


SCIENCE & ENGINEERING FOR THE COMMODORE



A DATA BECKER BOOK BY RANIER BARTEL

YOU CAN COUNT ON
Abacus 
Software

SCIENCE & ENGINEERING

on the Commodore-64

by Ranier Severin

A DATA BECKER BOOK

Abacus You Can Count On  **Software**

Chapter 3 - Input and Output.....71

3.1	The INPUT command.....	71
3.2	The GET command.....	73
3.3	Screen input and output.....	74
3.4	Handling data with the 1541 disk drive.....	78
3.5	Relative files on the 1541.....	80
3.6	Storing vectors and matrices.....	90
3.7	The PRINT USING subroutine.....	92

Chapter 4 - Sorting routines.....95

4.1	The bubble sort.....	97
4.2	Linear sort.....	100
4.3	The shell sort.....	101
4.4	The quick sort.....	104

Chapter 5 - Mathematical programs.....111

5.1	Zero-point determination.....	111
5.2	Romberg differentiation.....	116
5.3	Numerical integration.....	119
5.3.1	Simpon's Rule.....	119
5.3.2	Romberg Integration.....	122
5.4	Linear regression.....	123
5.5	Probability calculation.....	128
5.5.1	Binomial distribution.....	133
5.5.2	Chi ² distribution and adaptability test.....	141
5.6	Fourier analysis and systhesis.....	149
5.7	Numerical solutions to initial value problems.....	162
5.7.1	Comparison program for DEQ systems.....	162

Table of Contents

Chapter 1 - Introduction.....	1
1.1 The Commodore 64 and science.....	1
1.2 Assumptions.....	3
1.3 Necessary minimal configuration.....	4
Chapter 2 - Programming solutions to scientific problems...	5
2.1 BASIC - Advantages and disadvantages.....	5
2.2 BASIC dialects.....	9
2.3 Pascal and other languages.....	17
2.3.1 Pascal - Solving problems with a structured language.....	17
2.3.2 ADA - A programming language from the DoD.....	20
2.3.3 FORTH - Creative programming.....	21
2.3.4 Logo and other languages.....	23
2.4 From problem to algorithm.....	24
2.4.1 The flowchart.....	26
2.4.2 The structogram.....	31
2.4.3 Linear notation.....	36
2.5 Structured programming.....	38
2.6 Creating a readable listing.....	41
2.7 Variables in Commodore BASIC.....	44
2.8 Mathematical functions and operators.....	52
2.8.1 Functions	52
2.8.2 Operators.....	58
2.9 Functions in high-resolution.....	62
2.10 Accuracy and speed of the C-64.....	64

Chapter 3 - Input and Output.....	71
3.1 The INPUT command.....	71
3.2 The GET command.....	73
3.3 Screen input and output.....	74
3.4 Handling data with the l541 disk drive.....	78
3.5 Relative files on the l541.....	80
3.6 Storing vectors and matrices.....	90
3.7 The PRINT USING subroutine.....	92
 Chapter 4 - Sorting routines.....	 95
4.1 The bubble sort.....	97
4.2 Linear sort.....	100
4.3 The shell sort.....	101
4.4 The quick sort.....	104
 Chapter 5 - Mathematical programs.....	 111
5.1 Zero-point determination.....	111
5.2 Romberg differentiation.....	116
5.3 Numerical integration.....	119
5.3.1 Simpon's Rule.....	119
5.3.2 Romberg Integration.....	122
5.4 Linear regression.....	123
5.5 Probability calculation.....	128
5.5.1 Binomial distribution.....	133
5.5.2 Chi ² distribution and adaptability test.....	141
5.6 Fourier analysis and systhesis.....	149
5.7 Numerical solutions to initial value problems.....	162
5.7.1 Comparison program for DEQ systems.....	162

5.8	Vector calculations.....	168
5.9	Matrix calculations.....	177

Chapter 6 - Programs for Chemistry.....183

6.1	Periodic system file program.....	183
6.2	pH value calculations.....	188
6.3	Titration.....	196
6.4	Thermodynamics of real and ideal gases.....	203
6.5	The HMO procedure.....	213

Chapter 7 - Applications in Physics.....236

7.1	Measuring time.....	236
7.2	Determining an isolation error.....	241
7.3	Optical geometry.....	250
7.4	Planetary Orbit Calculations.....	255

Chapter 8 - Computers in Biology259

8.1	The predator-prey model.....	261
-----	------------------------------	-----

Chapter 9 - The IEEE bus.....272

Chapter 10 - Technical programs.....275

10.1	Heat transmission.....	275
10.2	Pulley-length calculation.....	280

10.3 Analysis of complex networks.....	289
10.3.1 Complex numbers.....	289
10.3.2 Current and potential sources.....	300
10.3.3 Network current analysis.....	303
10.3.4 Node potential analysis.....	315
10.4 Measurement and control.....	325
 Chapter 11 - Computer Aided Design (CAD).....	333
 11.1 What exactly is CAD?.....	333
11.2 CAD with the Commodore 64.....	335
11.2.1 Three-dimensional representations.....	336
11.3 Creating a printed-circuit board layout.....	341

1. Chapter - Introduction

1.1 The Commodore 64 and Science

The Commodore 64 is leading the way as the most popular hobby and games computer. It is also being used in universities and schools for more serious applications. In universities the use of computers was previously limited to terminals and mainframes, while in lower levels of education, computers are just beginning to be used in the classroom.

Now that more programs are available which can be used in the natural sciences and engineering, computers are being used more often for solving problems as well as aiding researchers and engineers with simulations, computer-controlled equipment and software tools. The Commodore 64 offers an ideal chance to introduce previously uninterested people to the natural sciences because this machine is in very widespread use and supports impressive graphics and programming possibilities.

This book serves as an introduction to the use of computers in the natural sciences, engineering and mathematics. It provides a sweeping look into the work and thought processes of science.

Naturally, a good detail must be eliminated in order to cover such a wide range of topics in such short space. It is impossible to discuss everything dealing with science here and many individual topics require an entire book in themselves. With this book we hope to at least introduce the

reader to a variety of areas in which computers (specifically the Commodore 64) can be used in science.

A few words about the sample programs in this book. Graphics, sound, disk, and printer applications are not emphasized at all. The primary aim of these programs is to present the algorithms together with the scientific foundations by individual examples.

The sample programs are not intended as only isolated problem solutions, but should stir the reader to action himself. Value has been placed on well-documented listings and modular techniques. This means that you can easily integrate the program for linear regression in a program for determining reaction constants, for example.

At this point I would like to thank the people who have supported the creation of this book with their advice and deeds: Holger Jakobs with his tips for file creation, my brother Andreas, Hermann Schimtz and Juergen Thomas as well as Guido Schulwitz, and other current and former students at the University of Duisburg.

All that remains is to wish you a pleasant time on your excursion into the world of science and engineering, in which you will quickly learn that your computer can do more than just play Pac Man.

1.2 Assumptions

Every use of the computer presupposes certain knowledge of its operation. The first few pages of many books contain quite a terrible picture of the knowledge required to use the computer.

We've tried to keep these assumptions about the readers knowledge to a minimum. You should know how to use your C-64 in general: how to turn it on, to save and load with the disk drive or datasette, how to erase unwanted files, and so on.

You should naturally work through the Commodore User's Guide. One advantage of doing so is that you will get a fundamental knowledge of the BASIC programming language. You'll also gain some experience in the use of mathematical laws and numerical methods. But this should not be seen as the extent of the knowledge required to do scientific programming. We will first reveal many concepts through intensive work and second, we will document every program and the mathematics pertaining to it extensively so that no great programming problems should arise.

1.3 Equipment required

You will need a Commodore 64. In addition you will need some kind of external storage, either a disk drive or a cassette recorder (datasette) for storing programs and data.

You must also have a display screen, either a television or better, a monitor. Whether this device is color or not has no real significance for our purposes.

It's also a good idea to have a printer available. A printer is not assumed, however, it is very advisable in order to get your listings and data printed or perhaps even to get the data in usable form (such as tables with more than 40 characters per line). If you think about the effort involved in finding errors in programs, you will realize the value of a listing on paper.

Your computer set-up should consist of the following:

1. A Commodore 64
2. A VIC 1541 disk drive or a datasette
3. A monitor or television, such as the VIC 1702 (color)
4. Possibly a printer, such as the MPS-801, VIC 1525, or an Epson RX-80 (with an appropriate interface).

In addition you will need blank disks or cassettes and paper for your printer.

Do you have everything together? Then we can start!

Chapter 2 : Programming solutions to scientific problems

2.1 BASIC - Advantages and disadvantages

Using BASIC in the natural sciences and engineering seems like a contradiction in terms to many people. On one side, this microcomputer language lacks some features for solving mathematical problems. On the other hand, it is at least as widely used in these areas as are other languages, if not more so.

This is because of two reasons: First, BASIC is implemented in most small to medium-sized computers. Secondly BASIC is easy to learn, something which one cannot say about languages like FORTRAN or even Pascal.

Almost all programmers know how to program in BASIC. The disadvantages are not so obvious, because they often concern things which go unnoticed until it is too late. This is possible, for example, when a particular numerical accuracy is desired. You can see this problem if you enter PRINT 60-60.1 in the command mode. Surprised? And now enter the same thing in your pocket calculator. The calculator is more accurate.

These accuracy problems are a known weakness of most BASIC interpreters. They often do not affect the integer portion of numbers, only the fractional parts. These are simply rounded off by the C-64, which leads to errors in long calculations or expressions with complicated mathematical formulas. In the natural sciences, an important

change is often first noticed in the fractional part of a result, which makes this problem even more aggravating.

Unfortunately this is not the only obstacle which BASIC presents for the scientist or engineer. Structured programming is not supported by BASIC. The same applies for recursive techniques. By recursive we mean that a given subroutine can call itself. Such a possibility is permitted in Pascal, for example. But contrary to popular opinion, one can program recursively in BASIC, as can be seen in the following example. Enter this small BASIC program and RUN it:

```
100 INPUT A
110 GOSUB 1000
.....
1000 PRINT A
1010 A=A-1
1020 IF A>0 THEN GOSUB 1000
1030 RETURN
```

Now you have just executed a recursive routine. The C-64 allows a maximum of only 26 recursive calls, which means that a RETURN must follow this many GOSUBs. This limitation, which involves the stack, results from an internal routine.

But BASIC has a number of advantages to offer. Despite its many dialects, it is a very universal language. One can do virtually anything with BASIC. It also belongs to a group of interactive programming languages, which mean that BASIC has continual access to all program parts and lines, and can change these lines, set breakpoints, halt the program to examine variables and then continue again. This is not

possible in Pascal. Here we must first create our source code and then compile it. The compiled code cannot be changed, however--we need the source code to make any changes to the program. But this must first be loaded to make any changes to it. BASIC provides a complete programming environment.

Despite these advantages and the large, well-priced software library of BASIC extensions, it remains unclear whether software products such as Simon's BASIC, which improves the structured programming in certain areas, represents a solution to BASIC's limitations in the long run.

A better solution would probably be the intensive support of a modern language, which does not necessarily mean Pascal. There is a whole palette of high-level programming languages such as LISP or LOGO. Languages like FORTH and Ada must also be mentioned in this context.

The training in these languages is not particularly easy, but the work involved in learning these languages will pay off through the extended capabilities of each language which are all too easily left by the wayside when programming in BASIC.

But don't let yourself be irritated, concise programming is possible even in BASIC. It will take more concentration and consideration to create a passably structured BASIC program. The first step is to make the program listing readable.

If you are interested in controlling or measuring with BASIC programs, you must keep in mind the relatively slow speed of the BASIC interpreter. In addition, parallel processing (as in Ada) is not possible in BASIC. This presents a number of problems in reference to real-time applications.

2.2 The differences between various BASIC dialects

Most small computers are equipped with the Microsoft BASIC interpreter, but every microcomputer has a somewhat different command set in it's BASIC.

The graphics, sound, input/output, and editing capabilities are particularly effected. The first two are more a function of the operating system. The same applies to the I/O (Input/Output) with peripheral devices. These include the screen, the disk drive, the cassette recorder and the printer.

For this reason we will briefly explain the special differences between the operating system and other software (languages, etc.). We also want to go into the individual system levels and instruction types.

We shall begin with the various system levels of a computer. Every computer has what is called a hierarchy, which means that one level is ranked above several others (except at the bottom, of course). First a small table.

Table 1: System hierarchy

-
- a. Machine language level, direct programming access to the CPU.
 - b. Operating system, management of all functions of the computer, editing, I/O to other devices, etc.
 - c. Programming level, BASIC input or use of other languages

Here are three examples of the above areas of the computer:

STA \$43B0 -:Level a. The command stores the contents of register A in the memory location hexadecimal 43B0 of the computer.

LOAD"*",8 -:Level b. Loads the first program from the diskette in drive 0 into the Commodore computer.

10 PRINT "TEST":GOTO 10 -:Level c. Writes the word TEST on the screen and repeats this procedure until interrupted with RUN/STOP.

All of these levels can be accessed with BASIC, although it is difficult to use the machine language level from BASIC.

An easy way to access the machine language level through BASIC is using the instructions POKE and PEEK. We can then execute the machine language programs with the instruction SYS (address) or the command USR(x).

You must know exactly what you are trying to do and how the computer is internally organized if you want success. You can access many routines which perform quite specific tasks, such as disabling a key or turning the high-resolution graphics on and off. If you want to enter entire machine language programs, you should use a monitor or better yet, an assembler. These are the tools (programs) which allow you to work with machine language directly. With a disassembler or monitor you can easily view the contents of memory locations or the construction of a machine language program.

This book is not an instruction guide to machine language programming, so we recommend that the reader refer to one of the books dealing with this topic specifically. Two excellent texts in this area are The Machine Language Book for the Commodore 64 by Lothar Englisch and the book Programming the 6502 by Rodney Zaks. The last is written for the 6502 processor in general, but the 6510 processor in the C-64 contains the same instructions set as the 6502.

After this little excursion we now come to the main subject of this section, the varieties of BASIC. Differences occur in many areas. One major difference occurs in the use of PEEKs and POKEs. For example, you can store a zero in memory location 55376 with the command POKE 55376,0, which on the Commodore 64 causes a character at a specific location on the screen to turn black.

You can use this command only on the Commodore 64; it can have the same function only on computers with the identical organization of RAM and ROM. Naturally, you can enter this command on other computers, perhaps even without "crashing" the computer, but it is likely to have little effect.

The commands POKE and PEEK (PEEK returns the contents of an address) never have the same effect on different computers. The same applies to the system command SYS (address). If you have a listing which is not intended specifically for the C-64 and want to use it, you must investigate all of the PEEKs and POKEs in the program and make the corresponding changes for the C-64. There are also difference in the internals of machines made by the same manufacturer, the VIC 20 and the C-64, for instance.

Naturally, we cannot compare all of the BASIC dialects on the market--this would require a book in itself. We will limit ourselves to the wide-spread versions TRS-80 LEVEL II BASIC and APPLESOFT BASIC. We will concentrate on the most important differences, namely the graphics and the control commands for peripheral devices.

Many programs contain a large number of POKEs, PEEKs, and SYS instructions or use commands such as USR(x). It doesn't make much sense to try to convert programs like this because the amount of time required would be too great to make it worthwhile. Programs which can be easily converted are those which use mainly "standard" BASIC commands.

This applies in certain measure to the two most common computers in addition to the C-64, the Apple II+/IIe/IIc and the TRS-80. Applesoft BASIC is an enhanced version of CBM BASIC. There are a number of mathematical and scientific programs written for the TRS-80 that restrict themselves for the most part to standard BASIC commands.

There are of course differences. The principle differences involve input, output, graphics, and file management. The Apple has a whole package of graphics commands which are not found at all on the C-64. These are presented here so that you know where the compatibility between the Apple and '64 ceases.

Table 2: Apple graphics commands

```
-----
PLOT    HLIN    VLIN    SCR()    HGR

HCOLOR=  SCALE=  COLOR=  ROT=

HPLOT    DRAW    XDRAW    SHLOAD
```

The TRS-80 is not so extravagantly equipped, due to its lower-resolution graphics. It does have the ability to draw and erase points with SET and RESET. These and the Apple commands can be implemented on the C-64 only with sets of POKES and PEEKs or by using an extension such as Simon's BASIC, ULTRABASIC-64, or VIDEO BASIC 64.

Some computers have the instruction LINE INPUT or LINEINPUT (TRS-80 disk BASIC). This permits strings containing a comma or quotation marks to be read. The manner in which programs are loaded from the disk or cassette is also different. Programs on cassette are loaded into the Apple simply with LOAD (no name), on the C-64 with LOAD "name" and on the TRS-80 with CLOAD "name".

The command to read individual characters directly from the keyboard is also different.

Table 3: Reading the keyboard

```
Commodore 64:  10 GET A$ : IF A$="" THEN 10
```

```
Apple : 10 GET A$ : RETURN
```

```
TRS-80 : 10 A$=INKEY$ : IF A$="" THEN 10
```

The command implemented on other computers for formatted number output, PRINT USING, can be implemented on the C-64 as machine language or BASIC program subroutines. If you do not feel at home with programming, you can find information about such programs in magazines and books. In addition, there are books dealing specifically with the differences between the various versions of BASIC. Input and output of data over ports is generally done with the commands IN and OUT on computers using 8080/Z-80 processors, while it is done with the PEEK and POKE on the C-64.

In addition to the differences already mentioned there are deviations in the handling of strings and string arrays. The expression A\$(4) can refer to the first four characters of the string A\$, or it can mean the string with index 4 in the array A\$. One should therefore be careful when converting string expressions to Commodore BASIC. For such purposes we must use the expressions LEFT\$, RIGHT\$, and MID\$.

One can have a particularly nasty time trying to convert listings from computers such as the TI-99. This version of BASIC has syntax elements as CALL or ACCEPT at

its disposal. The CALL command calls a specific system routine of the computer, such as a routine to clear the screen. The ACCEPT command serves to input certain data types. In addition, our computer lacks the matrix operations which are so widely used in the natural sciences. All commands beginning with MAT fall into this category. A matrix (two-dimensional number array) is often read in with a command such as MAT READ, for example. Sometimes one finds expressions like DEFSTR and DEFINT (TRS-80). After DEFSTR A-D is encountered by the interpreter, all variables starting with A, B, C, or D are treated as string variables (throughout the entire course of the program). The same sort of thing applies for DEFINT A-D; these variables are treated as integers.

Also different is the error handling in Commodore BASIC. In other versions there are easy ways of catching an error with ON ERROR GOTO, GOSUB. If an error occurs, one can respond with an controlled error message on the screen without leaving the program. If the user produced the error through input, the program continues to run without interruption. One can also produce errors in order to move the computer to a specific program section.

In some BASIC dialects it is possible to set the accuracy of number representation. The command DEFDBL is used to switch to the double precision numbers with 16 significant digits. But rest assured: The C-64 is more accurate with normal accuracy than the TRS-80 is with double precision (because normal precision on the TRS-80 is 6 digits and all intrinsic functions are carried out in single precision).

Finally there are the commands which concern the printer and screen control. Many computers use LPRINT for printer output. Here we must first open a file with OPEN file number, device address, secondary address and then we can output to the printer with PRINT# command. The file must be closed after the output is complete.

Other machines often make use of the PRINT AT instruction for screen output. This directs the beginning of the output to a specific location on the screen. We do not have the command in Commodore BASIC. We can get by with some POKES, however.

A routine for positioning the cursor is best set up as a subroutine. The variable L could be used for the line number and C for the column number. Then one could assign values to these variables and call the subroutine to set the cursor at this location.

There are still more differences in BASICs, as many as there are dialects. Anyone who wants to work with this material intensively, or who must, can find comprehensive comparison lists in other literature.

2.3 Pascal and other languages

This book makes exclusive use of the programming language BASIC. There are however some programming languages which are specially suited for solving problems in the natural sciences. Some of these languages are also available for the Commodore 64.

2.3.1 Pascal - Structured problem-solving

The best known and most widespread high-level programming language next to BASIC is Pascal. There are several versions of Pascal available for the C-64. One is Pascal 64, a versions which also supports the special graphics capabilities of the Commodore 64. There is also a version of the well-known UCSD Pascal. This Pascal represents a standard for Pascal implementations on small systems. Versions of Pascal on 16-bit computers are generally oriented toward this standard. In the meantime however, there is a struggle to implement a standard for Pascal which everyone will follow.

What is the advantage of Pascal? The largest advantage is the ability to write highly structured programs and to use techniques such as recursion. While recursive programming is possible with limitations in BASIC, a structured BASIC program requires a certain discipline and experience on the part of the programmer. Furthermore, structured BASIC programs use large amounts of memory. When using Pascal, one is forced to write structured programs by the very nature of the language. GOTO statements are very

rare in Pascal (although they are there). Pascal is faster than BASIC and permits more accurate calculations (generally). The higher speed comes about because Pascal is a compiled language. This means that something called a source program is created using an editor, and this source program is then converted into ready-to-run machine code by the compiler. Any changes made to the code must be made at the source-code level, and not at the machine-code level. After the source has been saved, the compiler is loaded and it proceeds to compile the source code. The machine language program is saved and can be loaded and executed once the compiler is finished. The Pascal compiler itself is usually written in Pascal which increases its portability to other computer systems.

Not all versions of Pascal translate the source program to normal machine code as does Pascal 64. These compilers compile the source code into an intermediate code called p-code. This p-code is then interpreted by a run-time interpreter.

The next page contains a Pascal source listing for a bubble sort subroutine (procedure).

```
PROC (* b. sort *) sort(VAR...);

VAR i,j: integer;b: real;
BEGIN (* proc. b. sort *)
  FOR i:= 1 to n - 1 DO
    FOR j:= i + 1 to n DO
      BEGIN
        IF a(i) > a(j) THEN
          BEGIN
            b:= a(i);
            a(i):= a(j);
            a(j):= b;
          END
        END (* FOR *)
      END (* proc. b. sort *)
```

There are more interesting properties of Pascal. It is possible to define new data types in this language, for instance. There also additional data types implemented in Pascal like FILES and ARRAY as well as the type RECORD which belongs to the class of structured data types, as does ARRAY.

A few words about the history of Pascal. The language belongs to the Algol group. In fact, it uses many of the same syntax elements and structures as did the considerably older Algol 60. The developer of Pascal is the Swiss Niklaus Wirth, who has also written several excellent books on algorithms and structured programming. See the reference in section 2.4.

2.3.2 Ada - A programming language from the Department of Defense (DoD)

Another programming language built on Pascal is Ada. This language is very interesting because it is the official programming language of the United States Department of Defense (DoD). Ada is certainly a more powerful language than many of its competitors, but certain capabilities were left out of the language at the request of the DoD. Many programmers think Ada has become too extensive to use easily.

Nevertheless, Ada offers nested blocks AND procedures, in contrast to Pascal. Parallel processing is also possible and permits true real-time applications.

Ada should not be viewed as being in its finished form; changes are possible to its contents. Whether you believe it or not, Ada is already available for your Commodore 64! The only '64 version known to me is available from Abacus Software. This version is a training course for the Ada language.

2.3.3 FORTH - Creative programming

This is the ideal language for people for whom no command set is large enough or who do not like rigid syntax rules. FORTH is not a "normal" programming language like BASIC or Pascal whose command set cannot be expanded as desired by the user without knowledge of machine language. FORTH represents a totally new type of higher-level programming language.

FORTH is not really a new programming language; it was developed over a decade ago by an American astronomer. He sought a fast programming language with which to control a telescope. In the meantime however, this language has found friends all over the world. A FORTH interest group (FIG) has even been organized. This group is dedicated to the propagation and further development of FORTH.

