing out values
#INCLUDE "in.occ"
#INCLUDE "out.occ"

-- main program

SEQ
  inp(T,h)
  --initialization
  tp:=0.0(REAL64)
  t:=0(INT)
  n := INT ROUND (T/h)
  p[0]:=y0[0]
  p[1]:=y0[1]
  so.write.nl(fs,ts)
  so.write.string.nl(fs,ts,"   tp        t     z               y(1)")
  so.write.string.nl(fs,ts,"   [s]      [us] ")
  outp(p,tp,t)
  TIMER ti:
  SEQ k=1 FOR n                    -- for n steps
    PRI PAR
      SEQ
        ti?tb
        tp:=(tp+h)

        -- declarations of procedures of analogue elements
        #INCLUDE "integr1.occ"
        #INCLUDE "invertor.occ"

        -- interconnecting analogue elements
        CHAN OF REAL64 z,y1,y2:
        PAR
          integrator.1(p[0],y2,z)
          integrator.1(p[1],z,y1)
          invertor(y1,y2)

        --printing results
```

```
          ti?te
          t:= te MINUS tb
          outp(p,tp,t)
        SKIP
    so.exit(fs,ts,sps.success)
: -- end
```

The following procedures for reading and printing were used:

```
--in.occ
PROC inp (REAL64 T,h)
  BOOL error:
  SEQ
    error:=TRUE
    WHILE error
      SEQ
        so.write.string(fs,ts,"Tmax                       : ")
        so.read.echo.real64(fs,ts,T,error)
        so.write.nl(fs,ts)
        IF
          error
            so.write.string.nl(fs,ts,"err")
          TRUE
            SKIP
    error:=TRUE
    WHILE error
      SEQ
        so.write.string(fs,ts,"Integration step h         : ")
        so.read.echo.real64(fs,ts,h,error)
        so.write.nl(fs,ts)
        IF
          error
            so.write.string.nl(fs,ts,"err")
          TRUE
            SKIP
: -- end
```

```
--out.occ
PROC outp (VAL [] REAL64 y,VAL REAL64 tp,VAL INT t)
  SEQ
    so.write.real64(fs,ts,tp,3,3)
    so.write.int(fs,ts,t,6)
    so.write.string(fs,ts,"  ")
    SEQ i=0 FOR SIZE y
      so.write.real64(fs,ts,y[i],3,10)
    so.write.nl(fs,ts)
:
```

In the text which follows, for clearness' sake, the computation of the system (53),(54),(55) is shown for h=0.1s, T=1s. Deviations from the analytical solution are of course influenced by the quality of the numerical integration method used.

```
Booting root transputer...ok
Tmax                      : 1.0
Integration step h        : 0.1

   tp       t     z             y(1)
   [s]     [us]
   0.0      0     2.0           0.0
   0.1     31     2.0           0.2
   0.2     31     1.98          0.4
   0.3     32     1.94          0.598
   0.4     32     1.8802        0.792
   0.5     32     1.801         0.98002
   0.6     31     1.702998      1.16012
   0.7     32     1.586986      1.3304198
   0.8     31     1.45394402    1.4891184
   0.9     32     1.30503218    1.634512802
   1.0     31     1.1415808998  1.76501602
```

Note: For completeness' sake the column " t[us]" gives the time of computation of one integration step (in microseconds).

A program designed in this way is set up for exactly one particular dynamic system. After the program has been started, it requires the time of program termination and the integration step.

As in the program in its present form only very simple communication with the operator in the form of a "running column of numbers" is possible, the TKSL/TRANSP has been created so that a repeated start of the calculation is possible without having to load a new program in the transputer.

19.2 TKSL/TRANSP

The source code of the computation of the system (53),(54),(55) (based on Fig.18.1) in the simulation language TKSL/TRANSP is in the form:

```
int2(inv1),(0),t1,"z";
inv1(int1),(),t1,"y2";
int1(int2),(2),t1,"y1";
#sin;
```

Note: t1 means that all elements are placed in the transputer 1. The meaning of the first line of the source code is as follows:

Invertor 1 (inv1) represents the input of the integrator 2 (int2); initial condition of integrator 2 is set to 0; the integrator 2 is placed in the transputer 1 (t1); z is the name of the variable.

From the above analysis, it is clear that the TKSL/TRANSP created for simulating dynamic systems makes it possible to automatically write a program for a transputer or a network of transputers from a textual description. Thus, when simulating, it is no longer necessary to debug a program in OCCAM.

19.3 Results

Almost all the work with OCCAM is done automatically. This eliminates one of the main problems of using the transputers for practical computing.

When using a transputer network, any element of the state diagram can be placed practically in any transputer of the network. The problem is, however, the serial communication. The reason is that the serial communication may be almost as time consuming as the calculation itself.

The system elements must then be placed in the network in such a way that the transputers communicate with one another as little as possible.

For further development in this area it will be necessary to construct a tool for optimizing the layout of the elements of a particular dynamic system in the transputer network used. The number of communication channels among the transputers may be the optimization criterion.

Chapter 20

CONCLUSION

The principal idea of an extremely accurate and fast method for numerical solutions of technical initial problems is presented in the paper. The number of Taylor series terms and word width are of great importance during computation process.

The numerical solutions of technical initial problems are most exact when the method order ORD is accordingly high for the given integration step h. The accuracy of the result can be influenced in a considerable way by the word width.

Typical technical initial problems and their solutions are shown in the paper. All solutions were obtained by means of the simulation language TKSL.

The greatest contribution of the method is that it facilitates the detections and solutions of stiff systems and systems with discontinuities.

In a very straightforward way it is possible to solve partial differential equations by the method of lines.

Linear and nonlinear algebraic equations can also be solved by the corresponding differential equations.

It is known that the Taylor series method is a parallel one and therefore attempts at a parallel interpretation in a transputer network have been analyzed and new simulation language TKSL/TRANSP based on TKSL/ORCAD has been created. TKSL/TRANSP created for simulating dynamic systems makes it possible to automatically write a program for a transputer or a network of transputers from a textual description. Thus, when simulating, it is no longer necessary to debug a program in OCCAM.

Chapter 21

REFERENCES

[1] Barton,D., Willers,I.M.,Zahar,R.V.M.:
"Taylor Series Methods for Ordinary Differential Equations - An Evaluation"
In Mathematical Software, ed. John R. Rice
Academic Press, New York and London, 369-390,1971

[2] Barton,D.:
"On Taylor Series and Stiff Equations"
ACM Trans. on Mathematical Software, Vol.6, No.3, 280-294,1980

[3] Brown,W.S., Hearn,A.C.:
"Applications of Symbolic Algebraic Computation"
Computer Physics Communications, Vol.17, 207-215, 1979

[4] Buehrer,R.E., Brundiers,H., Benz,H., Bron,B.,
Friess,H., Haelg,W., Halin,H.J., Isacson,A., Tadian,M.:
"The ETH-Multiprocessor EMPRESS: A Dynamically Configurable MIMID System"
IEEE Transactions on Comp., Vol. c-31, No.11,1035-1044, 1982

[5] Byrne,G.D., Hindmarsh,A.C.:
"A Polyalgorithm for the Numerical Solution Of Ordinary Differential Equations"
ACM Trans. on Mathematical Software, Vol.1, 1975

[6] Carver,M.B., Mac Ewen,S.R.:
"Numerical Analysis of a System Described by Imlicitly-Defined Ordinary Differential Equations Containing Discontinuities"
Appl. Math. Modelling, Vol.2, 280-286, 1978

[7] Chang,Y.F.:
"Automatic Solution of Differential Equations"
in Lecture Notes in Mathematics No. 430, Constructive and Computational Methods for Differential Equations,
ed D.L. Colton and R.P. Gilbert, Springer-Verlag, Berlin, Heidelberg, New York, 1974

[8] Chang,Y.F., Fauss,J., Prieto,M., Corliss,G.:
"Convergence Analysis of Compound Taylor Series"
Proc. Eight Manitoba Conference on Numerical Math. and Computing, 129-152, 1978

[9] Chang Y.F.:
"Taylor Series as an Analytic Tool for PDE Solutions"
Proc. Seventh Manitoba Conference on Numerical Math. and Computing, 269-277, 1977

[10] Collatz,L.:
The Numerical Treatment of Differential Equations
Springer-Verlag, Berlin, Heidelberg, New York, 1960

[11] Corliss,G., Lowery,D.:
"Choosing a Stepsize for Taylor Series Methods for Solving ODEs"
Journal of Computational and Applied Mathematics
Vol.3, No.4, 251-256, 1977