FORTH is an interactive, stack-oriented language. You no doubt know what a stack is if you own an HP calculator. FORTH, like Hewlett-Packard calculators, uses "Reverse Polish Notation" (RPN) for its mathematical operations. In RPN, one enters the operators AFTER the operands, as opposed to standard algebraic notation or "infix" notation which places the operators between the operands. The infix-notation expression $(A+B) * (C-D)$ would be written as $A B + C D - *$ in RPN (also called postfix). FORTH does not use or requires parentheses.

The RPN arithmetic is not the object of FORTH's special position. The deciding factor here is the dictionary concept of the language. Every implementation of FORTH contains a relatively standard kernel of command words. The user can

then combine these basic commands to form new commands. These new syntax elements are placed in the dictionary and are available for use in the future. These new words can further be combined into more command words, and so on. This allows you to define a complex mathematical procedure simply with a succession of word definitions.

Unfortunately, it is also clear from what has been said what the major disadvantage of FORTH is. It is very difficult for anyone other than the programmer to read through a FORTH program. After a while, this also applies to the author of the program. If one is writing a FORTH program for others, comprehensive documentation is indispensable.

FORTH is also a very interesting language from another perspective. It essentially combines a compiler and interpreter in one package. In addition, there are FORTH implementations which contain an assembler as well. When a new command word is defined in FORTH, it immediately compiles into a table of pointers to the other command words of which the new word is made up. The new word is then interpreted when called. Only a relatively small number of command word definitions contain actual machine code.

It should also be mentioned that the space required by a FORTH system and FORTH programs is very small, especially compared with other high-level languages. The entire language, including compiler, fits into the computer at one time and does not absolutely require external storage media. On the other hand, the control of external storage media is not very well supported by FORTH since one must essentially write the necessary DOS (Disk Operating System) commands one's self.

2.3.4 LOGO and other languages on the C-64

There are other languages which should be mentioned here only in passing. LOGO is probably known to most computer users, at least in name. LOGO is a derivative of the LISP programming language which was developed as a language for artificial intelligence research. LOGO is used in computer-aided instruction and also as a first programming language for children. The turtle graphics is the most well-known Logo application. A small "turtle" can be moved on the screen and be made to draw on the screen with simple commands.

If you are interested in methods of computer-aided instruction and testing, you should also look at the language PILOT. PILOT is an unstructured language with few pretensions about the program-technical capabilities of the user/programmer. This does not mean that PILOT is a "dumb" language. One can write excellent teaching programs with it or question-and-answer tests very easily. PILOT is even better when combined with turtle graphics.

There are of course many more languages which could be mentioned, but we have now covered the most important.

2.4 From problem to algorithm

An algorithm is a mathematical statement which declares the relationship or pattern for a specific situation. The relationship is dependent on the initial quantities of the process. An algorithm can perform various processes depending on the initial quantities.

For example, a dish washing machine program has an algorithm. Depending on whether the water is turned on or not, the machine waits to perform the agitation and heating. The heater is turned on only until the desired water temperature is reached. When cold water is used, the heater is switched on for 10 minutes, but perhaps for only 3 minutes when hot water is used.

The algorithm is the same for both cases. It is: Turn the heater on when the water is in the machine and leave it on until the water reaches 140 degrees Fahrenheit.

This spoken instruction is an algorithm, but NOT a program. A program is always designated for a specific processor (not only for a certain language) so that it can understand the instructions. In the English language the algorithm would be for people who understand English. The people are then the processors.

You must first formulate your problem as an algorithm and then transform the algorithm into a program. This transformation is called coding. Many programmers use the "trial and error" method to develop the program and algorithm simultaneously on the screen. This creates

problems because one has no documentation after the development of the program and the programmer might not be able to understand his own program when a change or correction has to be made. Most programmers overestimate their own memories in this regard.

One can write an algorithm in a variety of ways. The first way is in English or any other human language. Human languages have the disadvantage that they are not always unambiguous and are above all very intricate. It is for this reason that program development aids were developed.

Three methods of algorithm expression have practical significance:

1. The flow chart
2. The structogram (Nassi-Schneidermann chart)
3. Linear notation (pseudo-code)

The development of an algorithm can be omitted for trivial programming tasks, but the time required to find an error-free algorithm for a task increases rapidly with large problems.

For large problems, the development of the algorithm is considerably more work than the transformation to code in the programming language used.

2.4.1 The flow chart

The oldest, most well-known method for formulation of an algorithm is the flow chart. The advantages of such a plan are to be found above all in the ease with which the representation can be understood. Unfortunately, this only applies as long as the algorithm is not too complex. For a comprehensive programming project, a flow chart leads to chaos because of the growing number of branches. It is also difficult to realize a structured method of writing. A flow chart can be quite helpful for small programs, however.

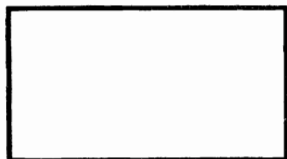
As a rule, the chart reflects the exact course of the subsequent program, without representing a particular type of coding. There are methods for creating a flow chart so detailed that almost anyone could write the corresponding program. In this case, all actions which are expected of the program are entered in the chart symbols, including the assignment of variables and mathematical operations.

What do the elements used in a flow chart look like? There are actually two types of symbols. We distinguish between the program-course plan and the data-flow plan. While the program-course plan documents, as the name implies, the course of the program, a data-flow plan reflects the path of the data through the data processor. One can then recognize when and where storage media are used, where the input and output through which peripheral devices occurs, and whether or not user intervention is expected. Let us take a look at the various elements of the flow-charting technique. The symbols given here correspond to the DIN 66001 standard.

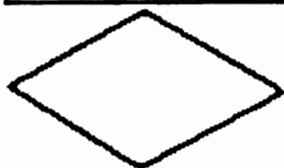
Flow Chart Symbols

a). Program operation

General operation

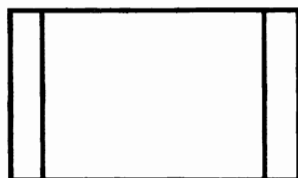


Program branch



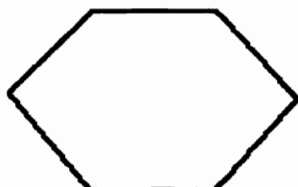
Subroutine

several Inputs/Outputs



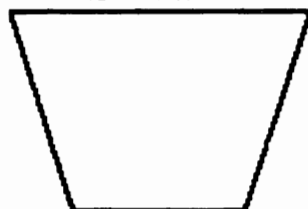
Program modification

memory changes



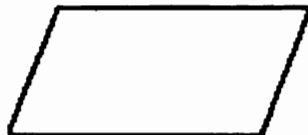
User intervention

.eg disk change



Input/Output

by hand or machine



Flow line



Reunion



Crosses indicate no reunion

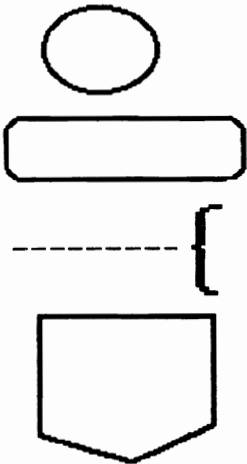


**jump to other diagram parts
(for large flow charts)**

Begin or End

comments use as desired

move to next page



b) Data flow chart

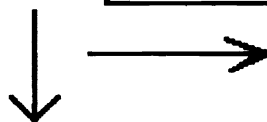
general operation



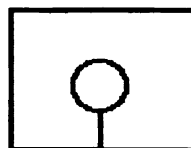
Input/Output



course directions



diskette



punch card

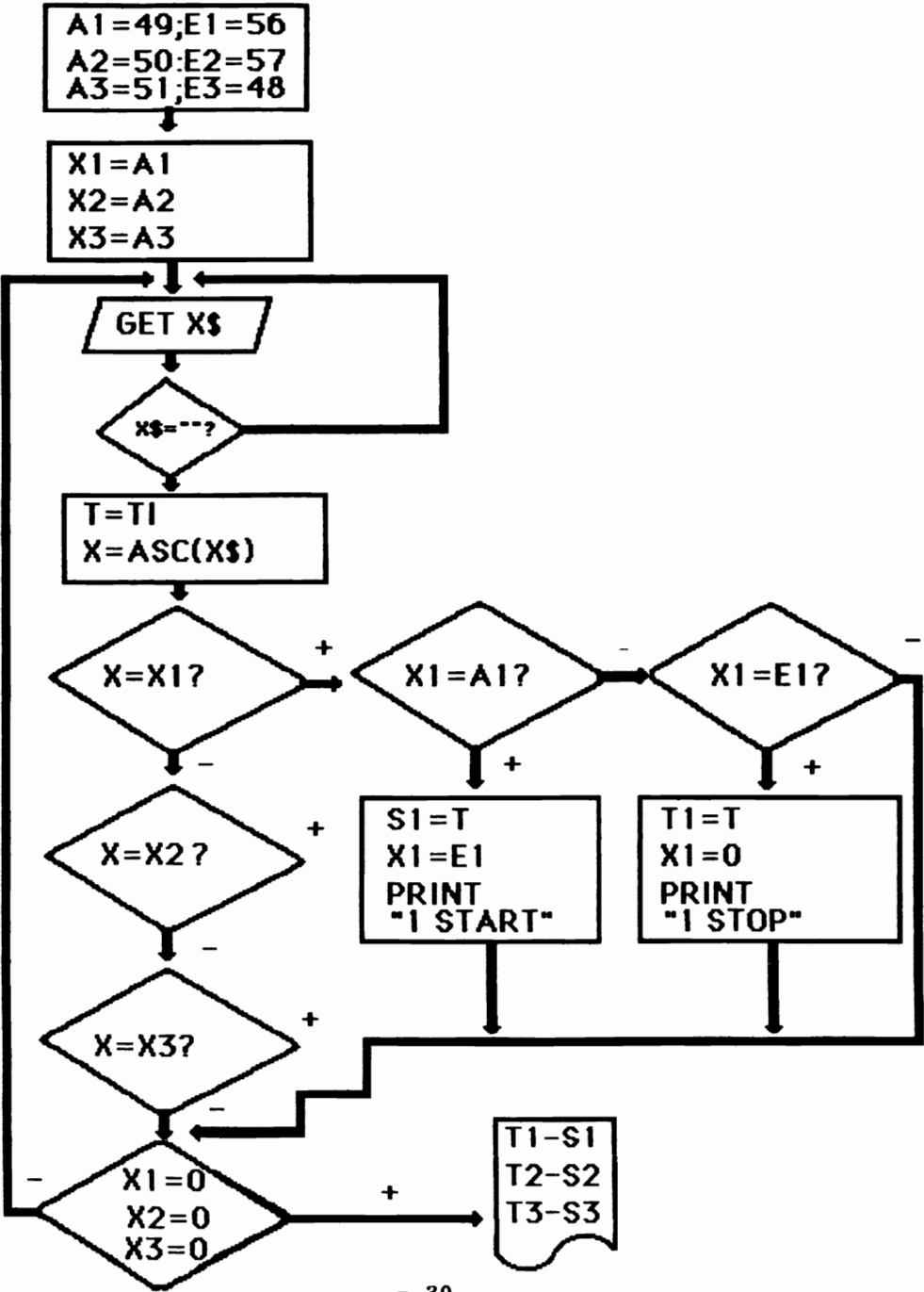


punch tape



data transmission





2.4.2 The structogram

In the beginning of the seventies, structured programming became more and more popular. Naturally, this was not as popular in the early home or hobby area than in the mini and mainframe computer areas. The limitations of the flow chart for the representation of such techniques were quickly recognized. There are no possibilities for the closed representation of loop and selection structures in the symbol set for flow charts. In order to get around this disadvantage, the notation initiated by Nassi and Shneidermann using structograms was introduced.

This method is directed exactly toward the requirements of structured programming. One uses structure blocks, although a standard symbol list as for flow charts does not exist. Elementary block types are named, however.

All structure blocks have in common that they can be accessed through only one entrance and exit point, corresponding strongly to structured programming. A complete structogram is composed of such blocks. We will now discuss the elementary block elements. At the end of the discussion you will find the figures mentioned.

Elementary block types (Nassi/Shneidermann)

Simple structure block:

Arbitrary processing commands described through the insertion of additional block elements and text in the block. See figure 2.4.2a.

Selection block:

Branches according to conditional criteria. All branches must reunite at the exit of the block. The individual branches therefore represent simple structure blocks themselves. There are two possibilities for the selection structure block. The first is the two-way branch (IF THEN...ELSE) and the second is the multiple-way branch (CASE...OF). See figure 2.4.2b.

Loop structure block:

Used for the representation of loops which are controlled by conditions. Two possibilities also exist here. The first corresponds to a loop which checks the condition before the loop is executed. In the second case, the criterium for continuation of the loop is checked within the loop. Here a decision about the continued execution of the loop is first made after part of the loop has already been executed. Correspondingly, one speaks of "pre-checked loops" and "post-checked loops." See figures 2.4.2c,d.

After the figures containing the elementary block structures you find the same program as in section 2.4.1, this time represented as a structogram. As you can see, structograms are always constructed from the elementary blocks, whereby it should also be clear that the logical compactness as well as the impossibility of subsequent changes forces structured programming. One must draw a new structogram to correct errors and to implement improvements in the algorithm. Another advantage should also be apparent: Such a diagram no longer stretches over several pages, which often leads to complications with a flow chart.

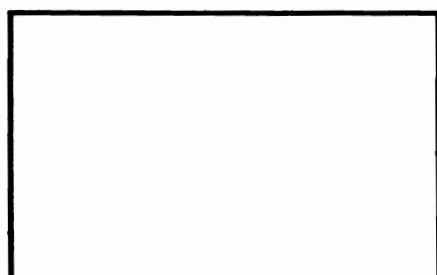


fig.2.4.2a

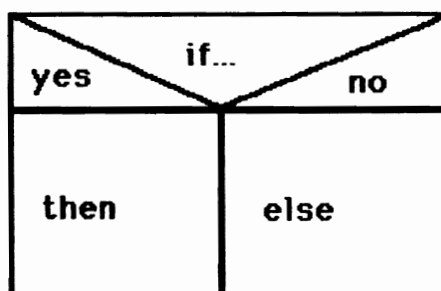


fig.2.4.2b

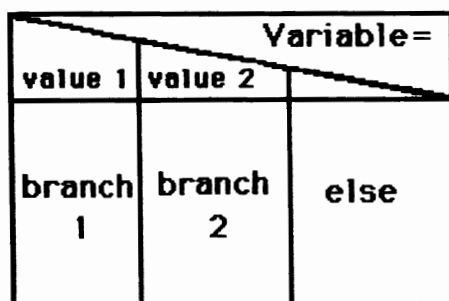


fig.2.4.2c

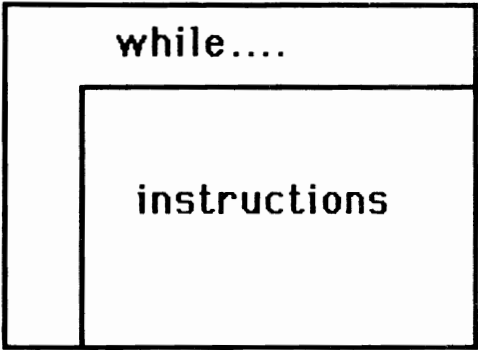


fig.2.4.2d

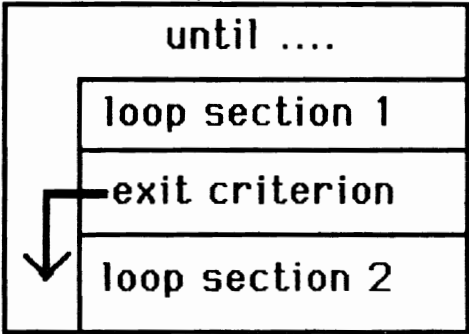
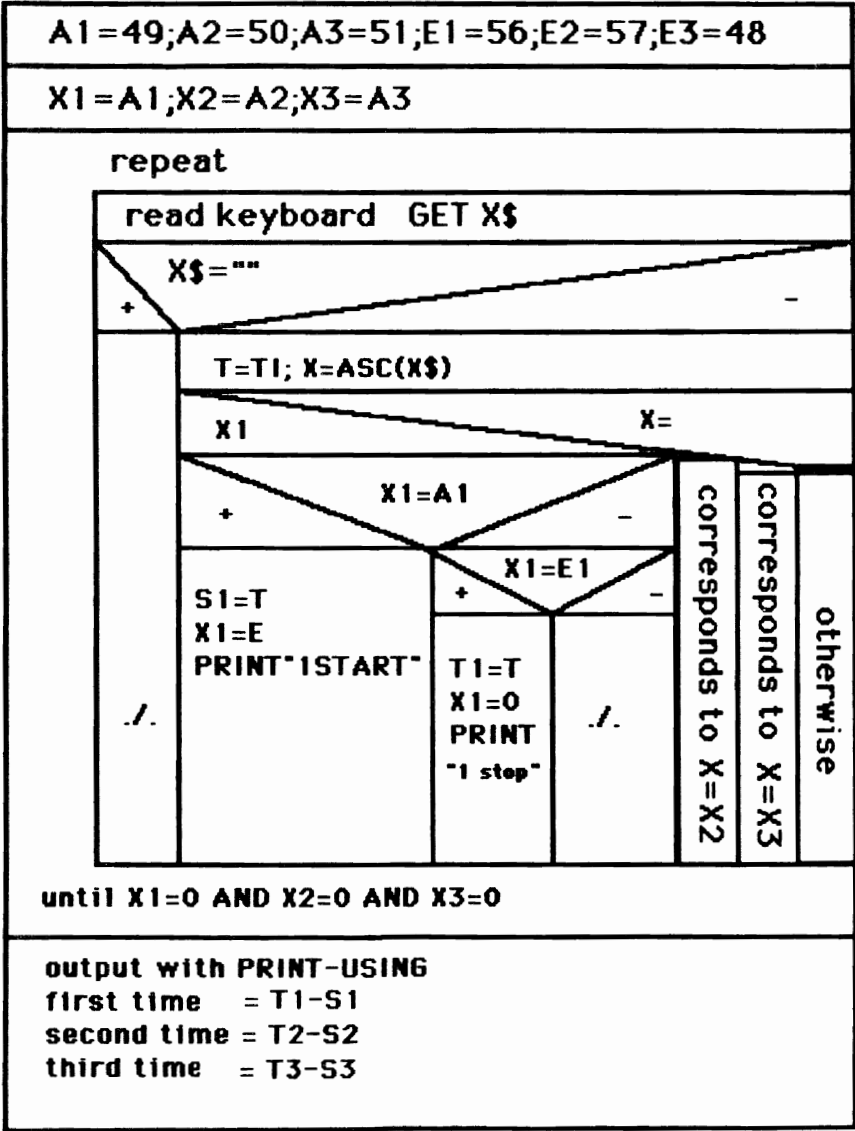


fig.2.4.2e



2.4.3 Linear notation

Pseudo-code is a notation in which all instructions are designated as sets of commands, one after the other. The result is a linear text. For the sake of clarity, algorithms in linear notation are ususally arranged graphically such that the structure levels are immediately recognizable.

Levels are, for example, various loops nested in each other. The pseudo-code can also be designated a language. It has its own vocabulary which can be changed to meet the needs of a specific case and should be because it need not be read by a machine which has an inflexible vocabulary.

Some symbols are precisely defined and should not be changed. The basic symbols are:

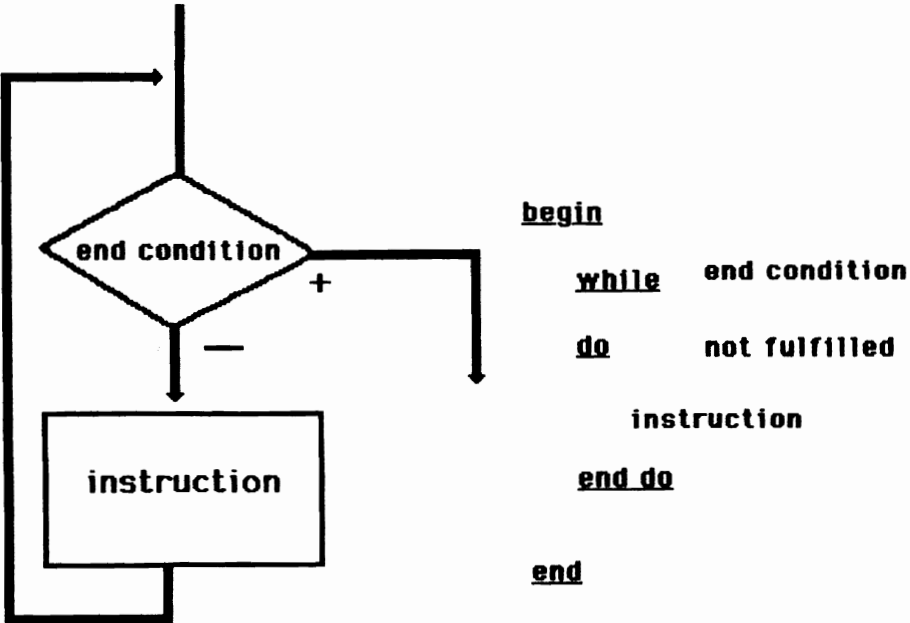
A to Z, a to z	letters
0 to 9	digits
+ - * /	arithmetic operators
OR AND NOT ^	logical operators
= >< ≥ ≤ ≠	comparison operators
()	calculation parentheses
[]	brackets for indexing
(* *) {}	brackets for comments
begin end	instruction block delimiters
:=	value assignment
if then else case of	instruction separators
while do repeat until for to	instruction separators

The vocabulary can be expanded as desired. One should clarify the extensions used so that others can take note of the new symbols.

The pseudo-code bears a great resemblance to Pascal. In fact, Pascal was developed from linear notation. Additional literature about linear notation can be obtained from a book by the author of Pascal, Niklaus Wirth. The book is called Systematic Programming.

It is significant that the often-used linear notation instruction "while ... do" is not present in BASIC and must be simulated. This loop instruction is one which is checked before the loop is ever executed so that if the condition is fulfilled, then the loop is never executed. With a FOR-NEXT loop in BASIC, the loop is always executed at least once. The condition is checked after the first execution.

In program-course plan and linear notation, this looks as follows:



2.5 Structured programming

This is not the first time that you have encountered the term "structured programming" in this book. This shows what significance this method of programming has today. As already mentioned, we cannot give a complete introduction to this area here considering the limited space. Some fundamentals should be discussed, however.

The primary intent of structured programming is the organization of the program as a block-structured object. This way we obtain a modular character of the individual parts of the program. Before we get into what such a block (also called a module) should look like, we must formulate a few rules to which every programmer should keep. This applies particularly to those who write programs for use by others. The requests which one must make of modern software are:

1. The program must fulfill its function without error.
2. The listing and with it the structure must be readable and clear.
3. The program should be user-friendly and not crash at every opportunity (never, if possible).
4. The program should be transportable to other systems without great changes (portability).
5. The program should fulfill its task in the least amount of time and with optimum results.

Unfortunately, the BASIC programming language is not oriented to the syntax of structured programming. There are here, for example, such structures as IF THEN ELSE (see 2.4), DO WHILE (condition), or CASE (2.4). Such possibilities for loop programming, for instance, cannot be realized in BASIC. However, some useful fundamentals of structured programming can be followed in BASIC.

When first starting to program, you should avoid going into too much detail. One should proceed according to the "top-down" programming model. Here we start with the problem as a whole and break it down, level by level, into smaller and smaller tasks, continually refining the algorithm until the final result is program code. Naturally, this process is interrupted by the testing of individual program parts. Now the block structure comes into play. Each block (in the sense of 2.4.2) can be checked separately.