[12] Corliss,G.F.:
"Integrating ODE's in the Complex Plane-Pole Vaulting"
Math. Comp., Vol.35, No.152, 1187-1189, 1980

[13] Deprit,A., Zahar,R.V.M.:
"Numerical Integration of an Orbit and its Concomitant Variations by Recurrent Power Series"
ZAMP, Vol.17, 425-430, 1966

[14] Ehle,B.L.:
"High Order A-Stable Methods for the Numerical Solution of Systems of DEs"
BIT, Vol.8, 276-278, 1968

[15] Engeli,M.E.:
"Achievements and Problems in Formula Manipulation"
in Information Processing, ed. A.J.H. Morrell

Vol.68, North Holland Publ. Co., Amsterdam, 24-32, 1969

[16] Enright,W.H.:
"Studies in the Numerical Solution of Stiff Ordinary Differential Equations"
Tech. Rept. 46, Dept. of Computer Science, University of Toronto, Toronto, 1972

[17] Enright,W.H., Bedet,R., Farkas,I., Hull,E.:
"Test Results on Initial Value Methods for Non-Stiff Ordinary Differential Equations"
Tech. Rept. 68, Dept. of Computer Science, University of Toronto, Toronto, 1974

[18] Fatunla,S.O.:
"Numerical Integrators for Stiff and Highly Oscillatory Differential Equations"
Mathematics of Computation, Vol.34, No.150, 373-390, 1980

[19] Gautschi,W.:
"Computation of Successive Derivatives of f(z)/z"
Math. Comp., Vol.20, 209-214, 1966

[20] Gear,C. William.:
Numerical Initial Value Problems in Ordinary Differential Equations
Prentice-Hall, Englewood Cliffs, N.J., 1971

[21] Gibbons,A.:
"A Program for the Automatic Integration of Differential Equations Using the Method of Taylor Series"
Computer Journal, Vol.3, 108-111, 1960

[22] Goldfine,A.:
"Taylor Series Methods for the Solution of Volterra Integral and Integro-Differential Equations"
Math. Comp., Vol.31, No.139, 691-707, 1977

[23] Halin,H.J.:
"Integration of Ordinary Differential Equations Containing Discontinuities"
Proc 1976 Summer Computer Simulation Conf.,
Washington, D.C., 46-53, 1976

[24] Halin,H.J.:
"Integration across Discontinuities in Ordinary Differential Equations Using Power Series"
SIMULATION, 33-45, 1979

[25] Halin,H.J., Kriz,J.:
"On the Accurate Treatment of Fixed and Variable Time Delays"
Proc. 1979 Summer Computer Simulation Conf.,
Seattle, Wash., 130-134, 1979

[26] Halin,H.J., Buehrer,R., Haelg,W., Benz,H.,
Bron.,B., Brundiers,H., Isacson,A., Tadian,M.:
"The ETH Multiprocessor Project: Parallel Simulation of Continuous Systems"
SIMULATION, 109-123, 1980

[27] Halin,H.J.:
"A Fast and Accurate Method for the Integration of Imlicit Differential Equations and tha Treatment of Algebraic Loops"
accepted for publication in Mathematics and Computers in Simulation, ed. R. Vichnevetsky

[28] Halin,H.J.:
PSCSP - Power Series Continuous - System Simulation Program
Reference Manual, ETH, Zurich

[29] Hanson,J.W., Caviness,J.S., Joseph, C.:
"Analytic Differentiation by Computer"
Communications of the ACM, Vol.5, 349-355, 1962

[30] Hull,T.E., Johnston,R.L.:
"Optimum Runge-Kutta Methods"
Math. Comp., Vol.18, 306-310, 1964

[31] Kedem,G.:
"Automatic Differentiation of Computer Programs"
ACM Transactions on Mathematical Software
Vol.6, No.2, 150-165, 1980

[32] Korn,G.A., Wait,J.V.:
Digital Continuous-System Simulation
Prentice - Hall, Engewood Cliffs, 1978

[33] Kunovsk,J.:
Special Metodology of Parallel Processing in Modelling, ASIM 90, Wien, Austria, 1990,pp.323-326

[34] Kunovsk,J.:
Specialized Multiprocessor system, ACA, Newtown,Pensylvania, USA, 1991,pp.138-152

[35] Kunovsk,J.,Mikulek,K.:
TKSL - Taylor Kunovsk Simulation Language, Eurosim 92, Capri, Italy ,pp.169-174

[36] Kunovsk J.:
Exact and Fast Numerical Solutions Of Nonlinear Differential Equations, ADIUS 92, Farnborough, England

[37] Morrison,D.:
"Optimal Mesh Size in the Numerical Integration of an Ordinary Differential Equation"
Journal of the ACM, Vol.9, 98-103, 1962

[38] Nordsieck,A.:
"On the Numerical Integration of Ordinary Differential Equations"
Math. Computation, Vol.16, 22-49, 1962

[39] Norman,A.C.:
"Expanding the Solutions of Implicit Sets of Ordinary Differential Equations in Power Series"
The Computer Journal, Vol.19, No.1, 63-68, 1976

[40] Rall,L.B.:
Automatic Differentiation: Techniques and Applications
Lecture Notes in Computer Science No.120
Springer-Verlag, Berlin, Heidelberg, New York, 1981

[41] Ralston,A., Wilf,H.S.:
Mathematical Methods for Digital Computers, Vol.1
John Wiley and Sons, New York, 1960

[42] Reiter,A.:
"Automatic Generation of Taylor Coefficients (TAYLOR) for the CDC 1604"

MRC Tech. Summary Rept. No. 830
University of Wisconsin-Madison, 1967

[43] Rentrop,P.:
"A Taylor Series Method for the Numerical Solution of Two-Point Boundary Value Problems"
Numer. math., Vol.31, 359-375, 1979

[44] Richtmyer,R.D.:
"Detached-Shock Calculations by Power Series"
International Atomic Energy Commission Research and Development Report NYU-7973, Courant Inst. of Math.Sci., 1957

[45] Shampine,L.F., Watts,H.A., Davenport,S.M.:
"Solving Nonstiff Ordinary Differential Equations-The State of The Art"
SIAM REVIEW, Vol.18, No.3, 1976

[46] Stone,H.S. (ed.):
Complexity of Sequentiell and Parallel Algorithms
Academic Press, London and New York, 1973