How does one get a block structure of the program? For this purpose, the program to be solved is divided into smaller algorithms. These then represent the modules. Each individual module fulfills a very specific task. A block may have only one entry point and one exit point. A module may never be exited by means of GOTO, for example. Only GOSUB or ON ... GOSUB is allowed.

What do we do if we want to leave a block with GOSUB/RETURN? One possibility (in BASIC) would be:

```
1000 REM *** START OF THE LOOP ***
1010 .....
.....
.....
1040 IF (STOP CRITERIUM FULFILLED) THEN 1200
.....
.....
1080 GOTO 1000
.....
.....
.....
1200 REM *** END OF THE LOOP ***
```

This control block or module has only one entry point and one exit point. Arbitrary jumps are permitted within the block.

The creation of "readable" listings belongs to the general class of structured programming. We will concern ourselves with this in the next section.

2.6 Creating readable listings

The foundation of readable listings is the frequent use of REM statements. The idea is to divide the blocks of your program so that each closed module is clearly visible and identifiable in the listing. Comment the purpose of the block in the header and enclose this comment in a frame, perhaps of asterisks. Are you worried that you will lose a lot of memory space and that the listings will become too long? In this case you should consider that after a while you will not be able to read through a long program listing, even if you have written the program yourself.

When you write a program, you should try to comment it such that anyone else picking up a copy of your program would be able to understand it. Listings enhanced by informative REM statements are more important than the memory space which they use up. You should also not pack too many instructions into a line. When you separate program sections with REM statements, use a few more, for example:

```
100 REM *****
101 REM *                               *
102 REM *           OUTPUT           *
103 REM *                               *
104 REM *****
105 REM
110 PRINT ...
```

Construct your listing and your program such that a loosely-woven picture appears from which you can visually identify the individual sections of the program. Give the

sections widely differing initial line numbers such as 8000 for input and 10000 for output to further distinguish routines.

Indented lines also help a great deal to make a listing more readable. This can be done in Commodore BASIC as is illustrated in the following program fragment:

```
1000 REM *** SUBROUTINE ***
1010 FOR I=1 TO N STEP 10
1020 : PRINT "Y=", "X="
1030 : FOR J=0 TO 9
1040 : : K=J+I
1050 : : PRINT Y(K),X(K)
1060 : NEXT J
1070 : GET A$ : IF A$="" THEN 1070
1080 NEXT I
```

You have two options for indenting. The most useful is to insert colons in front of the instruction. This is what was done in the above listing and allows exact identification of procedures which belong to a specific loop in the program.

Another possibility is somewhat more complicated and has one small weakness. This method involves putting a "shifted space" in the line by pressing <SHIFT><SPACE> after entering the line number. After the "shift space" you can insert as many regular spaces as desired and then enter the actual BASIC line. When you press RETURN the line remains as you entered it, in listing and program execution mode. When you edit the line you must repeat the process of entering shifted spaces.

One more thing to watch out for when programming in BASIC is not to use variable names more than once for different things (except for general purpose temporary or index variables, for instance), especially in long programs. BASIC does not recognize local variables, variables which apply only to a given routine, as do languages like Pascal. All variables are global in BASIC.

If you really take these suggestions to heart and practice them in your programming, then you have taken an enormous step in the direction of a "clean" programming style and of structured programming a la Nikolaus Wirth.

2.7 Variables in Commodore BASIC

In order to run a program of any importance on our computer we must make use of variables. These have the same purpose as the variables with which we are acquainted from mathematics, that is, we can assign various values to a variable. In contrast to variables, all direct input to the computer is done by way of what are called constants, for example PRINT 23+45. While variable contents can change over the course of a program, constants remain the same.

When programming, various data types occur which we cannot handle in the same way. Therefore there are various variable types as well.

The data types which occur most often are numbers without a decimal point, numbers with a decimal point and places after the decimal (fractions), and non-numerical data such as text and characters. Mixed data of characters and numbers also occur.

All of these types of data are also found in the BASIC on our computer. Commodore BASIC can represent three different kinds of data:

1. Whole numbers or integers
2. Floating-point numbers or real numbers
3. Strings (of characters)

Integers are numbers without a decimal or fractional portion. They can be either positive or negative, achieved by entering the sign + or -. Examples are +672 and -7. If no sign is entered for an integer, the computer assumes that it is a positive number.

Integers on the Commodore 64 must be in the range +32767 to -32768. Smaller and larger values are not allowed. This is the result of the internal representation of integers in the computer.

Integers are stored as two-byte or 16-bit representations. Each bit is assigned to a binary or base two number. The largest integer which can be expressed in 16 bits is

$$11111111\ 11111111 = 2^{15} + 2^{14} + 2^{13} + \dots + 2^1 + 2^0 = 65536$$

This illustrates positive or unsigned representation. We want to be able to represent both positive and negative numbers. The Commodore uses a representation known as two's complement representation. In two's complement, the number is represented as a fifteen-bit binary number which is then complemented (all 1 bits changed to 0 and 0 bits to 1) and 1 is added to this complement. All sixteen bits are complemented so that the 16th bit acts like sign bit. If this bit is set (equal to 1), then the number is negative, otherwise positive.

Since the 16th bit serves as the sign, only 15 are left over for the actual number so that the range -32768 to +32767 is all that is left (actually the range 0 to 65536 has only been shifted down 32768).

Integers can also be handled as floating-point numbers. This is automatically done by the computer when it finds integers in arithmetic operations.

The most-used representation of number on a computer are floating-point or real numbers. Most of the input that is encountered either contains a fractional portion (places after the decimal point other than 0) or is not in the range -32768 to +32767.

Floating-point numbers occur in two forms in Commodore BASIC. The first is a decimal number with or without decimal places and the other is an exponential representation using ten as the base. It is this representation that interests us in particular and is often called scientific notation.

The range of possible floating-point numbers is also limited, of course. For similar reasons as for the integers, we must take certain limitations into consideration. The range of numbers at our disposal is quite large and includes all of the integer range, reaching from $\pm 2.93873588\text{E}-39$ to $\pm 1.70141183\text{E}+38$. This suffices for most scientific tasks, although there are quantities in physics and astronomy which exceed this range.

The computer automatically selects the exponential notation when the number is smaller than ± 0.01 or greater than $\pm 999,999,999$.

The exponential representation has the following construction:

Example 2.3E+4 : 2.3 is the mantissa

E is the designation for Exponent

+4 is the exponent of base 10

We get $2.3E+4 = 2.3 * 10^4 = 23000$

If you exceed the above range for real variables, you will get an OVERFLOW ERROR. If you go below it, the computer replaces the number with zero. There is no underflow error message.

If your data contains non-numerical elements such as text or symbols, it must be represented as a string of characters.

Strings are for example "TEXT", "text", "AB123" or "please enter:". As shown here, strings must be enclosed in quotation marks. Strings also have a limitation placed on them. You know that the maximum length of a program line is 80 characters. A string in this case must be less than 80 characters long. This applies only to keyboard input; a total of 255 characters are allowed internally.

You may use all numbers, letters, graphics characters, symbols and even the symbols corresponding to the screen control keys in a string. The only exceptions are RUN/STOP, RETURN, and RESTORE keys.

These are the data types which are available in BASIC on the Commodore 64. In order to now use these in variables, we must take note of a few things. Every data type is assigned to a corresponding variable type. We will begin with the integers or whole numbers.

Variable names for integers consist of two letters or a letter followed by a digit, followed by a percent sign (%) which declares the variable as an integer. If you enter more than two characters before the percent sign, the computer will ignore them. Variable names such as DIVIDEND% and DIVISOR% are both considered to be DI% as far as the BASIC interpreter is concerned. The % sign is required in any event. What was said here about the length of the name applies to the other variable types as well.

Remember that when reading listings from other computers that there are versions of BASIC which permit more than two significant characters in a variable name. Names like SUM1 and SUM2 are identical to the Commodore 64.

Reserved words are not allowed as part of a variable name in BASIC. Reserved words are BASIC keywords including command words like FOR and GOTO and names of intrinsic functions like SIN and COS. Note also that the contents of the variables TI and TI\$ change their value constantly. These two variables are used to access the C-64's internal clock. You may also not assign expressions to these variables or you will receive an error message.

The names of real or floating-point variables are used in the same way as integer variable names, except that the % sign is left off. The same applies here about the length of the variable name. Longer names may naturally be used, to better document the program. Just be sure that the first two characters of all variable names intended to be unique are different. Note that longer names take up more memory space.

Finally, string variables have a \$ after the name in order to designate the variable as a string variable.

Here are some examples of variable names:

Integer	Real	String

D*=2345	D=25.89	D\$="TESTSTRING"
FG*=5	FG=4.986	FG\$="HELLO"
K5*=12000	K5=23500	K5\$="Input:"

The above expressions are assignments of a value to a variable. Several variable types may occur simultaneously in such an expression. Caution need be used only for comparison and assignment operations. If you compare a real variable with a string, for instance, you get a TYPE MISMATCH error. Expressions like IF A = A\$ THEN ... are not possible, although a representation like A=FR * G% is.

There is one possibility for the use of variables which we have not examined. You have doubtlessly worked with matrices and vectors before. These are arrangements of numbers in which each number is clearly localized through its positioning within the arrangement. Such a matrix, which can have more than two or three dimensions, can be described through indices i, j, k, etc. Let us take a look at a two-dimensional matrix which we will call A:

$$A_{ij} = \begin{pmatrix} 2 & 4 & 5 \\ 1 & 6 & 3 \\ 9 & 1 & 7 \end{pmatrix}$$

Normally the numbering of array elements begins in the upper left-hand corner with 1,1. In Commodore BASIC, however, the numbering begins with 0,0. This matrix has $i=3$ columns and $j=3$ rows. The number 6 in this matrix is localized with $A(1,1)=6$ and the number 7 with $A(2,2)=7$.

We can use the same type of representation in BASIC. One speaks of indexed variables or arrays. In order to use such a variable in BASIC, we must first tell the computer to reserve space for the array. This is done with the BASIC instruction `DIM arrayname(i,j,k,...)`. This is called dimensioning the array. If the array contains a maximum of 11 entries, the computer dimensions it itself and one need not execute the DIM instruction. The maximum number of dimensions which an array may have is theoretically limited to 255 on the Commodore 64, although the memory can be completely exhausted by arrays with far fewer dimensions. If we want to express a matrix with 10 columns and 10 rows as an indexed variable, the dimensioning instruction would read `DIM A(9,9)`. This array can contain a maximum of 100 entries. An attempt to access an array element outside of the defined range will result in a BAD SUBSCRIPT error.

It is a good idea to explicitly define every array, even if it has fewer than 11 entries. Below you find a small program which reads a one-dimensional array and then outputs it; it is intended to serve as a demonstration of array techniques.

```
100 DIM A(25)
200 INPUT "HOW MANY ELEMENTS";N
300 FOR I=1 TO N
400   INPUT "ELEMENT =";A(I)
500 NEXT I
600 FOR E=1 TO N
700   PRINT A(I)
800 NEXT E
900 END
```

Indexed variables are also important because they permit data of the same type which belong together to be logically tied together and accessed through indices.

In conclusion we present a table of the memory requirements for array variables:

Array construction	Memory required

Name of the array	5 bytes
For each array dimension	2 bytes
For each integer element	2 bytes
For each real element	5 bytes
Strings	3 bytes
For each character in element	1 byte

2.8 Mathematical functions and operators

2.8.1 Functions

The Commodore BASIC version 2.0 offers a comprehensive set of built-in functions. These are also called library or intrinsic functions. These functions can be thought of as ready-to-use subroutines in machine language which are always present in the operating system. We are particularly interested in the mathematical functions.

These are:

ABS(X)	returns the absolute value of the argument
INT(X)	returns the integer portion of X by truncating the places after the decimal. The number is not rounded. For negative numbers, 1 is subtracted after the truncation. The function can only work with values between -32768 to +32767.
RND(X)	creates random numbers between 0 and 1. These are evenly distributed and calculated internally to 10 places.
SGN(X)	returns the sign (+, -, or 0) of X
SQR(X)	calculates the square root of X
LOG(X)	returns the natural logarithm of X
EXP(X)	is the opposite function of LOG(X). Returns the value of X as a power of e.
FNvv(X)	calculates the value of a user-defined function vv for the expression X. vv is an arbitrary variable name.

SIN(X) returns the sine of X. X is assumed to be in radians.
COS(X) as above, except for cosine.
TAN(X) as above, except for tangent.
ATN(X) calculates the angle with tangent X.

In some cases we must pay attention to certain characteristics of the functions in question. As a rule, the mathematical functions are oriented in their valid range to the conditions found in mathematics. In the sciences we generally deal with powers of ten and therefore with logarithms of base 10. If we want to calculate with these, we divide LOG(X) by LOG(10). Then we get the base 10 logarithm.

We begin to become aware of the C-64's lack of accuracy with the function INT(X). INT(X), as said before, does not round. If you get a value which you expect to 1, it may easily occur that the computer actually calculates it to be 0.99999998. You however do not notice any of this and you want to output this value. The INT function does not yield the integer 1 from this expression, but rather a zero! This messes things up for comparisons, for example.

Another limitation for many scientific applications arises from the lack of support for complex numbers. For instance, the logarithm of 0 using LOG(X) is not defined. An attempt to execute the square root function SQR(X) with a negative argument also leads to an error and the interruption of the program.

One very nice feature which is not implemented on other computers such as the TRS-80 without disk BASIC is the ability to write user-defined functions. You can define any function you like with the DEF FNvv(X) instruction and then call this later in the program with FNvv(X). This avoids having to call subroutines all of the time. Here is a typical example for the use of FNvv(X):

```
1000 DEFN FN F(X) = X*SIN(X)+5
1010 REM
1020 GOSUB 3000
      .....
      .....
      .....
3000 FOR I=1 TO N
3010 : A = FN F(I)
3020 : PRINT A
3030 NEXT I
3040 RETURN
```

We now come to a problem of a type which is quite easy to solve. Commodore BASIC contains only the angle functions SIN, COS, TAN, and the inverse of the last. The rest of the trigonometric functions (secant, cotangent, arccosine, etc.) can be made by a combination of the existing functions. The following table gives these equivalent functions:

Table of replacement trig functions

secant	$F = 1/\cos(X)$
cosecant	$F = 1/\sin(X)$
cotangent	$F = 1/\tan(X)$
arcsin	$F = \text{ATN}(X/\text{SQR}(-X*X+1))$
arccos	$F = -\text{ATN}(X/\text{SQR}(-X*X+1))+\pi/2$
arccot	$F = \text{ATN}(X)+\pi/2$

The same applies for the hyperbolic trigonometric functions; here is a selection:

Table of hyperbolic trig functions

sinh	$F = (\text{EXP}(X)-\text{EXP}(-X))/2$
cosh	$F = (\text{EXP}(X)+\text{EXP}(-X))/2$
tanh	$F = \text{EXP}(-X)/(\text{EXP}(X)+\text{EXP}(-X))*2+1$
coth	$F = \text{EXP}(-X)/(\text{EXP}(X)-\text{EXP}(-X))*2+1$

Additional equivalent equations for various functions can be found in the Commodore user's guide.

The arguments in the functions expressions discussed above may consist of arbitrary expressions. Care need only be taken that no undefined expressions are used. If such an error occurs, one truly misses an error handling routine such as ON ERROR GOTO. One can, however, simulate such a routine (called an error trap) in the following manner:

The error message vector of the C-64 lies at decimal address 768 and 769. In case of an error, this vector directs the computer to execute the routine at address \$A43A in the kernal. This address is responsible for outputting the error message. One can, however, change the jump destination with POKE to jump to a small machine language program which can trap a certain kind of error.

Here in an example:

First the program is loaded and executed:

```
100 FOR I = 49152 TO 49159
120 : READ A
130 : POKE I,A
140 NEXT I
150 REM * DATA CONTAINING MACHINE LANGUAGE PROGRAM *
200 REM
210 DATA 134,2,32,163,168,76,174,167
250 REM
300 POKE 768,0
310 POKE 769,192
320 STOP
```

The program is then erased with NEW. One then places the following lines at the beginning of a BASIC program:


```
10 LN% = line number : REM LN% is the line number which
                        will be jumped to in the
                        event of an error
20 POKE 20,(LN%/256-INT(LN%/256))*256
30 POKE 21,(LN%/256)
```

Admittedly, this method is a very complicated replacement for ON ERROR GOTO.

Another problem is the input of mathematical functions in a program without interrupting it. No version of BASIC known to us contains such a command. We can simulate this, the following subroutine allows the insertion of a function in a BASIC program at line 5000.

```
10000 REM *** FUNCTION INPUT ***
10010 REM
10020 INPUT "Function F=";F$
10030 PRINT"{clr}"
10040 PRINT"{2 crsr down} 5000 F=F$"
10050 PRINT"GOTO 12000 {home}"
10060 POKE 198,2:POKE 631,13:POKE 632,13:END
12000 RETURN
```

2.8.2 The operators

Now that we have heard some interesting things about functions, we want to take a look at the operators which BASIC 2.0 offers us. One distinguishes between three types of operators:

1. Mathematical operators
2. Comparison operators
3. Logical operators

1. The mathematical operators

Our BASIC offers the following computational operators:

- = equality character, assignment of a value to a variable, also used as a comparison operator (see below).
- subtraction and creation of negative arguments
- ^ exponentiation, ex. $2^3 = 8$
- * multiplication, $3*6 = 18$
- / division, $4/8 = 0.5$
- + addition

If several of these operators occur in a single expression, they are executed in a rigidly specified order:

1. Exponentiation
2. Multiplication/division
3. Addition/subtraction

Operators which are enclosed in parentheses are executed first, regardless of the above rules. These rules do apply within the parentheses.

Many BASIC interpreters cannot handle an exponential base of zero. The Commodore is not one of these. Instead of $(A-B)*(A-B)$, you can write $(A-B)^2$ without fear. Should the argument be zero with $A=B$, the value zero is returned. Be sure to place negative arguments, or arguments that may become negative in parentheses if you want to raise them to a power. A form such as $(A-B)*(A-B)$ is somewhat more accurate and in the case of squaring also faster than exponentiation.

If you want to process numbers with more than 9 places, the use of what is called parallel arithmetic is recommended. Information about this can be found in such sources as the CBM Programmer's Handbook. For high accuracy requirements the mathematical functions can also be formulated in machine language. Since we want to avoid a detailed excursion into machine language in this book, and also for reasons of program portability, we direct you to some tips along this line found in The Anatomy of the Commodore 64.

2. Comparison operators

The following comparison possibilities exist in BASIC:

Comparison operation	Example
= equal?	10 IF A = B THEN GOTO ...
< less than?	10 IF A < B THEN GOTO ...
> greater than?	10 IF A > B THEN GOTO ...
<> not equal?	10 IF A <> B THEN GOTO...
<= less than or equal?	10 IF A <= B THEN GOTO...
>= greater than or equal?	10 IF A >= B THEN GOTO...

The comparison operators serve as decision criteria in programs, to stop a program upon the occurrence of certain conditions, for example. The uses are clear so that we will not dwell on them here. We should warn our readers once again about the limited accuracy of the numbers in Commodore BASIC. It can happen that many number pairs never appear to be equal, although this is the case for all practical purposes. For example, .999999994 and 1 are not identical to the computer (indeed, they are not in a literal sense), even though the error in the first number may have been fabricated by the computer itself.

3. Logical operators

There are three logical operators available to us, namely:

AND OR NOT

These check a condition for "truth" and "falsehood." They allow multiple decision parameters to be checked at once, for example.

```
10 IF X = 0 AND Y = 0 THEN GOTO 200
```

Here both the first and the second conditions must be fulfilled in order to execute the jump. If we replace AND with OR, only one condition or the other need be fulfilled.

The NOT operation checks to see if a number (condition) is false or equal to zero. If the number in question is zero, the condition is fulfilled.

The logical operators use integers. If you are working with real numbers, these will automatically be converted to integers before the operation is performed. Logical operations are not usable on strings. They work only with 0 and 1, binary digits. This becomes clear in the boolean truth table.

Boolean truth table (0 = false, 1 = true)

A AND B:	1 AND 1 = 1	A OR B:	1 OR 1 = 1
	1 AND 0 = 0		1 OR 0 = 1
	0 AND 1 = 0		0 OR 1 = 1
	0 AND 0 = 0		0 OR 0 = 0
NOT A:	NOT 1 = 0		
	NOT 0 = 1		

2.9 Functions in high-resolution

Probably everyone has wanted to display graphs of functions on the screen with high-resolution graphics. Commodore has made this rather difficult to do, unfortunately. As you have certainly learned, you can only access the 64,000 screen points by having knowledge of the computer's internals and through the use of a number of cryptic POKE statements. The BASIC command set does not contain the slightest hint of support for the high-resolution graphics. The need to learn about the internal workings and construction of the machine may be fine for some, but for those of us who use the C64 for study or in the laboratory (yes, there are C64's in the laboratory), this is not good at all. Using high-res graphics in "normal" BASIC is as a rule too complicated and also too slow.

For these reasons we have avoided presenting high-resolution graphics in standard BASIC, except for a bar chart. These methods appear unreasonable for our purposes in the natural sciences. There are a fair number of graphic and BASIC extensions which make the Commodore similar to computers like the Apple II which has built-in graphics commands. In this connection, a few packages should be mentioned which appear particularly well suited.

Graphics aids for the Commodore 64

Simons' BASIC (Commodore) contains a comprehensive set of commands for graphics and sound in addition to commands for reading the A/D converters (paddles) and the function keys. Possibilities for structured programming are also available (REPEAT, etc.)

ULTRABASIC-64 (Abacus) offers capabilities similar to or Simons' BASIC but is more concentrated on graphics and sound. LOGO-like turtle graphics are also available.

Other products which support graphics do exist, but the ones mentioned here are the best-known and the most developed. The graphics routines in this book are written for Simon's BASIC. One may well assume that this extension is quite wide-spread.

If you are interested in graphics programs without Simons' BASIC or ULTRABASIC-64, books such as The Graphics Book for the Commodore 64 from Abacus software will be of great help.

2.10 Accuracy and speed of the Commodore 64

Perhaps this has already happened to you: You checked two variables for equality and the computer claimed that they were not equal. You then checked the entire calculation and came to the conclusion that the variables at this point simply must be equal!

By the rules of mathematics, you were probably right. The computer does not operate by rules of pure mathematics, however, it can only use floating-point arithmetic, and this arithmetic has a limited number of places of accuracy. For example, no periodic decimal fractions can be represented exactly with finite arithmetic.

It is probably best that the computer calculate with as many places as possible. How many places does the C-64 have? Nine! Oh well, the old calculator had ten. But this is better than other computers such as the TRS-80 models which have only six places. But wait! These computers also have double-precision variables. This makes them then significantly better than the C-64!

And then the speed of the C-64--only 1 MHz. Other computers have clock speeds which are at least twice as fast!

We have run a few tests. We ran the C-64 against a Video Genie I and obtained the following results:

First the accuracy. We calculated the formula

$$\left(\left(\left(\frac{1}{N} \right)^{\frac{1}{N}} \right)^N \right)^{-1}$$

The result through simplification of terms is simply the value of N. This is what the computers came up with:

N	Genie I		C-64
	single	double	
1	1	1	1
11	11	10.999984741210	11
111	110.997	110.99673461914	110.999998
1111	1110.77	1110.7668457031	1110.99946
11111	11088.2	11088.221679687	11110.9871
111111	107881	107881.375	111108.357
1111111	873082	8073082	1111052.35

As you can see, more places do not necessarily mean better accuracy. As a second check we calculated the sine of steadily-decreasing numbers. The sine approximates the line with slope 1 the closer it comes to the zero-point. We want to check this empirically with the help of the Video Genie I and Commodore 64 computers. It must be mentioned that the sine function on the Video Genie works only with single precision, so it doesn't help to use double precision.

$$X = 10^{-N}$$

N	Genie I	C-64
1	.00998334	.0841470958
2	9.99983 E-03	9.99983334 E-03
3	1.00003 E-03	9.99999833 E-04
4	9.99934 E-05	9.99999995 E-05
5	1.00181 E-05	9.9999894 E-06
6	1.02989 E-06	9.99998797 E-07
7	9.36268 E-08	9.99984511 E-08
8	0	9.9975593 E-09
9	0	9.97184394 E-10
10	0	0

It is obvious here that the C-64 has better accuracy here, and even that it has more places with which it can compute. Sooner or later, however, every computer will claim that the sine of a very small number is zero and not equal to or very close to that number.

Now the calculation speed:

The Commodore 64 required 0.9 seconds to calculate formula (1) ten times with the value 1 and the Video Genie took about 2.5 seconds for double-precision and about 2 seconds for single precision.

The Commodore needs 14 seconds to execute an empty FOR-NEXT loop 10000 times while the Video Genie requires 27 seconds. A loop which prints the numbers 1 through 1000 on the screen requires 45.5 seconds on the C-64 but only 28.5 on the Genie I.

To calculate SIN(I), SQR(I), and LOG(I) for I between 1 and 100 and display these on the screen takes 22 seconds on the C-64 and 18.5 on the Video Genie I.

Both computers required about three seconds to calculate the sines of integers between 1 and 100 and assign these to a variable.

As you can see, the '64 is fairly fast at calculations and also reasonably accurate. It should suffice in both categories for most applications. Only when screen output is required is it slower than most other computers. But then, how fast can you read? Certainly not as fast as the C64 can write.

You probably know that the VIC-1541 disk drive for the Commodore 64 is connected to the computer via a serial and not a parallel bus. Drastic reductions in data transfer rate must be made in order to use a serial bus, but for so little money one really cannot demand anything more. Remember that the disk drive is an "intelligent" device which offers many other advantages. There are also companies which have interfaced the 1541 to the IEEE bus so that it is structurally identical to the 2031. You must then purchase the IEEE bus cartridge in order to connect these drives.

A few more details about the handling of the "inaccurate" arithmetic of a floating-point processor:

Never test two floating-point variables for equality! Instead, check to see if their difference is less than some small number, say $1E-8$. You must test in each individual case how small the difference must be for the two quantities

to be considered equal. Remember that a maximum of 9 significant places are available and after accounting for errors, only 7 or less are available.

For purely integer operations you can also use integer variables. The value range is limited to +/- 32767. On the Commodore 64, however, even integers should not be tested for equality because the C64 has no integer arithmetic but instead internally converts integers to floating-point values before working with them. The conversion also takes time of its own, so we have in general avoided integer variables.

Significant places can eliminate each other! If calculate the difference of two large numbers which have a very small difference, this difference will have very few significant places. An example wil help to clarify this:

```
1000004
- 1000002
-----
      2
```

The result has only one significant place. It will become still clearer if you want to find out if the associative property holds:

Six-place floating-point arithmetic is assumed.

```

100004
+      5.95000
-----

```

```

100009

```

```

    ^^^^^

```

truncated places!

```

100009
- 100002
-----
      7.00000

```

Calculated the other way around, whereby the associative property should be demonstrated:

```

10004
- 10002
-----
      2.00000

```

```

      2.00000
+      5.95000
-----
      7.95000

```

As you can see, this cancellation of significant places can have disastrous consequences. Pay attention, when possible, to the order of calculation so as to have arithmetic operations working with values of similar magnitude in order to reduce this type of error.

You can also not be sure that the index values in a FOR/NEXT loop will be exact. Try a step size of 0.1 once, outputting the index values on the screen, and see what happens. You will be surprised. Again, do not test for equality!

Chapter 3 : Input and Output

The input and output with the Commodore 64 is handled differently than with other computers. The use of the cursor keys within strings which will be PRINTed is very unusual, and confuses many users at first.

3.1 The INPUT command

Variables are read from the keyboard with the INPUT command. The INPUT command prints a question mark on the screen and places the cursor behind it. The user cannot determine whether a number or string must be entered. You must provide this information yourself in the question. The question can be given in the INPUT command like so

```
15000 INPUT "Size";S
```

Unfortunately, no variables may be output in the INPUT statement. One cannot ask for the nth element of a vector. For this you need a PRINT command before the INPUT. All numbers are preceded by a space if they are positive or by a minus sign if they are negative. A space is always printed after the number. In order to print something directly behind the number, the cursor must be moved one location to the left. "<-" represents the cursor left key.

```
15000 PRINT R;"<-th element";:INPUT X(R)
```

The semicolon inhibits the otherwise automatic carriage return/line feed.

One positive point about the input of non-numeric data when a number is specified is that the program does not stop but rather issues the message "REDO FROM START" and the input must be re-entered. It is therefore unnecessary to check to see if what is supposed to be a number contains any non-numeric characters.

When you ask for a yes/no response or for a number which for which there is a standard default, you can pre-formulate the most common answer so that the user usually need only press RETURN. Otherwise the user simply types an alternate value. This is done with the help of the cursor left key again. This character is symbolized by a "<- " here.

```
15000 PRINT "ONCE MORE  N<-<-<-<-";:INPUT X$
```

The question mark generated by the INPUT command is placed after "MORE" and the N is under the cursor as if it had already been typed in and the cursor moved back one character. A "Y" for yes can be entered as if the "N" were not there since it will replace the "N".

The INPUT command can unfortunately not be abbreviated. It must always be typed out in full.

3.2 The GET command

The GET command bears a certain resemblance to INPUT, but it reads only one character at a time and does not wait for a key to be pressed. The computer can be made to wait for input with a loop which waits until something other than an empty string has been read (the GET command returns an empty string when no key has been pressed).

```
15000 GET A$ : IF A$="" THEN 15000
15010 B$=B$+A$ : IF A$=CHR$(13) THEN 15030
15020 GOTO 15000
15030 ...
```

The check for CHR\$(13) causes the loop to be ended when the RETURN key is pressed.

3.3 Screen output and input

As we mentioned in the introduction to this chapter, the cursor control keys are used for output on the screen. Unfortunately they are not so easily recognizable as such in the program listing because they appear as inverse letters or special characters. Type a quotation mark and then press the cursor keys, the CLR/HOME keys, and the INST/DEL key. This is how these keys are represented in the program listing. Make note of these as they are important for controlling screen output.

Color control can also be incorporated into a string. One proceeds here exactly as above. Pressing a color control key after a quotation mark results in another inverse character which represents that color.

Even turning the inverse mode on and off, which is not at all visible when used directly is represented as inverse characters in a string. What these characters look like depends on the selected screen representation--upper/lower case or graphics/upper case.

Switching to the other character set within a program is achieved with CHR\$(14) and CHR\$(142). The manual switching can be inhibited with CHR\$(8) and re-enabled with CHR\$(9). Additional possibilities in this vein can be found on page 135 of the user's guide.

The inclusion of the CLR/HOME key in the output is especially important in subroutines since the subroutine does not know where the main program left the cursor. The

subroutine can then set the cursor in the upper right-hand corner of the screen by using the HOME key, or it can clear the screen as well with shifted HOME.

The PRINT AT command of other computers can be simulated with some POKE commands and a SYS call in the following manner:

```
POKE 211,column : POKE 214, line : SYS 58732
```

Dimensioned variables, vectors and matrices, are generally printed element by element. This should look as follows on the screen:

```
A(1)=14  
A(2)=23  
A(3)=7
```

Unfortunately, all positive numbers are printed with the preceding space. Space should be left for the sign of a negative number, but this blank should be suppressed for positive numbers. Also, a space always follows the numbers. This is reason for the spaces before and after the index between the parentheses. There is only one way to avoid this. The index must be converted to a string from which the first character is then removed. In a program this would look like this:

```
100 PRINT "A(";RIGHT$(STR$(I),LEN(STR$(I))-1);")=";A(I)
```

instead of

```
100 PRINT "A(";I;")=";A(I)
```

The above version requires a certain calculation time, which slows the output. This is more than outweighed by the better-looking output because of the C64's high processing speed. This method also works with output to the printer.

An important instruction, especially for those who write their own programs and work on them alone:

Annotate all output!

Even when you know exactly what is in the output table when you are working with the program, it is easier to look at the head of a table to find out what is printed in a given column. Furthermore, you will grow less and less familiar with your programs as time goes by. Think about it!

The TAB command is of great help for tabular output. It is similar in action to the tab key on a typewriter. It causes the next output to appear in the column specified. Unfortunately, all numbers are left-justified when printed and the PRINT USING command is missing as well. This command can be replaced, however, using a subroutine. The listing for this subroutine is given in section 3.7. You must assign the value to be printed to the variable PX, the desired number of places after the decimal to S1, and the column number to S2.

The TAB command does not work in lines which have been controlled with a cursor key or "clr/home". If you must use tabs on lines such as these, you should use the direct cursor positioning with POKE 211 and POKE 214 as explained earlier. The TAB command also does not work on the first line to which this PRINT AT simulation jumps. You can get

around this by jumping to a line one higher than required and then issuing a PRINT command which contains a line feed/carriage return. Then you can tab as usual.

Note: When used on a printer, the TAB command functions like the space command! On the MPS 801, a CHR\$(141) can be used to generate a carriage return without a line feed. Then you can use TAB or SPC to achieve the tabulation by printing the line in multiple passes. Otherwise you must convert the numbers to strings which have the same number of characters in front of the decimal point with the help of the PRINT USING program for printers. This method is used in the program for determining isolation errors, for instance.

3.4 Data handling with the VIC-1541

The saving and loading of programs is discussed in detail in the user's guides for both C64 and 1541 disk drive, so I will limit the discussion here to the transmission of data.

Use only file numbers between 1 and 127. You cannot open more than 10 files at the same time. With file numbers higher than 127, each data record ends a line feed/carriage return combination, which is not necessary for disk files. This can be suppressed as outlined on page 19 of the disk drive manual, but it is easier to simply avoid the higher file numbers.

Whenever possible, it is a good idea to select the same number for both the file number and the secondary address.

When saving with the PRINT# command, be sure that each individual input contains a CHR\$(13) (=RETURN). This occurs automatically when one sends a data item to the disk with a PRINT# command with no trailing semicolon or comma. If a semicolon follows the PRINT list, the next item output will be combined with or joined to the previous item so that when reading the data in again there is no way of telling where the first item ends and the second begins. This may be reasonable for strings, but it certainly is not for numbers.

If you want to write several data items to the disk with a single PRINT# command, place a semicolon and a CHR\$(13) behind each item:

```
110 PRINT#1,A;CHR$(13);B;CHR$(13);C
```

A CHR\$(13) after C would be superfluous since it is included automatically with the PRINT# command. Note that when reading the data later that you do not try to read alphanumeric data into a numeric variable. If you do try to do so, you will receive a SYNTAX ERROR without line number and the program execution will stop.

As long as the stored strings are not longer than 88 characters, they can be read in with a simple INPUT#. With longer strings you must GET# and check after each byte is read to see if it is a CHR\$(13) to test for the end of the string.

It is not a good idea to use the comma as a separator between data items because this always inserts 12 spaces between the items. These spaces will be interpreted as zeroes with numeric data. The decimal points will also be interpreted as zeroes, making it difficult to distinguish.

If a sequential file must be read several times in succession, you need only close it and open it again. This puts you back at the beginning of file. If you want to read a particular item, you must first read all of the previous items. Individual items cannot be accessed directly in a sequential file.

3.5 Relative files on the VIC-1541

As a use of a disk drive you are no doubt acquainted with the format of the directory listing. In the last column it lists information about the type of the file stored. You have probably seen the abbreviations SEQ and PRG there.

There are two more types of files, however, USR and REL files. The first is easily unmasked since these are nothing more than sequential files whose names appear in the directory differently. These too can only be written and read sequentially and it is not easy to simply add data to the end of a file. We can get around this defect fairly easily, however. We build a new file containing the data to be appended to the first and then combine these two files into one new file which then contains the entire set of data. This goes like this, for example:

```
10 OPEN 3,8,3,"NEW FILE,S,W"  
20 PRINT "NEW DATA:";:INPUT N$  
30 PRINT#3,N$  
40 CLOSE 3  
50 OPEN 15,8,15,"COPY:TOTAL FILE=OLD FILE,NEW FILE"
```

Line 10 creates the new file while line 20 gets the new data from the keyboard. Line 30 takes care of writing this new data in a new file and line 40 closes the new file "NEW FILE".

The command sequence in line 50 combines the new file with the old file "OLD FILE", creating the file "TOTAL FILE" which contains the contents of both files. To delete the files, you must address this separately, such as:


```
60 PRINT#15,"SCRATCH:OLD FILE,NEW FILE"
```

At the end you should close file 15 with

```
70 CLOSE 15
```

Through this procedure you can avoid having to read the entire file into memory in order to expand it with new data. If the file is too large for available memory, the read and write files would have to be open simultaneously and each line read would have to be immediately written back out again--a considerable expenditure of time.

We now come to relative files, which can be managed by the 1541 disk drive and even by its predecessor, the 1540. We should first clarify what we mean by relative and random files because they are not the same thing by any means.

A random file is a file with free access. All the blocks of this file have a length of 256 bytes, exactly the length of a sector (block) on the diskette. In order to effect a read or write access of the file, the track and sector numbers must be given. This can be calculated from the running record number, although this is somewhat complicated. This is because the tracks have different numbers of sectors. An example of this can be found in the program "random file example" on the 1541 TEST/DEMO diskette.

In contrast to random files, relative files have an arbitrary record length, although this length may not exceed 255 bytes. The individual data records are directly addressable via their record numbers and are available in

seconds. New records can be added at any time and old records can be easily rewritten. Parts of records can even be read and changed. There is no longer a difference between read and write files with relative files: There are one and the same.

It is unfortunate that the BASIC V2.0 in the Commodore 64 does not contain any commands which directly support the REL files. These can be found only in the 4.0 BASIC of other Commodore computers or in diverse extensions of the BASIC standard, such as MASTER-64 from Abacus Software. But what can we do if we have neither of these?

Simulation is the key word! The RECORD# command, for example, can be replaced by a command sent over the command channel with secondary address 15. If you want to open relative files, you use an L and a number for the length of the data records instead of the S, W (S=sequential, W=write). You cannot open files with a length of 42, 58, or 63 bytes per record. This causes a SYNTAX ERROR in the DOS. In addition to the record length you can also specify where in the record you want to "land." A sample program will illustrate this:

```
10 PRINT "RECORD LENGTH:";:INPUT RL
20 OPEN 1,8,2,"RELATIVE,L,"+CHR$(RL)
30 OPEN 15,8,15: REM *** COMMAND AND ERROR CHANNEL ***
40 PRINT "RECORD NUMBER:";:INPUT I:IF I=0 THEN 400
50 PRINT#15,"P"+CHR$(2)+CHR$(I AND 255)+CHR$(I/255)
   +CHR$(1)
```

The "P" here stands for positioning the pointer and the "2" in the first CHR\$() stands for the relative file with secondary address 2. The record number "I" is converted to binary notation. The AND between two numbers executes a logical AND between the individual bits of the binary representation. In other words: X AND 255 returns the least-significant byte of the binary number X, while the most-significant byte can be obtained with X/256. This division naturally returns a fractional portion but this is automatically truncated by the CHR\$ function. The following chart illustrates the logical AND to obtain the least-significant byte:

	256	128	64	32	16	8	4	2	1

255 =	0	1	1	1	1	1	1	1	1
300 =	1	0	0	1	0	1	1	0	0
AND	0	0	0	1	0	1	1	0	0

The fact that the most-significant byte can be calculated by dividing with 256 should be clear. The second and third CHR\$()'s then transmit the I and the drive positions over the desired record. The CHR\$(1) positions the read/write head over the first byte in the record. Any integer up to 255 is possible as long as the record is long enough.

```
60 INPUT#15,S,B$
70 IF S=50 THEN PRINT "NEW RECORD":GOTO 140
```

The error channel (same as the command channel) is read in line 60. If it transmits message 50, the corresponding record does not exist. The program prints an appropriate message and jumps to line 140. It is not necessary to ask if the record should be read or written since a non-existent record can naturally not be read.

```
80 IF S <> 50 AND S <> 0 THEN PRINT "ERROR";S;B$
```

Line 80 checks for other errors. Message 0 means that no error occurred. Possible errors include inserting the wrong diskette, errors on the disk, and a write protect on the disk.

```
90 PRINT "READ OR WRITE (R/W)?"  
100 GET L$ : IF L$="" THEN 100  
110 IF L$="R" THEN 240  
120 IF L$="W" THEN 140  
130 GOTO 100
```

Line 100 reads the keyboard. If no key is pressed, the program remains in line 100, otherwise it continues with 140 or 240. The following lines write the string I\$ to file 1. Since the pointer is positioned on the record with number I, exactly this record is written. If more write accesses are made to file 1 without repositioning the pointer, the drive will always write to the next record. This makes sequential processing possible without repositioning each time.

```
140 PRINT "CONTENTS OF THE RECORD";:INPUT I$  
150 PRINT#1,I$
```

The lines 160 through 290 which follow take care of reading the contents of the record. Note: A maximum of only 88 characters can be read in with INPUT#. For longer records GET# must be used to read byte by byte.

```
160 GOTO 40
240 INPUT#1,I$:PRINT"RECORD #:";I;"CONTAINS:";I$
250 PRINT "PRESS C TO CHANGE, OTHERWISE RETURN"
260 GET V$:IF V$="" THEN 260
270 IF V$=CHR$(13) THEN 40
280 IF V$="C" THEN 300
290 GOTO 260
```

Changing the record contents can be done in lines 300 and 310. Since one can also access parts within a record, the start byte is asked for. Normally it will be the first because this is the only way to change the whole record.

Line 320 asks for the new contents which will be written in the record at byte number BY; the actual writing takes place in line 330.

```
300 PRINT "CHANGE AT WHICH BYTE";:INPUT BY
310 PRINT#15,"P"+CHR$(2)+CHR$(I AND 255)+CHR$(I/256)
    +CHR$(BY)
320 PRINT "CONTENTS OF THE RECORD";:INPUTI$
330 PRINT#1,I$
340 GOTO 40
400 CLOSE 2:CLOSE 15:END
```

If the record number 0 is given in line 40, the program jumps to line 400 which closes the files and ends the program.

If you want to divide a record into several fields, you can specify that the second field begins at position 25 (the 25th character), the third at position 35, and so on, and then set the pointer for each field or send several sections, separated by RETURNS, together as a record to the drive. You can also combine the two by sending sections of a set length separated by RETURNS. Then you can read the fields with the INPUT# command and also pick out individual fields with the byte pointer.

```
500 INPUT F1$:INPUT F2$:INPUT F3$
510 PRINT#1,F1$+CHR$(13)+F2$+CHR$(13)+F3$
```

The CR behind F3\$ is automatically included when no semicolon follows. The length of fields F1\$ through F3\$, including the three CRs, may not be longer than the preset record length or else the error 51, OVERFLOW IN RECORD will occur.

Note that you cannot change an individual field within a record without also rewriting all of the following fields as well. So don't forget to first read all of the following fields of the same record and also rewrite them. Otherwise this information will be irretrievably lost.

ISAM FILES

Unfortunately, the C64 cannot manage ISAM files (Index Sequential Access Mode) in which direct access to a record can be made not only by record number but also through the contents of a field in the record, the index field.

You can create these yourself, however, and manage them with the help of your program. The access times are, with appropriate organization and especially for large files, noticeably shorter. For this purpose it is a good idea to use a larger disk drive such as the 8250 which can be attached using an IEEE interface. The data transfer rate for these drives is five times higher than that for the 1541 and the storage capacity is more than five times greater per diskette.

In order to be able to access a record via an index field, the index fields must be saved separately in an index file. It is possible and advisable to organize this file sequentially for small sets of data. This way one only has to search the small index file, which can be completely read into memory, instead of the large main file which usually won't fit into memory. With sequential organization of the index file, the record number in the main file need not be stored as well since it corresponds with the running numbers of the indices. The trick described above can be used for the index file when expanding the data file.

If the data base is just too large to make reading the entire index file into memory possible or sensible, then the index file should also be organized as a relative file. Here the record number in the main file must be stored along with the index field. The records in the index file are divided into two parts. The index file is then sorted by the contents of the index field. The sorting can be running during input or can occur after the input or expansion is completed.

When a search is made for a specific record, it is first checked to see if the field being searched for is greater or less than the index field in the middle of the file. This comparison functions for both numeric and alphanumeric data. The second comparison is made in either the upper or lower half of the file, that is, for N records the second comparison will be made with either the $N/4$ th record or the $3/4*N$ th record. In each case the number of records remaining to be searched is cut in half. This type of search is called a binary search.

The advantages of a binary search are evident only for large files because the number of comparisons necessary increases subproportionally to the number of records, or more exactly, with the base 2 logarithm of the number of records. In order to find one record out of 20, five comparisons are needed, for 500 records, 9 comparisons, and for one million, 20 comparisons.

In every case, be sure to pay attention to the clarity of your key field. The chemical symbol for an element is a clear key with which the index file management can exactly determine the element. This book contains an example using the periodic system of elements which uses relative files.

With this method of data access, there is also the option of using several different index files, of course. This allows you to perform searches on different index fields without having to store the entire file multiple times or resort it. It is with such multiple accesses and large files that this file management method shows its true qualities. For small applications the advantages do not warrant the large effort on the part of the programmer.

The management of a data base is also very involved. Since when changing a record or inserting a new one, one or more index files may be concerned, ALL index files must be recreated, which usually takes quite a while. When one only changes a file in the index file by which the search was made, the record number in the index file is also known and the change can be made directly to the index file as well as the main file. Then only a resort of the one index file is necessary.

When many changes of various data records are necessary, one can first read all of the changes and then recreate all of the index files at one time in order to save time. One must make sure that the search algorithm still functions properly after the first changes so that it still works for the next record to be corrected. This means that one will not be able to avoid a temporary storage. One must accept this as part of the price for the reduced update time.

3.6 Storing vectors and matrices

In this book there are many programs which must process large sets of data. Sometimes the results of one program will be used in another. It would be very complicated, time-consuming, and error-prone to copy down all of the intermediate results from the screen or print them out and then type them all in again.

This is particularly unbearable for matrix operations. Therefore one should reach an agreement on how vectors and matrices will be stored so that programs can exchange such data between themselves. We present one possibility here, which we do not propose as THE solution, but which at least will ensure compatibility.

We use sequential files for vectors. The first data element in the file is the length of the vector. Following this are the vector elements, separated from each other by carriage returns (CHR\$(13)). In order to designate the files as vector files, the first two characters of the filename is always "v ". The following code saves the vector under the name contained in A\$:

```
OPEN 2,8,2,"V "+A$+",S,W"
PRINT#2,N
FOR I=1 TO N
PRINT#2,A(I)
NEXT I
CLOSE 2
```

A similar procedure can be used for matrices, in which one first saves the number of rows and columns and the matrix elements, row by row. Assuming that the matrix name is contained in A\$, the filename used in the OPEN command is "M "+A\$+",S,W".

For very large matrices in which one needs only single elements, relative files are to recommended. The record number can be calculated from the indices and then directly accessed. This also works for arrays of more than two dimensions.

Above all when testing programs still in development, it makes a great deal of sense to store the test data in a file so that the results remain comparable. This also removes the necessity of having to retype the data each time the program is rerun.

3.7 The PRINT-USING subroutine

Unfortunately, the C64 does not possess any built-in commands for formatting numeric output. This is very important for tabular data. There are various machine language subroutines for simulating the PRINT USING command which certainly run fast but which must always be loaded separately. In order to be able to include a PRINT USING command in a program without requiring a BASIC loader for a machine language subroutine, we have developed a PRINT USING subroutine completely in BASIC.

This naturally slows down the output, but we feel that the convenience of the BASIC subroutine and indeed of the formatted output more than make up for the lack of speed. When outputting to the printer, the program can easily keep up.

An explanation of the algorithm and its limitations:

The program converts the variable PX into a string with the name PX\$. This string then has S1 places after the decimal point and S2 in front. Excessive places after the decimal will be truncated, missing places will be filled with zeroes. Rounding is not performed, but this is easy to build in later by multiplying PX by 10^{S1} in line 20020, adding 0.5, taking the integer portion of this, and dividing again by 10^{S1} .

For numbers between -1 and +1, a zero will be placed in front of the decimal point. When a number is internally slightly smaller than 1 or slightly larger than -1, but appears as only 1 or -1 on the screen, it can happen that an

additional leading zero will be included with an even 1 or - 1. This seldom happens, however.

Numbers which are represented internally on the computer in scientific notation cannot be represented with this routine. This defines the value range of the subroutine. The largest representable number is +/- 9999999999, and the smallest is +/- 0.01.

If you want to format the output of numbers in scientific notation, use the program with the name PRINT USING EXP. Note that it requires more time to execute since it is correspondingly more complicated. In this version the mantissa is formatted according to the given information and the exponent is written behind as usual.

The execution time of this version can be considerably reduced by assigning STR\$(PX) to a string variable so that it need not be generated three times. But we wanted to keep the number of variables used in the program down to an absolute minimum. If you like, you can change this as you see fit.

After returning to the calling program, PX\$ must be output instead of PX, either to the printer or to the screen. With identical formatting, it is no longer a problem to print a column of numbers so that the decimal points all line up under each other.

Note that the program uses the variables PX, PX\$, S1, S2, PP, DP, NU\$, and II. The program should not use these variables to store any information which will be used after the subroutine call.

In various programs in this book you will find condensed versions of this routine, some of which display only on the screen. The REMs will make the differences clear, so we will forego additional explanation here.

```

20000 REM *****
20005 REM *          PRINT-USING EXP *
20010 REM *****
20015 :
20020 PX$=STR$(PX):NU$=".000000000"
20025 :
20030 IF ABS(PX)<.01 OR ABS(PX)>999999999 THEN PX$=LEFT$(P
X$,LEN(PX$)-4)
20050 PP=LEN(PX$):DP=PP+1
20060 IF ABS(PX)<1 AND ABS(PX)>.01 THEN PX$="0"+RIGHT$(P
X$,PP-1)
20070 IF PX<0 AND PX<=-.01 THEN PX$="-"+PX$
20080 IF -1<PX AND PX<0 THEN PP=PP+1
20090 FOR II=1 TO PP:IF MID$(PX$,II,1)=". " THEN DP=II:NU$=
"000000000":II=PP
20100 NEXT :PX$=PX$+NU$:PX$=LEFT$(PX$,DP+S1)
20110 FOR II=1 TO S2-PP+2:PX$=" "+PX$:NEXT II
20120 IF ABS(PX)<.01 OR ABS(PX)>999999999 THEN PX$=PX$+
RIGHT$(STR$(PX),4)
20130 RETURN

```

```

20000 REM *****
20005 REM *          PRINT-USING *
20010 REM *****
20015 :
20020 PX$=STR$(PX):PP=LEN(PX$):DP=PP+1:NU$=".000000000"
20025 IF ABS(PX)<>1 AND PX<>0 THEN PX$="0"+RIGHT$(PX$,PP-
1)
20027 IF -1<PX AND PX<0 THEN PX$="-"+PX$:PP+1
20030 FOR II=1 TO PP:IFMID$(PX$,II,1)=". " THEN DP=II:NU$="0
0000000":II=PP
20040 NEXT:PX$=PX$+NU$:PX$=LEFT$(PX$,DP+S1)
20050 FOR II=1 TO S2-DP+2:PX$=" "+PX$:NEXT II
20060 RETURN

```

READY.

Chapter 4: Sort routines

Methods for sorting data are particularly important for natural science programs which are devoted to the management of measured values. Sorting programs serve this purpose. Most of the programs are used as subroutines. Several procedures have become particularly widespread. We want to take a closer look at the most well-known sorting routines later.

There are certain elementary examples for sorting algorithms. The following procedures are used, listed according to the method used. The ordering also gives the approximate efficiency of the methods:

- a. Sorting by insertion. Examples: linear sort and ripple sort.
- b. Sorting by exchange. Example: bubble sort.
- c. Sorting by selection. Example: shell sort.
- d. Special procedures and mixed types. Example: quicksort (mixture of insertion and selection).

We will assume that the data is numerical, that is, that they consist of numbers and are stored in an array. There are better methods than those introduced in this section, but these are either customized for special cases or are considerably more complicated. In addition, there are also special variants for sorting files on external storage media such as the floppy disk.

The most well-known sorting methods are the linear sort, the bubble sort, the quick sort, and the Shell-Metzner sort. We will explain these methods quickly because these procedures have fundamental significance. In this context we will also give a speed comparison of the various methods. We will restrict ourselves to non-recursive techniques. Furthermore, the function SWAP, which exchanges the contents of two variables, is not available to us in BASIC 2.0. We will avoid recursive methods because of the poor support which BASIC lends to such techniques. It should be mentioned, however, that recursive programming in these cases is normal in other programming languages such as Pascal. Pascal can in any event be recommended for completing mathematical problems. But back to our BASIC. After the individual presentations of the algorithms mentioned, you find the comparison program which illustrates the various time requirements of each of the methods.

You can build the sorting routines into your programs as desired. This is useful, for example, if you have written a program which is supposed to draw a curve from a set entered data. These values can be placed in increasing order according to their size by a sort routine and the plot can be carried out continuously.

It certainly can't hurt if you try to improve any of the sort algorithms. An increase in speed can in most cases be obtained. Now to our first example, the bubble sort.

4.1 The bubble sort

The bubble sort is probably the best known of all sort methods. It is possible to program it incorrectly, however, and thereby waste a good deal of time (as we will see).

How does this method work? The bubble sort is not only the best-known, but also the most primitive sorting algorithm. The procedure is as follows:

1. Find the largest element in the value set.
2. Exchange the largest element with the first.
3. Repeat steps 1 and 2 until all elements are in the proper order.

The program consists for the most part of two nested FOR/NEXT loops. The outer loop moves from 1 to the number of data to be sorted minus 1. The inner loop goes from the current value of the outer loop plus 1 up to the maximum number of values. In this manner, all of the smaller elements constantly move toward the start of the value set. The smaller values "bubble up."

Although the bubble sort is easy to understand and to implement, it is unfortunately too slow for large sets of data. This results from large number of passes through the loop which must be made. We can recognize this with the help of a formula. The program contains a case decision of the form:

```
IF X(I) > X(J) THEN ...
```

The number of times the above instruction is executed can be obtained from the following calculation:

$$(n+1)+(n+2)+\dots+2+1 = n(n-1)/2 = c \text{ (comparisons)}$$

In other words, the number of executions of this instruction increases with the square of the number of data elements to be sorted, n . But, as we will see, we can do without this instruction. Here is the listing of the simple bubble sort:

```
30000 REM
30010 REM *** ROUTINE BUBBLE SORT ***
30020 REM
30030 FOR I=1 TO N-1
30040 : FOR J=I+1 TO N
30050 : : IF X(I) < X(J) THEN 30090
30060 : : B = X(I)
30070 : : X(I) = X(J)
30080 : : X(J) = B
30090 : NEXT J
30100 NEXT I
30110 RETURN
```

We want to improve the bubble sort a little by changing the previously-mentioned line. The modified listing then looks as follows:

```
30000 REM
30005 REM *** IMPROVED BUBBLE SORT ***
30006 REM
30010 FOR I=1 TO N-1
30020 : B=X(I):K=I
30030 : FOR J=I+1 TO N
30040 : : IF B<=X(J) THEN 30060
30050 : : B=X(J):K=J
30060 : NEXT J
30070 : X(K)=X(I)
30080 : X(I)=B
30090 NEXT I
30100 RETURN
```

This routine uses the same principle as the first example, "sort by exchange." The direction of the sort can be determined, either increasing or decreasing, by changing the case decision in line 30040.

But why is the second version of the bubble sort faster? The lies simply in the fact that during a pass through the outer loop, only an exchange of X is made. The routine notes whether or not the current value was already exchanged. The temporary variable B is used for this purpose. The maximum number of operations is thereby reduced to $(n-1)OP$.

4.2 Linear sort

This procedure follows the principle of "sorting by insertion." By this we mean a procedure which one might use when sorting file cards by hand. One compares and moves to the left if the adjacent card has a lower value. This way, elements which are already at the right place are not moved. This procedure is very fast if the data is already sorted or almost in the appropriate order. It requires the most time when the set is in the greatest disorder (order opposite to that desired). The following listing demonstrates the linear sort:

```
30000 REM *** LINEAR SORTING ***
30010 REM
30020 FOR I=2 TO N
30030 : A(0)=A(I)
30040 : B=I-1
30050 : IF A(0) <= A(B) THEN GOTO 30090
30060 : A(B+1)=A(0)
30070 : B=B-1
30080 : IF B <> 0 THEN 30050
30090 : A(B+1)=A(0)
30100 NEXT I
30110 RETURN
```

As you can see, the linear sort is not exactly the ultimate sort. In fact it has no great significance. The above-mentioned bubble sort is considerably more convenient.

The total number of maximum operations comes to $OP = (n*n+3*n-4)/2$.

4.3 The Shell sort

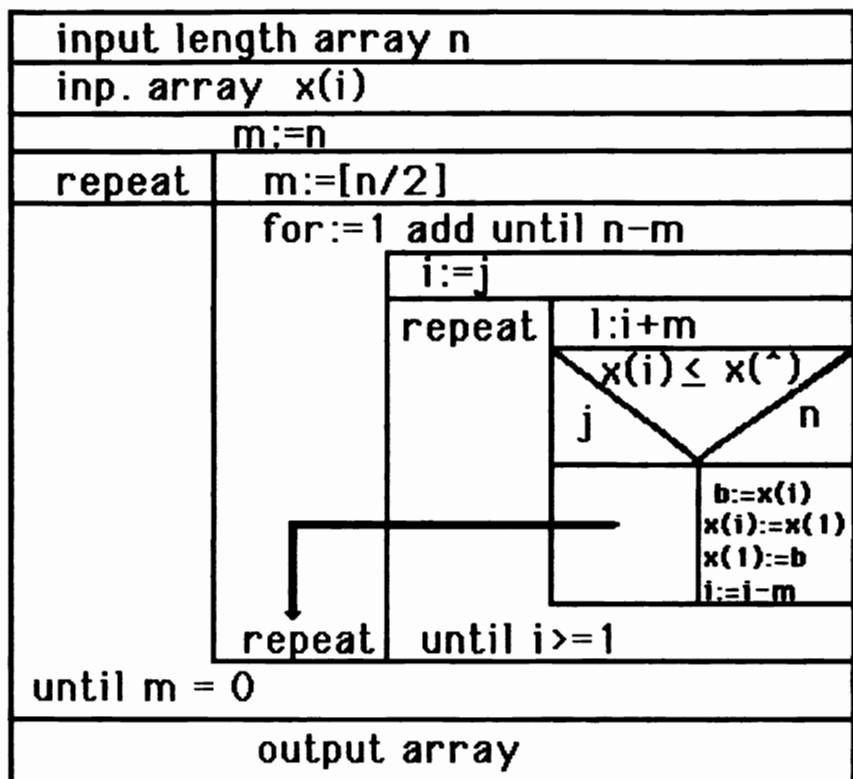
A better, if somewhat more difficult to understand algorithm is given in the next example, the Shell or Shell-Metzner sort.

The Shell sort was introduced in 1959 by a Mr. D.L. Shell. With this method we no longer proceed with a "direct" algorithm as in the previous examples. The data to be sorted will not be compared with neighboring values, but rather over great distances.

The Shell sort actually represents a refinement of the insert algorithm. We will therefore not repeat an exact examination of the procedure. Niklaus Wirth, in his book Data Structures and Algorithms, which we have already mentioned, raised some mathematical questions concerning certain surprising properties which remain unanswered yet today. For example, it remains unclear why the most convenient step size of the procedure in regards to speed cannot be predicted.

A introduction to the principle of the Shell sort is probably done best with the following structogram. The algorithm consists mainly of comparing the data of the value set to be sorted over a constantly decreasing distance and exchanging when necessary. Slowly one reaches the minimum distance possible and adjacent elements are compared and exchanged. Then the procedure is done and the values are (hopefully) sorted.

Here then is the structogram:



The step size is generally set at half the length of the data array for lack of a better way of calculating it. This means that for n data we have a step size of n/2. One can beforehand calculate with a proportional value of $n^{1.2}$ in known cases, in the negative case one reaches at most $n^{3/2}$.

Before we come too the end of our selection with the quicksort, we will first present the listing of a Shell sort:

```
30000 REM
30010 REM *** ROUTINE SHELL SORT ***
30020 REM
30030 S=N
30040 S=INT(S/2)
30050 IF S=0 THEN 30180
30060 D1=N-S
30070 FOR J=1 TO D1
30080 : I=J
30090 : S1=I+S
30100 : IF X(I) <= X(S1) THEN 30160
30110 : H=X(I)
30120 : X(I)=X(S1)
30130 : X(S1)=H
30140 : IF I > 0 THEN 30090
30150 NEXT J
30160 RETURN
```

4.4 The quicksort

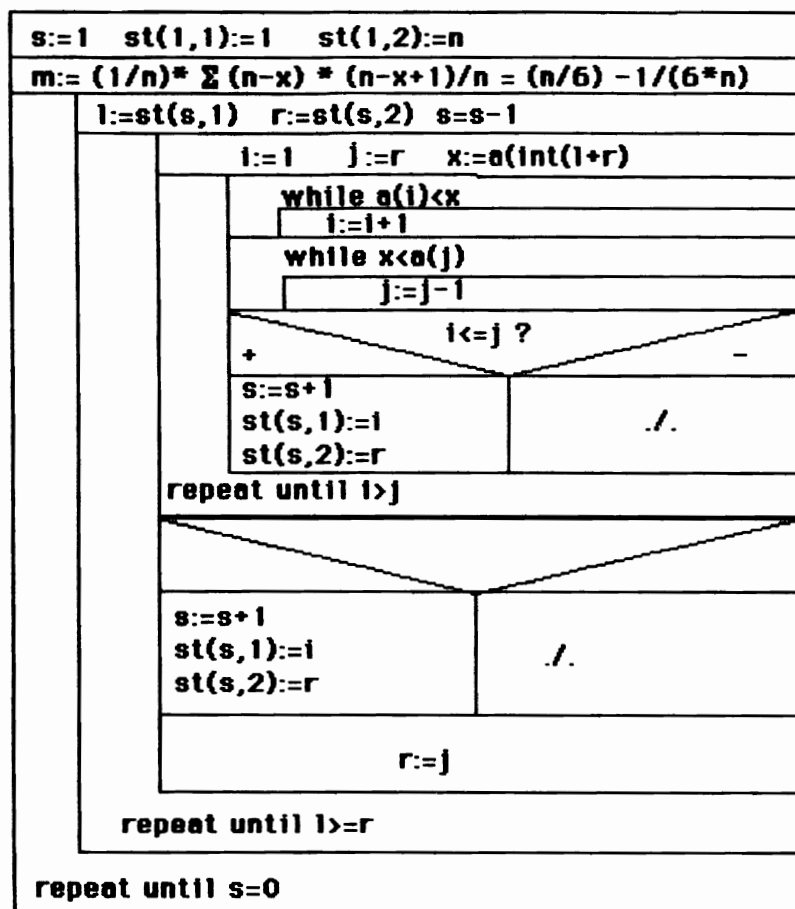
The fastest method of putting greatly disordered data in the desired form is through the quicksort. It represents a further refinement of the bubble sort. It shows off its qualities best when the data to be sorted are as disordered as possible. One sometimes hears the name Hoare sort, after the name of inventor of the method.

We will sketch the procedure for the quicksort; it is not easy to understand. Here again the comparisons are made at a greater distance away. This is a characteristic of higher-level sorting algorithms. In contrast to the Shell sort we proceed from the outside to the inside. The comparison value X we give therefore has significance. The average element of the array is most often chosen. In the following array

34 56 12 33 78 90 21 10 8

this would be the number 33 since it is the median. This is the one of n values which is both greater than or equal to one half of the set while less than or equal to the other half.

Those who wish to learn more about quicksort and especially how to find the median value are referred to Wirth's book mentioned above. Below is again the listing, given this time in the form of a structogram because of the complexity of the algorithm.



quicksort structogram

```

10 REM *****
20 REM *
30 REM *      COMPARISON PROGRAMM      *
40 REM *
50 REM *      SORT ROUTINES            *
60 REM *
70 REM *****
80 REM
90 POKE53280,0:POKE53281,0
100 GOSUB 15000:REM * INPUT              *
110 GOSUB20000:REM * X(I)=Y(I)          *
115 PRINT"(GRY2)IMPROVED BUBBLE - SORT{C/DN}{C/DN}"
120 GOSUB 1000:REM * BUBBLE I           *
130 GOSUB20000
135 PRINT"SIMPLE - BUBBLE - SORT{C/DN}{C/DN}"
140 GOSUB 2000:REM * BUBBLE II          *
150 GOSUB20000
155 PRINT"SHELL - METZNER - SORT{C/DN}{C/DN}"
160 GOSUB 3000:REM * SHELL-METZNER *
170 GOSUB20000
175 PRINT"QUICK - SORT{C/DN}{C/DN}"
180 GOSUB4000:REM * QUICK-SORT          *
190 GOSUB20000
195 PRINT"LINEAR SORT"
200 GOSUB5000:REM * LINEAR-SORT         *
230 REM
250 GOSUB10000:REM * BAR CHART*
300 REM
310 GOSUB12000:REM * OUTPUT TIMES
320 REM
400 PRINT"(CLR)REPEAT ?"
410 GET A$:IF A$="" THEN 410
420 IF A$="Y" THEN RUN
430 IF A$="N" THEN END
440 GOTO 410
1000 REM
1001 REM *****
1002 REM *      BUBBLE-SORT EFFICENT      *
1003 REM *****
1004 REM
1006 S=TI
1010 FOR I=1 TO N-1
1020 : B=X(I):K=I
1030 : FOR J=I+1 TO N
1040 : : IFB<=X(J) THEN 1060
1050 : : B=X(J):K=J

```

```
1060 : NEXT J
1070 : X(K)=X(I)
1080 : X(I)=B
1090 NEXT I
1100 E=TI:T(1)=(E-S)/60
1120 RETURN
2000 REM
2001 REM *****
2002 REM *   BUBBLE-SORT INEFFICIENT   *
2003 REM *****
2004 REM
2020 S=TI
2030 FOR I=1 TO N-1
2040 : FOR J=I+1 TO N
2050 : : IF(X(I)<X(J)) THEN 2090
2060 : : B=X(I)
2070 : : X(I)=X(J)
2080 : : X(J)=B
2090 : NEXT J
2100 NEXT I
2110 E=TI:T(2)=(E-S)/60
2120 RETURN
3000 REM
3001 REM *****
3002 REM *   SHELL-METZNER SORT   *
3003 REM *****
3004 REM
3020 SA=TI
3030 S=N
3040 S=INT(S/2)
3050 IF S=0 THEN 3180
3060 D1=N-S
3070 FOR J=1 TO D1
3080 : I=J
3090 : S1=I+S
3100 : IF X(I) <= X(S1) THEN 3160
3110 : H=X(I)
3120 : X(I)=X(S1)
3130 : X(S1)=H
3140 : I=I-S
3150 : IF I>0 THEN 3090
3160 NEXT J
3170 GOTO 3040
3180 E=TI:T(3)=(E-SA)/60
3190 RETURN
4000 REM
4001 REM *****
4002 REM *   QUICK - SORT   *
```

```
4003 REM *****
4004 REM
4005 S=TI
4010 LI=1
4020 B(LI)=N+1
4030 M=1
4040 J=B(LI)
4050 I=M-1
4060 IFJ-M<3THEN4220
4070 MI=INT((I+J)/2)
4080 I=I+1
4090 IFI=JTHEN4160
4100 IFX(I)<=X(MI)THEN4080
4110 J=J-1
4120 IFI=JTHEN4160
4130 IFX(J)>=X(MI)THEN4110
4140 X=X(I):X(I)=X(J):X(J)=X
4150 GOTO4080
4160 IFI>=MITHENI=I-1
4170 IFJ=MITHEN4190
4180 X=X(I):X(I)=X(MI):X(MI)=X
4190 LI=LI+1
4200 B(LI)=I
4210 GOTO4040
4220 IFJ-M<2THEN4250
4230 IFX(M)<X(M+1)THEN4250
4240 X=X(M):X(M)=X(M+1):X(M+1)=X
4250 M=B(LI)+1
4260 LI=LI-1
4270 IFLI>0THEN4040
4280 E=TI:T(4)=(E-S)/60
4300 RETURN
5000 REM
5001 REM *****
5002 REM *      LINEAR      SORT      *
5003 REM *****
5004 REM
5010 S=TI
5020 FORI=2TON
5030 :X(0)=X(I)
5040 :B=I-1
5050 :IFX(0)>=X(B)THEN5090
5060 :X(B+1)=X(B)
5070 :B=B-1
5080 :IFB<>0THEN5050
5090 :X(B+1)=X(0)
5100 NEXTI
5110 E=TI:T(5)=(E-S)/60
```

```

5120 RETURN
10000 REM
10010 REM *****
10020 REM *   BAR CHART           *
10030 REM *****
10040 REM
10050 A=1948:B=56220:B$="IBBL. BUBL. S-M. QUICK LIN. RES
.(HOME)"
10070 PRINT"{CLR}          S O R T - C O M P A R I S O N"
10080 POKE214,24:POKE211,2:SYS58732
10090 PRINTB$
10100 FORI=1TO6
10110 Y=T(I)/10
10120 IFY=0ANDI=6THEN10210
10130 IFY=0THENA=A+6:B=B+6:NEXT
10140 POKEA,105:POKEB,I
10150 FORJ=1TOY
10160 POKE A-J*40,160:POKE B-J*40,I
10170 NEXT
10180 POKE A-J*40+39,233:POKE B-J*40+39,I
10190 A=A+6:B=B+6
10200 NEXT
10210 WAIT653,4
10220 RETURN
12000 REM
12001 REM *****
12002 REM *   OUTPUT TIMES       *
12003 REM *****
12004 REM
12010 PRINT"{CLR}SIMPLE BUBBLE :      ";T(2);"SEC.{C/DN}{C/
DN}"
12020 PRINT"IMPROVED BUBBLE :      ";T(1);"SEC.{C/DN}{C/DN}"
12030 PRINT"SHELL-METZNER-SORT: ";T(3);"SEC.{C/DN}{C/DN}"
12040 PRINT"QUICK-SORT:          ";T(4);"SEC.{C/DN}{C/DN}"
12050 PRINT"LINEAR-SORT:         ";T(5);"SEC.{C/DN}{C/DN}"
12060 PRINT"RESERVE-STILL FREE  ";T(6);"SEC.{C/DN}{C/DN}"
12065 PRINT"TIME FOR          ";N;"SORTS"
12070 POKE214,22:POKE211,12:SYS58732
12080 PRINT"< CONTROL >":WAIT653,4
12100 RETURN
15000 REM
15001 REM *****
15002 REM *   INPUT               *
15003 REM *****
15004 REM
15005 PRINT"{GRY2}{CLR}SORT - COMPARISON "
15010 PRINT"{WHT}HOW MANY NUMBERS WILL {C/DN}{C/DN}"
15020 PRINT"BE SORTED ?{GRY2}"

```

```
15030 INPUTN:PRINT" {CLR}"
15035 DIMX(N),Y(N)
15040 FOR I= 1 TO N
15050 :Y(I)=RND(1)*20
15060 NEXT I
15070 RETURN
20000 REM
20001 REM *****
20002 REM *      EXCHANGE VARIABLES      *
20003 REM *****
20004 REM
20010 FORI=1TON
20020 :X(I)=Y(I)
20030 NEXTI
20040 RETURN
```

READY.

Chapter 5: Mathematical Programs

5.1 Zero-point calculation

If $y=f(x)$ is defined over an interval $[a,b]$ and is continuous over that interval, and furthermore, if $f(a)$ and $f(b)$ have different signs, then there is at least one zero (point) for $f(x)$ between a and b . This point is also called the root of the function.

We now introduce two procedures for finding this zero point:

A. Approximation procedure (Newton)

If an approximate value of the zero point for the equation $f(x)=0$ is known, we will get a more exact value if we take the tangent on the function graph and determine the point at which it intercepts the x -axis. We place this value in turn into a new zero-point calculation. A precondition for convergence is the existence of the first two derivatives in the given interval. Here the choice of the starting point comes into play. The iteration follows the rule

$$x_{i+1} = x - \frac{f(x_i)}{f'(x_i)} \quad (1)$$

This equation can also be viewed as a Taylor series broken off after the second member.

```
10 REM *****
11 REM *   NEWTON           *
12 REM *   APPROXIMATION   *
13 REM *****
14 REM
50 PRINT CHR$(147):POKE 53280,0:POKE 53281,0
60 PRINT:PRINTTAB(14);"N E W T O N":PRINT
70 GOSUB 15000
90 REM FUNCTION
100 DEF FNF(X)=10*X↑5-EXP(X*X)
110 DEF FND(X)=50*X↑4-EXP(X*X)
200 GOSUB 16000
300 GOSUB 1000
400 GOSUB 10000
500 END
1000 X=D
1200 DK=(-1)*FNF(X)/FND(X)
1300 X=X+DK
1400 IF ABS(DK)<10↑(-S%) THEN RETURN
1500 GOTO 1200
9990 REM
10000 REM OUTPUT
10010 PRINT CHR$(147):PRINT:PRINTTAB(10);"R E S U L T S":P
PRINT:PRINT
10200 PRINT"   ZERO AT           ":"X
10220 PRINT
10250 PRINT"   ORDINATE VALUE: ":"FNF(X)
10300 RETURN
15000 PRINT:PRINT"ENTER THE FUNCTION AND ITS"
15050 PRINT"FIRST DERIVATIVE AND RESTART"
15060 PRINT"THE PROGRAM WITH RUN 100"
15100 LIST100-110
16000 INPUT"INITIAL VALUE";D:PRINT
16100 INPUT"# OF PLACES OF ACCURACY";S%
16150 PRINTCHR$(147):PRINT:PRINT
16200 RETURN
```

READY.

B. Secant procedure (Regula falsi)

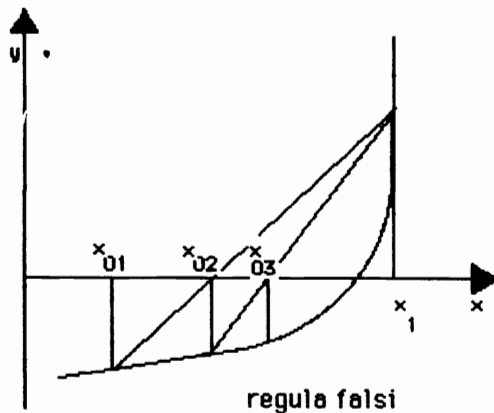
This procedure makes use of the proportionality rule which applies when a ray pair is cut by two parallel lines. The corresponding segments are proportional in length. From the equation for a chord which traverses an arc from the points X_1 to X_2 , we get

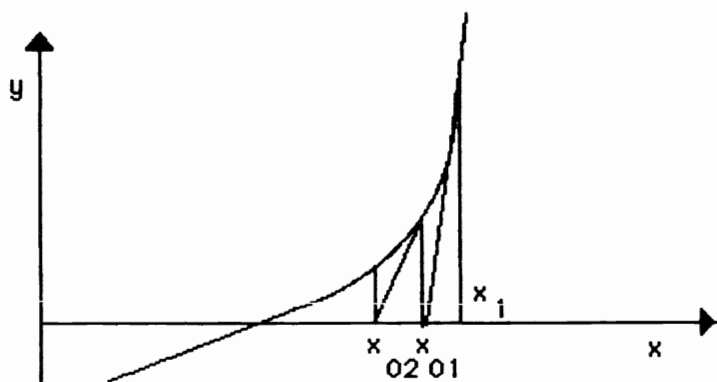
$$x_{11} = x_1 - \frac{(x_2 - x_1) * f(x_1)}{f(x_2) - f(x_1)} \quad (2)$$

or the iteration rule

$$x_{11} = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)} \quad (3)$$

A quick graphics comparison clarifies the two methods.





newton method

The first two derivations of the function need not be known in order to use this method, but this works to its disadvantage since it does not converge as well. Furthermore, two starting values are necessary.

The procedure of linear interpolation is often used for calculating logarithm tables.

```
10 REM *****
11 REM *
12 REM *      REGULA FALSI      *
13 REM *
14 REM *****
50 PRINT CHR$(147):POKE 53280,0:POKE 53281,0:PRINT:PRINT
100 DEF FNF(X)=(X-4)*(X-2.3)*(X+2.1)*(X+3.34)
110 DEF FNR(X)=INT(1E6*X)/1E6
200 PRINT TAB(8);"R E G U L A F A L S I":PRINT:PRINT
210 PRINT"ENTER THE INTERVAL BOUNDARIES"
220 PRINT
230 GOSUB 15000
240 GOSUB 1000:GOSUB 10000
250 END
900 I=0
1000 X1=A:X2=B
1100 Y1=FNF(X1):Y2=FNF(X2)
1200 X=X2-(X2-X1)/(Y2-Y1)*Y2
1250 I=I+1
1300 IF ABS(X2-X1)<1E-5 THEN RETURN
1400 X1=X2:X2=X:GOTO1100
10000 PRINT"*****ZERO POINT:";(X1+X2)/2
10100 PRINT
10200 PRINT:PRINT"****FUNCTION VALUE AT X: ";FNR(FNF((X1+X2)/2))
10250 PRINT:PRINT
10300 PRINT"****ITERATION STEPS: ";I
10400 RETURN
15000 INPUT"A: ";A
15050 PRINT"{C/UP}";TAB(10);:INPUT"B: ";B
15100 PRINT
15200 RETURN
```

READY.

5.2 The Romberg differentiation (Difference quotients)

For the differentiation of a function $F(x)$, one calculates the difference quotients of F for intervals of length $2H^J$.

The lengths are calculated iteratively with:

$$H(J) = H(J-1)/2^J, \quad H(0) = H \quad (1)$$

$$J = 1, 2, \dots$$

The difference quotients now have the form

$$D(0, J) = \frac{F(X_0 + H(J)) - F(X_0 - H(J))}{2H(J)} \quad (2)$$

The derivative of the function F is calculated with the help of these numbers $D(0, J)$ at the point X_0

$$D(K, J) = (2^{2K} * D(K-1, J+1) - D(K-1, J)) / (2^{2K-1}) \quad (3)$$

$$K = 0, 1, \dots$$

This procedure is executed until one has gone under a previously chosen margin of error E :

$$ABS(D(K, 0) - D(K-1, 1)) \leq E \quad (4)$$

```

1 REM
2 REM *****
3 REM *                                     *
4 REM * ROMBERG - DIFFERENTIATION *
5 REM *                                     *
6 REM *****
7 REM
8 PRINT "{CLR}"
10 DIM X(30),Y(30),D(30,30),H1(30),C(30)
100 POKE 53280,0:POKE 53281,0:PRINT "{GRY2}"
110 REM
115 POKE 214,12:POKE 211,6:SYS 58732
120 PRINT "ROMBERG - DIFFERENTIATION"
130 FOR W=1 TO 1000:NEXT
135 REM
140 GOSUB 15000: REM * INPUT      *
145 REM
150 GOSUB 5000: REM * FUNCTION *
155 REM
200 PRINT "{CLR}"X","Y1","Y2"
205 PRINT
210 FOR X=A TO B STEP IN
220 : H1=H
230 : FOR J=0 TO 10 STEP 1
240 : : H1(J)=H1/2↑J
250 : : D(0,J)=(FNF(X+H1(J))-FNF(X-H1(J)))/(2*H1(J))
260 : : H1=H1(J)
270 : NEXT
280 : K=1
285 : J=0
290 : D(K,J)=1/(2↑(2*K)-1)*(2↑(2*K)*D(K-1,J+1)-D(K-1,J))
300 : IFABS(D(K,0)-D(K-1,1))<T THEN GOSUB 10000:GOTO 370
310 : IF J=10 THEN 340
320 : J=J+1
330 : GOTO 290
340 : K=K+1
350 : GOTO 285
360 REM
370 NEXT X
375 POKE 214,12:POKE 211,12:SYS 58732
380 PRINT "{C/DN}{C/DN}{C/DN}MORE CALCULATIONS Y/N"
390 GET A$:IF A$="" THEN 390
400 IF A$="Y" THEN RUN
410 IF A$="N" THEN PRINT "{CLR}":END
420 GOTO 390

```

```

5000 REM
5001 REM *****
5002 REM *   DEF. THE FUNCTION   *
5003 REM *****
5004 REM
5010 DEF FNF(X)=C(0)+C(1)*X+C(2)*X↑2+C(3)*X↑3+C(4)*X↑4+C(
5) *X↑5
5020 RETURN
10000 REM
10001 REM *****
10002 REM *           OUTPUT           *
10003 REM *****
10004 REM
10010 X=INT(X*100+.5)/100
10015 Z=FNF(X)
10020 Z=INT(Z*10000+.5)/10000
10030 D(K,0)=INT(D(K,0)*10000+.5)/10000
10040 PRINT X, Z, D(K,0)
10050 RETURN
15000 REM
15001 REM *****
15002 REM *           INPUT           *
15003 REM *****
15004 REM
15010 PRINTCHR$(147)
15020 INPUT" {WHT}ENTER DEGREE OF POLYNOMIAL: ";N
15040 PRINT" {C/DN} {C/DN}ENTER THE COEFFICIENT"
15050 PRINT" {C/DN} {C/DN} K(I) {C/DN}"
15060 FOR I=0 TO N
15070 PRINT" {C/DN} COEFFICIENT K(";I;") {C/DN}"
15080 INPUT C(I)
15090 NEXT I
15095 PRINTCHR$(147)
15100 INPUT"START OF DIFFERENTIATION: ";A
15110 INPUT" {C/DN} {C/DN}END OF DIFFERENTIATION: ";B
15120 INPUT" {C/DN} {C/DN}STEP SIZE           : ";IN
15130 INPUT" {C/DN} {C/DN}STEP SIZE OF DIFF. COEFFICIENT
: ";H
15140 INPUT" {C/DN} {C/DN}TOLERANCE DEMANDED           : ";T
15160 RETURN

READY.

```

5.3 Numerical integration

We should present two methods for numerical integration at this point so that you are not lost if, when using the '64, you come across an integral in an equation. The methods which will be mentioned are of special significance when the integral cannot be easily solved.

5.3.1 Simpson's rule

Instead of the trapezoidal rule which approximates a definite integral using the areas of trapezoids, we will use Simpson's rule which uses parabolas. A parabola is divided into three equidistant curve pieces. After multiple halving of the interval, the sum of the areas under the parabola segments gives a good approximation for the original curve $y=f(x)$

$$\int_a^b f(x) dx = \frac{h}{3} (y(0) + 4y(1) + 2y(2) + 4y(3) + \dots + 4y(2n-1) + y(2n))$$

The disadvantages of the Simpson rule consist of the facts that an even number of partions are required in the interval and the Simpson's rule is not recommendable for intergrations over large intervals.

```

1 REM *****
2 REM *
3 REM *      INTEGRATION BY      *
4 REM *
5 REM *      SIMPSON'S      RULE      *
6 REM *
7 REM *****
8 REM
9 REM
10 POKE53280,0:POKE53281,0:PRINT"{GRY2}"
20 CLR:PRINTCHR$(147)
30 GOSUB 15000:REM * TITLE      *
35 POKE 198,0:PRINT"{CLR}"
40 GOSUB 10000:REM * INPUT      *
45 PRINT"{CLR}"
900 PRINT"{C/RT}{C/RT}{C/RT}{GRY2} INTERVALS"SPC(10)"SURFAC
E AREA","{WHT}{C/DN}"
910 REM
920 REM *****
1000 REM *      SIMPSON      *
1001 REM *****
1002 IN=(E-A)/N
1003 REM
1005 S2=0:S3=0
1010 FOR X=A+IN TO E-3*IN STEP 2*IN
1020 : S2=S2+FNF(X)
1030 : S3=S3+FNF(X+IN)
1040 NEXT X
1050 S2=S2+FNF(E-IN)
1060 F=(S1+4*S2+2*S3)*IN/3
1100 GOSUB 5000
1150 IF T>ABS((F-F1)/F) THEN 5150
1200 N=2*N
1210 F1=F
1300 GOTO 1000
5000 REM
5001 REM *****
5002 REM *      OUTPUT PORTION      *
5003 REM *****
5005 REM
5030 PRINTTAB(5)N;TAB(20)F
5100 RETURN
5150 PRINTTAB(8)"{GRY2}{C/DN}ACCURACY ACHIEVED !!!"
5160 PRINTTAB(11)"{C/DN}{C/DN}{RVON}PRESS A KEY!!!!"
5170 WAIT 198,1:POKE 198,0

```



```
5200 PRINT"{CLR}{C/DN}IF YOU WANT TO INTERGRATE  "
5210 PRINT"{C/DN}A NEW FUNCTION, YOU MUST  "
5230 PRINT"{C/DN}ENTER IT IN LINE 10010  "
5240 PRINT"{C/DN}AND PRESS          T  "
5245 PRINT"{C/DN}< RETURN >{C/DN}{WHT}"
5250 LIST 10010:END
10000 REM
10001 REM *****
10002 REM *      INPUT PORTION      *
10003 REM *****
10004 REM
10010 DEF FNF(X)=2/π*COS(SIN(X))
10015 PRINT"INTEGRATION BY {C/DN}"
10016 PRINT"SIMPSON'S RULE      "
10020 INPUT"{C/DN}{WHT}START OF INTEGRATION:";A
10030 INPUT"{C/DN}END OF INTEGRATION:";E
10040 INPUT"{C/DN}ACCURACY          :";T
10050 INPUT"{C/DN}START INTERVAL    :";N
10055 IF E=0 OR T=0 OR N=0 THEN RUN
10060 PRINT"{CLR}":F1=0
10070 S1=FNF(A)+FNF(E)
10100 RETURN
15000 REM
15010 REM *****
15020 REM *      TITLE SIMPSON      *
15030 REM *****
15040 REM
15050 POKE 211,8:POKE 214,12:SYS 58732
15060 PRINT"{GRY2}INTEGRATION BY SIMPSON"
15070 FOR I=1 TO 5000:NEXT:RETURN
```

READY.

5.3.2 Romberg Procedure

In general because of the computational accuracy of the Romberg procedure it is of excellent value. This method is a single improvement in trapezoidal rule procedures.

```

10 REM *****
20 REM *
30 REM *   ROMBERG INTEGRATION   *
40 REM *
50 REM *****
60 REM
90 PRINTCHR$(147)
100 DEFFNF(X)=X*X*EXP(-X*X)
200 PRINT:PRINTTAB(13);"R O M B E R G":PRINT:PRINT
300 GOSUB 15000
400 GOSUB 1000
999 END
1000 REM CALCULATION
1010 REM
1200 T(0)=(B-A)*(FNF(A)+FNF(B))/2
1250 FOR J=1 TO N
1260 : S=0
1270 :: FOR I=1 TO 2↑J-1 STEP 2
1280 ::: S=S+FNF(I*B/2↑J)*B/2↑(J-1)
1290 :: NEXT I
1300 : T(J)=(T(J-1)+S)*.5
1310 NEXT J
1350 GOSUB 10000:RETURN
9990 REM
10000 REM OUTPUT RESULTS
10010 REM
10100 PRINT:PRINTCHR$(147);TAB(10);" R E S U L T S":PRINT:
PRINT
10200 PRINT"      INTEGRAL VALUE";T(N)
10250 RETURN
15000 PRINT"  ENTER INTEGRATION BOUNDRIES!"
15100 PRINT:INPUT"  A:";A
15150 PRINT"{C/UP}";TAB(10);:INPUT"  B:";B
15200 PRINT"{C/DN}{C/DN}{C/DN}{C/DN}{C/DN}{C/DN}"      HOW MA
NY APPROXIMATION STEPS: ";:INPUT N
15250 RETURN

```

READY.

5.4 Linear Regression

In the natural sciences we often have things to do with sets of measured data. These can, for example, consist of dependent values y and the corresponding values x . For the sake of simplicity we assume that the x values are error-free. This is by and large true if the x 's represent time values or temperatures which we have given.

One tries to represent a functional connection between x and y in the form

$$y = ax + b$$

This form results in a line and therefore a linear relationship.

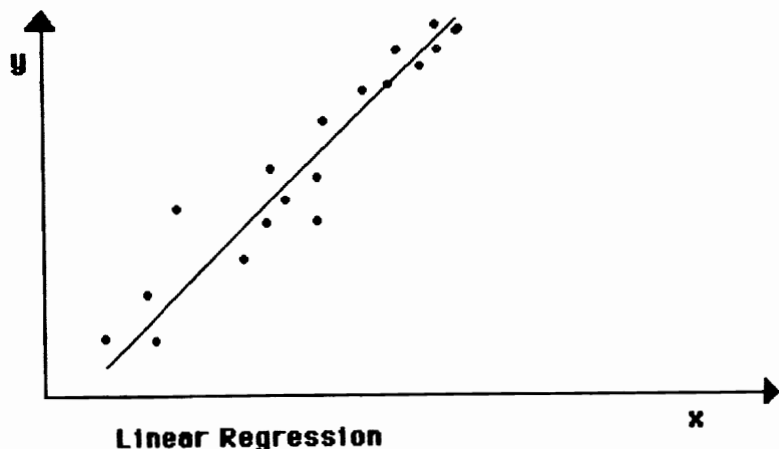
If one replaces x with a variable $x(s)$, one does not receive the expected value of $y = y(s)$. An error of

$$a * x(s) + b - y(s)$$

occurs.

When we now search for a linear relationship respective to a line which best fits our data, we must determine a and b such that the sum of the squares of the errors is as small as possible.

$$\text{sum } (a * x(s) + b - y(s))^2 = \min$$



The correlation coefficient

Since we don't have an exactly functional dependency between X and Y but can only expect a statistic (probability-dependent) dependency, one thinks of this dependency in the form of the correlation coefficients. This is defined as the quotient of the sum of all products xy divided by the product of sums x and y . Its symbol is r .

Our regression line corresponds more exactly to the original values the closer the amount of the coefficient coordinates comes to 1.

The following small program performs a linear regression using the formulas given above. It calculates the coefficients and allows the comparison between the original values and values calculated using the regression line. The program flow is easy to see and need not be explained in greater detail.

It should be noted that many non-linear relationships can be reduced to linear forms. In such cases we are not dealing with pure non-linear regression. When we want to linearize an exponential function, for example, we get a logarithmic representation, a line. By contrast for non-linear regression, it is assumed that a relationship in the form of curve is found between the values. This curve is then equalize at the points of the lines.

```

1000 REM *****
1010 REM * LINEAR REGRESSION *
1020 REM * *
1030 REM * *
1040 REM * *
1050 REM * *
1055 REM *****
1060 POKE 53280,0:POKE 53281,0:PRINT "{GRY2}"
1070 CLR
1080 GOSUB 5000
1100 DIM X(100),Y(100)
1105 PRINT "{CLR}":PRINT "{GRY2}LINEARE REGRESSION{C/DN}"
1110 INPUT "{WHT}NUMBER OF VALUE PAIRS";N
1115 IF N<=0 THEN 1105
1120 FOR I=1 TO N
1130 : PRINT "{C/DN}DATA PAIR NO.";I
1140 : INPUT "{C/UP}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C
/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}"
1150 NEXT I
1400 REM
1401 REM *****
1402 REM * SUMS X,Y,XX,XY *
1403 REM *****
1405 REM
1410 FOR I=1 TO N
1420 : S1=S1+X(I)
1430 : S2=S2+Y(I)
1440 : S3=S3+X(I)*X(I)
1450 : S4=S4+X(I)*Y(I)
1460 NEXT I
1500 REM
1501 REM *****
1502 REM * ERROR QUADRATIC EQUAT. *
1503 REM *****
1505 REM
1520 XM=S1/N:YM=S2/N
1530 FOR I=1 TO N
1540 : S5=S5+(X(I)-XM)*(Y(I)-YM)
1550 : S6=S6+(X(I)-XM)2
1560 : S7=S7+(Y(I)-YM)2
1570 NEXT I
1580 K=S5/SQR(S6*S7)
1590 M=(N*S4-S1*S2)/(N*S3-S1*S1)
1600 B=(S3*S2-S1*S4)/(N*S3-S1*S1)

```

```

1610 REM
1611 REM *****
1612 REM *   OUTPUT   REGRESSION   *
1613 REM *****
1615 REM
1620 PRINT"{CLR}"{C/DN}"
1630 PRINT"{GRY2}CORRELATION COEFFICIENT ={WHT}";K
1650 PRINT"{GRY2}"{C/DN}"{C/DN}"{C/DN}"      REGRESSION LINE   :
"
1680 PRINT"{GRY2}"{C/DN}"{C/DN}"SLOPE              ={WHT}";M
1690 PRINT"{GRY2}"{C/DN}"{C/DN}"ORDINATE SECTION  ={WHT}";B
1700 PRINT"{C/DN}"{C/DN}"{C/DN}"{C/DN}"DO YOU WANT TO COMPARE
EXPECTED      "
1705 PRINT" AND EMPIRICAL VALUES      Y/N":PRINT
1710 GET A$: IF A$="" THEN 1710
1720 IF A$="Y" THEN 1800
1730 IF A$="N" THEN 1890
1740 GOTO 1710
1800 REM
1801 REM *****
1802 REM * COMPARE EMPIRICAL /REG. *
1803 REM *****
1804 REM
1805 PRINT"{CLR}"
1810 FOR I=1 TO N STEP 10
1820 : PRINT"{GRY2} X-VALUES,"Y INPUT  ","Y THEORETICAL":P
RINT"QE"
1830 : FOR J=0 TO 9
1840 : : K=J+I
1850 : : PRINTX(K),Y(K),B+M*X(K)
1855 : : IF K=N THEN 1890
1860 : NEXT J
1865 PRINTTAB(11)"{C/DN}"{GRY2}"{RVON}"PRESS A KEY"
1870 : GET A$: IF A$="" THEN 1870
1875 PRINT"{CLR}"
1880 NEXT I
1890 PRINTTAB(11)"{GRY2}"{C/DN}"NEW VALUE Y/N"
2000 GET A$: IF A$="" THEN 2000
2010 IF A$="Y" THEN RUN
2020 IF A$<> "Y" AND A$<> "N" THEN 2000
2030 END
5000 REM
5010 REM *****
5020 REM * TITLE REGRESSION *
5030 REM *****
5035 PRINT"{CLR}"
5040 REM
5050 POKE 211,10: POKE 214,12:SYS 58732
5060 PRINT"{GRY2}LINEARE REGRESSION"
5070 FOR I=1 TO 5000:NEXT
5080 RETURN

```

5.5 Calculating probability

Events we cannot calculate because of the complexity of their relationships are generally denoted as random events.

We try again and again to describe these events in general. There have been many schemes for winning games of luck such as roulette or the lottery. So aggressive have these efforts been that at the borders of such calculations are bits of knowledge that have significance for the natural scientist and even greater significance for the social scientist.

If one has determined to use deterministic models for describing events in the natural science, one quickly recognizes the advantages of a mathematical judgement of a complex set of conditions which form the basis of a random event. If the event concerned is completely determined by the given conditions, one speaks of it as being a "certain" event and no longer a random event.

If we take as an example the throwing of dice, we can look at the dropping of the dice as a certain event; the resulting numbers on the die is, on the other hand, a random event. A characteristic of a random event is the condition that it is impossible to grasp the given conditions in detail.

The task of a probability calculation is the mathematical description of random events. The classic definition of the probability for an event A, as given by Laplace based on equally probable elementary events, is:

$$P(A) = \frac{\text{Number of valid results of A}}{\text{Number of possible results of A}}$$

If we furthermore assume that A and B are events which do not exclude each other, new criteria for the probabilities of simultaneous events must be formulated. The probability that A takes place under that condition of the previous event B with $p(B) > 0$, finds expression in the "conditional probability" of A under B, $p(A/B)$.

If we assume that $A(1), A(2), \dots, A(n)$ are incompatible results for which each always occurs together with an additional event B, the probability of B can be calculated using the multiplication rule of probability calculation

$$P(B) = \sum_{i=1}^n P(A_i) * P(B|A_i) \quad (1)$$

This is the statement of total probability.

The probability of B can also be calculated under the condition that B has occurred. The Bayes formula serves this purpose:

$$P(A_j|B) = \frac{P(A_j) * P(B|A_j)}{\sum_{i=1}^n P(A_i) P(B|A_i)} \quad (2)$$

This equation is designated as the "formula over the probability of hypotheses."

Example: Inside a software store you guess blindly at a program you are going to buy. In this software store they have both original software and pirated software. You have selected an program from Abacus Software. What is the probability that the program is pirated?

```
10 REM *****
11 REM *
12 REM * BAYES FORMULA *
13 REM *
14 REM *
15 REM *****
40 DIM A(11),B(11)
50 PRINTCHR$(147)
400 PRINTTAB(12);"{RVON}BAYES'S FORMULA{RVOF}";CHR$(17)
470 DEF FND(X)= INT(1E3*X+0.5)/1E3
480 INPUT"NUMBER OF EVENT PAIRS";EP%
500 REM CALL INPUT SUBROUTINE
510 GOSUB 15000:GOSUB 1000:GOSUB 10000:GOSUB 14000
520 CLR: GOTO40
990 :
1000 REM TOTAL PROBABILITY
1010 :
1100 FOR I=1 TO EP%
1200 : TW=TW+A(I)*B(I)
1300 NEXT
1490 :
1500 REM BAYES'S FORMULA
1510 :
1520 REM CALCULATION OF (A(I)ONB)
1530 :
1550 FOR I=1 TO EP%
1600 : PB(I)= A(I)*B(I)/TW
1650 NEXT
1690 :
1700 RETURN
9990 :
10000 REM OUTPUT
10010 :
10100 PRINTCHR$(147);TAB(15);"{RVON}RESULTS{RVOF}"
10150 POKE 211,0:POKE 214,8:SYS 58732
10200 PRINT"*****TOTAL PROBABILITY :";FND(TW)
10230 PRINT
10250 FOR I=1 TO EP%
10300 : PRINT"*****P(A(";I;CHR$(157);"| B) :";FND(PB(I))
10400 NEXT I
10450 POKE 211,15:POKE 214,22:SYS 58732:PRINT"<KEY>":WAIT
198,1
10500 REM OUTPUT TO PRINTER?
10590 :
10600 RETURN
```

```
13990 :
14000 PRINTCHR$(147);"DO YOU WANT TO DO MORE CALCULATIONS?
"
14010 PRINT"(Y/N)"
14020 FOR I = 1 TO 10:GETC$:NEXT
14030 GETC$:IF C$="" THEN 14030
14040 IF C$="Y" THEN RETURN
14090 :
14100 END
15000 REM INPUT
15010 :
15020 PRINTCHR$(147);CHR$(17);"PROBABILITY OF PROJECTED EV
ENTS!"
15030 PRINT
15100 FOR I= 1TO EP%
15120 : PRINT"P(A(";I;CHR$(157);")) ";
15130 : INPUT A(I)
15140 : IF A(I)>=1 OR A(I)<=0 THEN 15130
15150 NEXT I
15190 :
15200 REM INPUT
15220 PRINTCHR$(147);"PROBABILITY OF PROVEN EVENTS!"
15230 PRINT
15300 FOR I= 1TO EP%
15320 : PRINT"P(B|A(";I;CHR$(157);")) ";
15330 : INPUT B(I)
15340 : IF B(I)>=1 OR B(I)<=0 THEN 15330
15350 NEXT I
15400 INPUT "ALL O.K. (Y/N) ";P$
15450 IF P$="N" THEN 15000
15460 :
15480 :
15490 :
15500 RETURN
15510 :
```

READY.

5.5.1 Binomial distribution

Formulation of a problem by means of an example.

Let p represent the proportion of carp to other fish in a lake. P then gives the probability that a fisherman will, when he catches a single fish, will catch a carp (p is number between 0 and 1). Accordingly, the probability of catching another kind of fish = $1-p$.

We examine the following randomness experiment:

N fish are pulled from the lake with a net.

Let $x_i = 1$, if the i th fish is a carp

0, if not

$i=1, \dots, N$

These random variables X_i are Bernoulli-distributed with the parameters p .

The random variable

$$X = \sum_{i=1}^N X_i \quad (1)$$

is binomially distributed with the parameters N and p ($B(N, p)$ distributed).

It applies for probability that one will catch K or fewer carp:

$$P(X \leq K) = \sum_{j=0}^K \binom{N}{j} p^j (1-p)^{N-j} \quad (2)$$

As one can see, calculating this number will take a great deal of time if N is very large. It is for this reason that the sum is approximated by an integral function:

The function

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{t^2}{2}\right) dt \quad (3)$$

is called the Gaussian distribution function or the distribution function of the normal distribution.

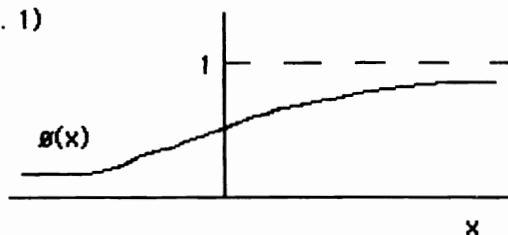
It possess the following characteristics:

$$\Phi(-X) = 1 - \Phi(X) \quad (3.1)$$

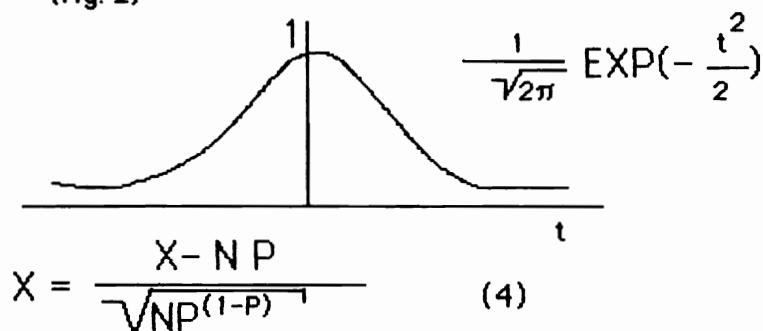
$$\Phi(0) = \frac{1}{2} \quad (3.2)$$

$$\Phi(X) - \Phi(-X) = 2\Phi(X) - 1 \quad (3.3)$$

(fig. 1)



(fig. 2)



X_1 is called the standardized randomness variable belonging to X .

$$X \leq K \iff X \leq \frac{K - NP}{\sqrt{NP(1-P)}} \quad (5)$$

And the following applies for a sufficiently large N :

$$P(K \leq X \leq L) \approx \Phi\left(\frac{L - NP + 0.5}{\sqrt{NP(1-P)}}\right) - \Phi\left(\frac{K - NP - 0.5}{\sqrt{NP(1-P)}}\right) \quad (6)$$

This is a consequence of the boundary-value theorem of de Moire and Laplace.

Rule of thumb: N is considered to be sufficiently large when $N \cdot p \cdot (1-p)$ is greater than 9. This means that the approximation in this case suffices for statistical purposes.

Formula (6) can be rewritten together with (5) and the short form

$$Z_1 = \frac{K - NP - 0.5}{\sqrt{NP(1-P)}} \quad Z_2 = \frac{L - NP - 0.5}{\sqrt{NP(1-P)}} \quad (7)$$

like so

$$P(K \leq X \leq L) = P(Z_1 \leq X_1 \leq Z_2) \approx \Phi(Z_2) - \Phi(Z_1) \quad (8)$$

The following cases are particularly interesting:

$$|X_1| \leq Z, \text{ i.e. } Z_1 = -Z, Z_2 = Z \text{ bzw. } \frac{K-L}{2} = NP$$

$$P(|X_1| \leq Z) = \Phi(Z) - \Phi(-Z) = 2\Phi(Z) - 1 \quad (9)$$

$$|X_1| \leq Z; \quad P(|X_1| \geq Z) = 1 - 2\Phi(Z) \quad (10)$$

$$X_1 \leq Z; \quad P(X_1 \leq Z) = \Phi(Z) \text{ i.e. } K = -\infty \quad (11)$$

$$X_1 \geq Z; \quad P(X_1 \geq Z) = 1 - \Phi(Z); \text{ i.e. } L = +\infty \quad (12)$$

These cases will be handled in the program.

The program

Explanations about the program

The non-elementary integral in (3) is approximated with the following function:

$$F(Z) = 1 - \exp(-Z^2) * (A1*T + A2*T^2 + A3*T^3) \quad (13)$$

where $A1=.4361836$, $A2=-.1201676$, $A3=.937298$

$T=1/(1+C*Z)$, $C=.33267$ (Z positive)

D is a correction factor; it increases the integration interval.

The cases for one-sided boundary in (9) and (10) are handled with an input of 1, while the cases with a two-sided boundary in (11) and (12) are handled with an input of 2.

Examples:

(i) Back to our lake:

The proportion of carp is 0.6. We catch 100 fish in our net.

(a) What is the probability that at least 50 carp will be in the net?

Solution: The boundary is one-sided (1).

sample size: $N=100$

probability: $p=0.6$

boundary: $k=50$ (i.e. $Z=-1.9392$)

The program calculates:

$P(X \geq 50) = .97375$ (at least 50 carp)

$P(X \leq 50) = .02625$ (at most 50 carp)

(b) What is the probability that at least 55 but at most 65 carp are in the net?

Solution: Here the boundary is two-sided, but symmetric since $l=2*N*p-k$

boundary: $k=55$ (i.e. $Z=-.91856$)

The program calculates:

$P(55 \leq X \leq 65) = .64168$

$P(X \leq 55, X \geq 65) = .35832$

(ii) A test:

We want to test to see if a roulette wheel favors the house (or if the ball favors the number 0).

The hypothesis H_0 :

"0 occurs as often or less often than the other digits."

Will be tested against the alternative H_1 : "0 occurs more often than the other digits."

H0 is synonymous with $p(0) \leq 1/37 = .027027\dots$

H1 is synonymous with $p(0) > 1/37$

With a sample of size $N = 3700$ (turning the wheel 3700 times), H1 can be rejected if the digit 0 occurs more than 120 times.

The program calculates:

$$P(X \leq 120) = .97869$$

$$P(X > 120) = .02131$$

If, after executing this randomness experiment, 0 occurs more than 121 times, one assumes hypothesis H0. The probability of error, that is, H0 is accepted although it is actually false, is .02131.

```

10 REM *****
11 REM *
12 REM *      BINOMIAL
13 REM *      DISTRIBUTION
14 REM *
15 REM *****
50 PRINT CHR$(147):POKE 53280,0:POKE 53281,0
100 PRINT
110 PRINT"B I N O M I A L D I S T R I B U T I O N"
120 PRINT:PRINT:PRINT
125 PRINT"ONE- OR TWO-SIDED BOUNDRY"
129 PRINT
130 INPUT" (1/2)";M%:PRINT
135 INPUT" SAMPLE SIZE";N:PRINT
140 INPUT" PROBABILITY";Q:PRINT
145 IF M%<1 OR M%>2 OR Q>1 THEN 50
150 A1=.4361836
151 A2=-.1201676
152 A3=.937298
153 C=.33267
160 PRINT
170 PRINT"      BOUNDARY K:";
171 INPUT K
174 REM
175 REM CALCULATION
176 REM
180 D=((N/2>K)+.5)*(N*Q*(1-Q)<20)
190 Z=(K-N*Q-D)/SQR(N*Q*(1-Q))
200 T=1/(1+C*ABS(Z))
300 F=1-EXP(-(Z↑2)/2)*(A1*T+A2*(T↑2)+A3*(T↑3))/SQR(2*π)
320 P=F
330 IF Z<0 THEN P=1-F
340 PRINT
350 PRINTCHR$(147):PRINT:PRINTTAB(10);"REDULTS":PRINT:PRIN
T
400 ON M% GOSUB15500,15100
410 GOTO 20000
14990 REM
15000 REM OUTPUT
15010 REM
15100 PRINT"****P( |X1 |<Z)=";2*F-1
15200 PRINT"****P( |X1 |>Z)=";2-2*F
15250 GOSUB 16000:RETURN
15500 PRINT"****P(X<=K)=P(X1<=Z)=";P
15600 PRINT"****P(X=K)=P(X1=Z)=";1-P
16000 PRINT:PRINT:PRINT"STANDARDIZATION"
16020 PRINT"      RANDOMITY VARIABLE: ";Z
16100 RETURN
20000 END

```

5.5.2 Chi-squared distribution and adaptability test

Considerations

In our lake from section 5.5.1 there are k kinds of fish of types S_1, S_2, \dots, S_k which have proportions to all of the fish of p_1, p_2, \dots, p_k . $p_1 + p_2 + \dots + p_k = 1$.

A sample of size N is taken (caught). We have N_1 fish of type S_1 , N_2 fish of type S_2, \dots, N_k fish of sort S_k .
 $N_1 + N_2 + \dots + N_k = N$

We shall now test to see if the probabilities p_1, p_2, \dots, p_k correspond to the results of the random sample:

The hypothesis $H_0: (N_1, N_2, \dots, N_k) = (N \cdot p_1, N \cdot p_2, \dots, N \cdot p_k)$ will be tested against the

alternative $H_1: (N_1, N_2, \dots, N_k) \neq (N \cdot p_1, N \cdot p_2, \dots, N \cdot p_k)$

H_0 will be rejected if the "distance" between these two vectors is too great.

A usable notion of "distance" for the statistician proves to be the randomness variable χ^2

$$\chi^2 = \sum_{i=1}^K \frac{(N_i - N \cdot p_i)^2}{N \cdot p_i} \quad (1)$$

But how is χ^2 distributed?

Letting $n_1=N*p_1$, $n_2=N*p_2, \dots, n_k=N*p_k$, the following holds:

$$\begin{aligned}
 P(\chi^2 = 0) &= P(N_1=n_1, N_2=n_2, \dots, N_k=n_k) \\
 (2) \quad &= \frac{N!}{n_1! * n_2! * \dots * n_k!} * p_1^{n_1} * p_2^{n_2} * \dots * p_k^{n_k} \quad (2)
 \end{aligned}$$

This distribution is called multinomial distribution and is a generalization of the binomial distribution.

$$\begin{aligned}
 (n_1, \dots, n_k) &:= \sum_{i=1}^K n_i = N \quad (f_{n_1, \dots, n_k}) = \sum_{i=1}^K \frac{(N_i - n_i)^2}{n_i} \\
 P(\chi^2 \leq t) &= \sum_{\substack{(n_1, n_2, \dots, n_k) \\ f(n_1, \dots, n_k) \leq t}} \frac{N!}{n_1! \dots n_k!} p_1^{n_1} \dots p_k^{n_k} \\
 (3)
 \end{aligned}$$

We now define the distribution function χ^2 :

$$G_{K-1}(x) = \frac{1}{\sqrt{2^K} \Gamma(\frac{K}{2})} * \int_0^x S^{K/2-1} * \exp(-S/2) dS, \quad (x > 0)$$

(4)

The following holds for a "sufficient large" N:

$$P(\text{Chi}^2 < x) \approx G_{k-1}(x) \quad (5)$$

We can compute much more easily with the function (4) than with (3).

$k-1$ is the number of degrees of freedom:

Because $p_1 + p_2 + \dots + p_k = 1$, one can choose only $k-1$ probabilities freely; the last is dictated by this equation.

Rule of thumb: N is considered to sufficiently large if for every i ($i=1, \dots, k$) $N \cdot p_i > 5$.

Practical procedure:

- (a) Choose SN as the standard of the test (probability error).
- (b) Get the value x with the help of (4) so that:

$$G_{k-1}(x) = 1 - \text{SN} \quad (6)$$

- (c) The hypothesis $H_0: (N_1, N_2, \dots, N_k) = (N \cdot p_1, N \cdot p_2, \dots, N \cdot p_k)$ will be rejected if the Chi^2 value from the random sample exceeds the value of x .

This is called the Chi^2 adaptability test.

The program

Explanations about the program and examples

The program first calculates the Chi^2 value according to (1). This number is calculated based on the conjectured p_1, p_2, \dots, p_k and the numbers N_1, N_2, \dots, N_k from the realization of the test.

The computer then finds the number x with:

$$G_{k-1}(x) = P(\text{Chi}^2 \leq x) = 1 - SN \quad (7)$$

The non-elementary integral occurring in (3) is an approximation through the first summands of the endless sum:

$$G_{k-1}(x) = \left(\frac{1}{2}\right)^x \frac{K-1}{2} * \frac{\exp(-x/2)}{\Gamma\left(\frac{K+1}{2}\right)} * \left(1 + \sum_{i=1}^{\infty} \frac{x^i}{(K+1)*(K+3)*\dots*(K+2i-1)}\right) \quad (8)$$

Finally, the values Chi^2 and x are compared to each other and determine if hypothesis H_0 can be accepted or not.

Note: The values x is calculated with an accuracy of only ± 0.1 in this program. In order to obtain more accurate values, one must reduce the value of V in the program by an appropriate number of places. This will cause the computation time to increase significantly. The approximation through (8) is very accurate.

Examples:

(i) We want (for standard level $SN=.01$) to test if a die is true or loaded.

$H_0: p_1=p_2=\dots=p_6=0.16666667$

The die is thrown 180 times. We get:

number	i	1	2	3	4	5	6
observed frequency	N_i	25	35	33	32	27	28
expected frequency	$N \cdot p_i$	30	30	30	30	30	30

First the number .01 is entered and then the degrees of freedom $5=6-1$ and then the values p_i and N_i alternately, $i=1,\dots,6$.

Output:

χ^2 value = 2.53334

χ^2 boundary for .01% probability error = 15.1

H_0 is accepted.

(ii) The proportion of carp in our lake is estimated to be 0.2, in addition: trout: 0.4, eels: 0.1, flounder: 0.1, perch: 0.2.

SN will be 0.05

We catch 100 with a net and count:

26 carp, 43 trout, 6 eels, 7 flounder, and 18 perch.

The number of degrees of freedom is 4.

Output:

Chi² value = 4.725

Chi² boundary for .05% probability error = 9.5

H0 is accepted.

```

1 REM *****
2 REM *                                     *
3 REM *      CHI2-DISTRIBUTION             *
4 REM *                                     *
5 REM *      ADAPTABILITY TEST             *
6 REM *                                     *
7 REM *                                     *
8 REM *****
20 PRINT CHR$(147)
30 PRINT
40 PRINT"  CHI-SQUARED DISTRIBUTION
41 PRINT"  AND ADAPTABILITY TEST
42 PRINT
50 POKE53280,0:POKE53281,0
55 PRINT:PRINT:PRINT:PRINTCHR$(5)
60 INPUT"SIGNIFICANCE LEVEL:";SN
70 PRINT:PRINTTAB(3);:INPUT" DEGREES OF FREEDOM:";K
80 K=K+1:DIM NE(K),NB(K),P(K)
90 :PRINT:PRINT
110 FOR I=1 TO K
120 PRINTCHR$(147);" "; I;" EXPECTED PROB.:";
121 INPUT P(I):PRINT
130 PRINT" ";I;" THE OBSERVED QUANTITY:";
131 INPUT NB(I)
135 IF P(I)<=0 OR NB(I)<=0 THEN 120
140 N=N+NB(I)
150 NEXT I
160 FOR I=1 TO K
170 NE(I)=P(I)*N
190 NEXT I
330 X=X+V
1450 FOR I=1 TO K
1460 Z=Z+(NB(I)-NE(I))2/NE(I)
1480 NEXT I
1490 GOSUB 3350
2000 REM CALCULATION OF THE CHI BORDERS USING INFINITE SER
IES"
2010 K=K-1
2100 G=1
2120 FOR F= K/2 TO .5 STEP-1
2130 G=G*F
2140 NEXT F
2150 IF INT(K/2)<K/2 THEN G=G*SQR(PI)
2170 IF G=0 THEN G=1
3120 V=2

```

```
3130 XK=(X/2)↑(K/2)*EXP(-X/2)/G
3140 S=1: A=1: B=K
3150 B=B+2
3160 A=A*X/B
3170 IF A<1E-9 THEN 3120
3190 GOTO 3150
3210 P=XK*S
3320 IF P<=1-SN THEN 3130
3330 IF P>=1-SN AND V=.05 THEN 3420
3340 IF P>=1-SN THEN V=.05 : X=X-3:GOTO 3130
3350 REM OUTPUT
3360 PRINTCHR$(147):PRINTTAB(10); "RESULTS":PRINT
3400 PRINT:PRINT"***OBSERVED"
3410 PRINT"***CHI↑2-VALUE";Z:PRINT
3415 RETURN
3420 PRINT"***P (CHI↑2≤X) =";P
3430 PRINT
3450 PRINT"***CHI↑2-BOUNDRY AT";SN;"Z"
3460 PRINT"***PROBABILITY ERROR:";X-V
3470 PRINT
3480 ON (Z(X)+2) GOTO 3490,3500
3490 PRINT"***HYPOTHESIS H0 IS ACCEPTED":GOTO 4000
3500 PRINT"***HYPOTHESIS H0 IS REJECTED":
4000 END
```

READY.

5.6 Fourier analysis and synthesis

One is confronted with periodic events in many areas. The periodicity may be a function of the time or spatial dependency. Everyone is familiar with periodic events; the rythmn of the heart belong to this category just as do the current from an electrial outlet or a square wave in electronics.

Graphs can be made of all of these examples, graphs which can then be used for further scrutiny and investigation. The natural scientist usually attempts to derive a rule-bound description from the measured values in the graph. Fourier analysis and synthesis are very useful for doing this.

The basis of the whole thing is the fact that every continuous, periodic function can be represented as a pure sine/cosine series. Of these series representations is designated as the Fourier series.

In the purely analytical analysis, the function to be represented must be dealt with in general since the corresponding Fourier series gives an exact reproduction of this function. We cannot proceed in this manner in practice because we usually only deal with a graph or a list of measurements. In this case we must use an approximation procedure. The result of this approximation should come as close as possible to our initial function.

The process of obtaining the Fourier series of a function is called Fourier ANALYSIS, while the calculation of the function values of the original function through the Fourier series is called Fourier SYNTHESIS.

What does such a Fourier series look like? In order to answer this question, we will take a look at a time-dependent, periodic function $f(t)$:

$$f(t) = f(t + T) \text{ or } f(t) = f(t + kT) \quad (1)$$

$$t = \text{time, } T = \text{period, } k = \text{integer}$$

The beginning of the period can be selected as desired. Now we represent functions of the type discussed through a linear combination of trigonometric functions, because these are linearly independent and periodic as well. In our case these are the following expressions:

$$\sin 2\pi \nu \frac{t}{T} = \sin \nu \omega t$$

$$\cos 2\pi \nu \frac{t}{T} = \cos \nu \omega t, \text{ with } \nu = 1, 2, \dots (2)$$

For the Fourier analysis it then follows:

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(k\omega t) + b_k \sin(k\omega t)) \quad (3)$$

The coefficients a_0 through a_k and b_0 through b_k are called Fourier coefficients. These are represented through the following integrals:

$$a_0 = \frac{2}{T} \int_{t_0}^{t_0+T} f(t) dt, \quad (4) \quad a_k = \frac{2}{T} \int_{t_0}^{t_0+T} f(t) \cos(k\omega t) dt, \quad (5)$$

$$b_k = \frac{2}{T} \int_{t_0}^{t_0+T} f(t) \sin(k\omega t) dt. \quad (6)$$

We can select t arbitrarily in order to make the integration as simple as possible. But with the help of our computer we want to get around the integration entirely and obtain the coefficients numerically. From the use of linear equalization methods we know the method of least squares. With this knowledge we assume:

$$\sigma = \sum_{i=1}^n (f(t_i) - t(t_i))^2 \stackrel{!}{=} \text{Minimum}. \quad (7)$$

Furthermore, the period length T is now set to the range $(0, 2\pi)$ and we can then write:

$$X = \frac{2\pi}{T} (t - t_0) = \omega(t - t_0) \quad (8)$$

Let us assume that we are dealing with a graph which resulted from a set of measurements. This curve is divided into n ordinate values, whereby the distances between the values must be the same (equidistant). The number n must be greater than the number of the coefficients to be found. The ordinate values must also be evenly distributed over the

period 2π . Then the following conditions for the interval width, abscissa (x value), and ordinate (y value) result:

$$\Delta x = \frac{2\pi}{n}, (9) \quad x_i = i \Delta x, (9a)$$

$$i=0,1,2,\dots,n-1 \quad y_0 = y_n$$

It is known that the functions $\sin kx$ and $\cos kx$ are orthogonal. The term orthogonal is borrowed from vector calculations and has the same meaning here as it does there. We will not concern ourselves anymore with this here, but rather concentrate on the actual implementation of these facts. We express the coefficients a and b as sums and calculate them through numerical methods:

$$a_0 = \frac{2}{n} \sum_{i=0}^{n-1} y_i, (10) \quad a_k = \frac{2}{n} \sum_{i=0}^{n-1} y_i \cos(kx_i), (10a)$$

$$b_k = \frac{2}{n} \sum_{i=0}^{n-1} y_i \sin(kx_i), (10b) \quad \text{with } k=1,2,3,\dots,n/2-1$$

The best-known algorithm for calculating the Fourier coefficients is the Runge method. We capitalize on the fact that the angular functions have the same values multiple times in four quadrants, even if they have different signs. If the number n of the selected ordinate values can be divided by 4, the calculations become significantly simpler.

We combine the ordinates with the same values. This is known as "folding" the ordinates. This is done twice and yields the following expressions:

First fold:

$$s_k = y_k + y_{n-k}, \quad d_k = y_k - y_{n-k}, \quad k = 1, 2, 3 \dots n/2.$$

Second fold:

$$s_k^1 = s_k + s_{n/2-k}, \quad d_k^1 = s_k - s_{n/2-k}, \quad k = 0, 1, 2, 3 \dots, n/2.$$

This gives us four different sums which replace the sum formulas from (10). We now take the curve and divide the x axis into equal parts (n=12 has proven to be a good choice) and read off the abscissas (0-12) of the corresponding ordinate value.

When the program will ask you for the number of divisions of the abscissa, enter exactly half of the value you have chosen (6 in this case), because the Fourier series has the following form for our special case n=12:

$$y(x) = a_0 + \sum_{k=1}^{\frac{n}{2}} a_k \cos(kx) + \sum_{k=1}^{\frac{n}{2} - 1} b_k \sin(kx), \quad (12)$$

It should be mentioned that the Runge method is carried out with special formulas for the divisions of multiples of 12, whereby no special mathematical knowledge is required in order to find the Fourier coefficients and thereby the series. There are even devices with which can obtain the coefficients simply by tracing the curve!

But back to our program. It calculates the Fourier coefficients of a periodic function when you enter the corresponding ordinate values from the graph. You can get the function values of the graph back with fair accuracy by substituting in the above form of the Fourier series.

If you draw a curve from the functions values calculated from Fourier series, the corresponding graph should come quite close to your initial curve. The greater the number of divisions (n), the more accurate the second curve will be. You can see this quite clearly in the figures on the next page. Figure 5.6a shows the initial function, while figure 5.6b shows the curve resulting from only a few values calculated through use of the Fourier series (synthesis). The correspondence is easy to see.

Naturally, the function shown is an ideal example for explaining the principle. In "real life" you will not get such clear function graphs. The function shown is a square wave function such as you might come across in electronics.

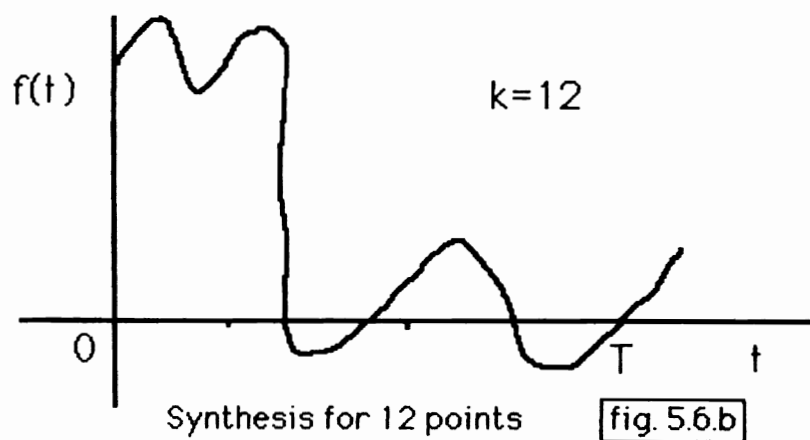
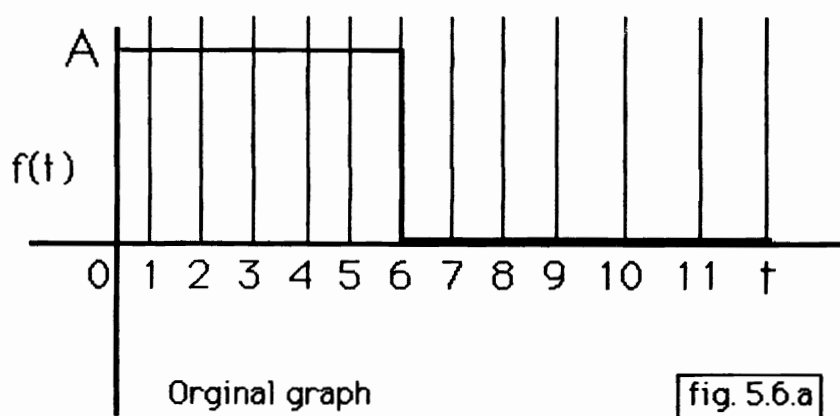
It should be mentioned that, as a rule, the Fourier analysis is limited to specific ranges (such as $0, 2\pi$). A complete analysis would pose difficulties, to say nothing of the fact that a periodic function is not necessarily continuous over its entire range.

About the program:

You will first be required to enter the degree of the function. Here we want the number of curve points to be taken. If we choose 12, for example, (on account of the symmetry of the periodic function we always enter $n/2=6$) the program asks for the individual values as read from the graph. You can gather from figure 5.6a how you must take the values. A period is represented through the interval (0,T).

Please note that no more than 12 points should be entered if you want a graphical representation. Otherwise the screen will not suffice to display the entire oscillation. If you do not want the screen display, the number of values entered can be practically as many as desired. A further restriction if screen display is desired is that the values must be restricted to the range +70 to -20 so that you do not exceed the vertical limits of the screen.

If you should want to enter a vary large number of values, it would be a good idea to save the entered and/or output coefficients on disk. Chapter 3 gives information on how to do this.



```

10 REM *****
20 REM *
30 REM *      FOURIER-ANALYSIS      *
40 REM *
50 REM *    OF A PERIODIC EVENT      *
60 REM *
70 REM *****
80 REM
90 REM
100 CLR
150 REM
200 POKE 53280,0:POKE 53281,0:PRINT"{GRY2}"
250 REM
300 DIM Y(200),S1(200),S2(100),S3(50)
310 DIM D1(100),D2(50),S4(50),D3(50)
320 DIM AK(50),BK(50),F(200),S(100)
330 REM
350 GOSUB 51000:REM * EXPLANATION*
400 REM
500 GOSUB 10000:REM * INPUT      *
550 REM
600 GOSUB 5000:REM * CALCULATION*
650 REM
700 GOSUB 50000:REM * OUTPUT     *
750 REM
800 GOSUB 15000:REM * SIMONS BASIC GRAPHICS*
850 REM
900 PRINT"{CLR}":RUN
1000 REM
2000 REM *****
2010 REM * BEGIN RUNGE-PROCEDURE *
2020 REM *****
2030 REM
2100 A1=0:A2=0
2200 B1=0:B2=0
2300 L=0:K=P
2400 RETURN
2999 REM
3000 REM *****
3010 REM *      -R - L,K      *
3020 REM *****
3030 REM
3100 L=L+2*N1
3200 IF L>=M THEN L=L-M
3300 K=K+2*N1

```

```

3400 IF K>=M THEN K=K-M
3500 RETURN
4000 REM
4010 REM *****
4020 REM * ACTUAL COEFFICIENTS *
4030 REM *****
4040 REM
4100 AK(N1)=(A1+A2)/N
4200 AK(N-N1)=(A1-A2)/N
4300 BK(N1)=(B1+B2)/N
4400 BK(N-N1)=(B1-B2)/N
4500 RETURN
5000 REM
5010 REM *****
5020 REM * START APPROXIMATION *
5030 REM *****
5040 REM
5100 DX=PI/N
5110 FOR I=0 TO P
5120 : S1(I)=SIN(I*DX)
5130 : S1(N-I)=S1(I)
5140 : S1(N+I)=-S1(I)
5150 : S1(M-I)=S1(N+I)
5200 NEXT I
5250 REM
5300 Y(0)=Y(0)/2
5310 Y(M)=Y(0)
5320 Y(N)=Y(N)/2
5330 REM
5340 REM *****
5350 REM * 1. FOLD *
5360 REM *****
5370 REM
5400 FOR I=0 TO N
5410 : S2(I)=Y(I)+Y(M-I)
5420 : D1(I)=Y(I)-Y(M-I)
5430 NEXT I
5500 S2(P)=S2(P)/2
5510 D1(P)=D1(P)/2
5520 REM
5530 REM *****
5540 REM * 2. FOLD *
5550 REM *****
5560 REM
5600 FOR I=0 TO P
5610 : S3(I)=S2(I)+S2(N-I)
5620 : D2(I)=S2(I)-S2(N-I)
5630 : S4(I)=D1(I)+D1(N-I)

```

```

5640 : D3(I)=D1(I)-D1(N-I)
5650 REM
5700 GOSUB 2000
5800 FOR I= 0 TO P STEP 2
5810 : A1=A1+S3(I)
5820 NEXT I
5900 FOR I= 0 TO P STEP 2
5910 : A2=A2+S3(I)
5920 NEXT I
6000 AK(0)=(A1+A2)/M
6010 AK(N)=(A1-A2)/M
6100 FOR N1= 1 TO P STEP 2
6110 : GOSUB 2000
6120 : FOR I=0 TO P STEP 2
6130 :: A1=A1+D2(I)*S1(K)
6140 :: B2=B2+S4(I)*S1(L)
6150 :: GOSUB 3000
6160 : NEXT I
6170 : L=N1
6180 : K=P+N1
6200 : FOR I=1 TO P STEP 2
6210 :: A2=A2+D2(I)*S1(K)
6220 :: B2=B2+S4(I)*S1(L)
6230 : GOSUB 2000
6240 : NEXT I
6250 : GOSUB 4000
6300 NEXT N1
6500 RETURN
10000 REM
10100 REM *****
10110 REM * INPUT ABCISSAS AND N *
10120 REM *****
10200 REM
10300 PRINTCHR$(147)
10400 PRINTTAB(11)" {GRY2}FOURIER-ANALYSIS{C/DN}{C/DN}"
10500 INPUT"{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/R
T}{C/RT}{C/RT}{C/RT}DEGREES OF FUNCTION";N
10600 M=2*N:P=N/2:PRINT"{CLR}"
10800 PRINTTAB(6)"ENTER FUNCTION VALUES"
10900 PRINTTAB(6)"FOR Y(0) TO Y(K) !"
11000 FOR I= 0 TO M-1
11100 PRINT"{WHT}{C/DN}Y(";I;")="
11200 INPUT"{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/U
P}";Y(I)
11400 NEXT I
11500 RETURN
15000 REM
15010 REM *****

```

```

15020 REM * FOURIER SYNTHESIS (SIMON) *
15030 REM *****
15040 REM
15050 HIRES 1,0
15055 TEXT 10,10,"FOURIER SYNTHESIS",1,1,17
15060 LINE0,150,320,150,1
15070 FOR I =0 TO 72
15080 :: FOR J = 1 TO 6
15090 :: F(I)=AK(J)*COS(J*I*PI/36)+BK(J)*SIN(J*I*PI/36)
15092 :: S(I)=S(I)+F(I)
15110 : NEXT J
15115 PLOT 4*I+6,150-S(I)-AK(0),1
15120 NEXT I
15130 GOTO 15130
50000 REM
50100 REM *****
50110 REM * OUTPUT FOURIER ANALYSIS *
50120 REM *****
50200 REM
50210 FOR I= 1 TO N STEP 4
50220 : PRINT"{CLR}{GRY2}FOURIER COEFFICIENTS A(K),B(K) : {C
/DN}{C/DN}{WHT}"
50230 : FOR J = 0 TO 3
50240 :: K=J+1
50250 :: PRINT"A(";K;")=";AK(K),"B(K";K;")=";BK(K),"{C/DN}
"
50260 : NEXT J
50270 : PRINTTAB(11)"{GRY2}{C/DN}PRESS A KEY"
50280 : GET A$: IF A$= "" THEN 50280
50290 : PRINT"{CLR}"
50300 NEXT I
50310 PRINTTAB(11)"{C/DN}FOURIER SYNTHESIS"
50315 PRINTTAB(11)"{C/DN}OR NEW VALUES {WHT}(Y/N)"
50320 : GET A$: IF A$= "" THEN 50320
50330 IF A$="N" THEN RUN
50340 RETURN
51000 REM
51010 REM *****
51020 REM * EXPLANATION AND TITLE *
51030 REM *****
51040 REM
51050 PRINT"{CLR}"
51100 R=PEEK(214):C=PEEK(211)
51110 POKE214,R:POKE211,C
51115 SYS58640
51120 PRINT"FOURIER ANALYSIS AND SYNTHESIS"
51130 FOR I = 0 TO 2000 : NEXT
51140 POKE214,R:POKE211,C:PRINT"{CLR}"

```



```

51150 POKE198,0
52000 PRINTTAB(5)"IN THIS PROGRAM YOU RECIEVE,"
52010 PRINTTAB(5)"BASED ON THE VALUES AT A"
52030 PRINTTAB(5)"PERIODIC EVENT, THE COEFFICIENTS"
52040 PRINTTAB(5)"OF THE CORRESPONDING FOURIER SERIES"
52050 PRINTTAB(5)"IN THE FORM"
52100 PRINT"{WHT}{C/DN}          N-1"
52110 PRINT"
52120 PRINT"      \  "
52130 PRINT"  Y(X)=A +  \  [ A COS(KX) + B SIN (KX) ] "
52140 PRINT"          0  ---  K          K"
52150 PRINT"          K=1{C/DN}"
52160 PRINT"{GRY2}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/D
N}{C/DN}PRESS A KEY"
52170 GETA$: IF A$="" THEN 52170
53000 PRINT"{CLR}":RETURN

```

READY.

5.7 Numerical solution of initial value problems

5.7.1 Comparing differential equation systems

Differential equations must always be solved in natural science and engineering. We find them in mathematical models, for describing natural laws, or simply for solving a technical problem. Many of these differential equations can only be solved numerically because they have no numerical solution. But even differential equations with analytic solutions can be handled more comfortably numerically. We must never forget, however, that the end result will always be just an approximation. This has to do with the nature of the numerical methods. An example of the application of a differential equation system and its solution can be found in section 8.1.

You will certainly ask yourself why we are talking about whole systems here. The solution methods for simple differential equations are only slightly different from the procedures given here. The program contains three different methods of solution. The first method uses the Euler method, then follow the half-step method as well as the Runge-Kutta algorithm.

If you execute the various methods sequentially on the same differential equation system, it will occur that the calculation from Euler to Runge-Kutta ends earlier and earlier. The Runge-Kutta is very efficient. A disadvantage, however, is that each integration step requires the calculation of 4 function values. "Automatic step width control" is also possible, which increases the accuracy.

More exact information about the solution methods mentioned here can be found in any text book on numerical mathematics.

The differential equation system contained in the program is the van der Polsche differential equation. This has a special significance in natural science, but we will not say more about here. It is interesting, however, that this equation is actually a second-order differential equation. It can be reduced to a system of two first-order differential equations. The resulting system consists of two mutually independent equations. This makes the solution easier to realize. We need only agree on the solution of the system.

There is not much to say about the structure of the program; it is fairly self-explanatory. You should refer to the appropriate literature for explanations of the mathematical routines. A precise understanding of the fundamentals is not necessary in order to use the program.

```

10 REM *****
20 REM *
30 REM * NUMERICAL INTEGRATION OF *
40 REM *
50 REM * DIFFERENTIAL EQUATION *
60 REM *
70 REM * SYSTEMS *
80 REM *
90 REM *****
100 CLR
110 DIM Y(500),Z(500),K1(50),K2(50),K3(50),K4(50),RZ(50)
120 POKE 53280,0:POKE 53281,0
130 PRINT "{CLR}"
150 REM
160 GOSUB 50000: REM * TITLE *
200 REM
201 REM *****
202 REM * SELECT A PROCEDURE *
203 REM *****
204 REM
205 REM
210 PRINTTAB(9)"{GRY2}DIFFERENTIAL EQUATIONS"
215 PRINTTAB(9)"SYSTEMS OF N-TH ORDER "
220 PRINTTAB(9)"SELECT THE PROCEDURE "
230 PRINT
240 PRINTTAB(9)"EULER - {WHT}F1"
250 PRINT
260 PRINTTAB(9)"{GRY2}HALF-STOP - {WHT}F3"
270 PRINT
280 PRINTTAB(9)"{GRY2}RUNGE-KUTTA - {WHT}F5"
290 PRINT
295 PRINTTAB(9)"{GRY2}END - {WHT}F7"
300 PRINT
310 GET W$: IF W$="" THEN 310
320 IF W$=CHR$(133) THEN GOSUB 1000:REM * EULER *
330 IF W$=CHR$(134) THEN GOSUB 2000:REM * HALF-STOP *
340 IF W$=CHR$(135) THEN GOSUB 3000:REM * RUNGE-KUTTA *
350 IF W$=CHR$(136) THEN PRINT "{CLR}":END
360 RUN
1000 REM
1001 REM *****
1002 REM * EULER PROCEDURE *
1003 REM *****
1004 REM
1005 N$="EULER PROCEDURE" :GOSUB 15000

```

```

1010 X=XA
1020 FOR J=1 TO N
1030 : Y(J)=Z(J)
1040 NEXT J
1050 GOSUB 10000: REM * OUTPUT *
1060 FOR I=1 TO NR-1
1070 GOSUB 5000: REM * SYSTEM *
1080 FOR J=1 TO N
1090 :: Z(J)=Z(J) *IN*ZP(J)
1100 : NEXT J
1110 : Y(I+1)=Z(1)
1120 : X=X+IN
1130 :GOSUB 10000
1140 NEXT I
1150 RETURN
2000 REM
2001 REM *****
2002 REM * HALF-STEP PROCEDURE *
2003 REM *****
2004 REM
2020 N$="HALF-STEP PROCEDURE" :GOSUB 15000
2030 X=XA
2040 FOR J=1 TO N
2050 : Y(J)=Z(J)
2060 NEXT J
2070 GOSUB 10000: REM * OUTPUT *
2080 FOR I= 1 TO NR-1
2090 GOSUB 5000
2100 FOR J=1 TO N
2110 : RZ(J)=Z(J)
2120 :: Z(J)=Z(J)+ZP(J)*IN/2
2130 : NEXT J
2140 : X=X+IN/2
2150 :GOSUB 5000
2160 FOR J=1 TO N
2170 :: Z(J)=RZ(J)+ZP(J)*IN
2180 : NEXT J
2190 Y(I+1)=Z(I)
2200 : X=X+IN/2
2210 :GOSUB 10000
2220 NEXT I
2230 RETURN
3000 REM
3001 REM *****
3002 REM * RUNGE-KUTTA PROCEDURE *
3003 REM *****
3004 REM
3005 N$="RUNGE-KUTTA PROCEDURE" :GOSUB 15000

```

```

3006 X=XA
3010 FOR J=1 TO N
3020 : Y(J)=Z(J)
3030 NEXT J
3040 GOSUB 10000: REM *   OUTPUT *
3050 FOR I= 1 TO NR-1
3060 GOSUB 5000
3070 :FOR J=1 TO N
3080 : RZ(J)=Z(J)
3090 ::K1(J)=IN*ZP(J)
3100 :: Z(J)=RZ(J)+K1(J)/2
3110 : NEXT J
3120 : X=X+IN/2
3130 :GOSUB 5000
3140 FOR J=1 TO N
3150 ::K2(J)=IN*ZP(J)
3160 :: Z(J)=RZ(J)+K2(J)/2
3170 : NEXT J
3180 :GOSUB 5000
3190 FOR J=1 TO N
3200 ::K3(J)=IN*ZP(J)
3210 ::Z(J)=RZ(J)+K3(J)
3220 : NEXT J
3225 : X=X+IN/2
3227 :GOSUB 5000
3228 : FOR J = 1 TO N
3230 ::K4(J)=IN*ZP(J)
3250 ::Z(J)=RZ(J)+(K1(J)+2*(K2(J)+K3(J))+K4(J))/6
3270 :GOSUB 10000
3280 NEXT I
3290 RETURN
5000 REM
5001 REM *****
5002 REM *                                     *
5003 REM *   ENTER SYSTEM EQUATIONS         *
5004 REM *                                     *
5005 REM *           HERE                     *
5006 REM *****
5020 REM
5030 ZP(1)=Z(2)
5040 ZP(2)=-Z(1)+(0.1*(1-(Z(1)*Z(I))))*Z(2)
5050 RETURN
10000 REM
10010 REM *****
10020 REM *           OUTPUT                   *
10025 REM *****
10027 REM
10030 PRINT "{CLR}"

```

```

10040 PRINT "{GRY2} X" TAB(10) "Y(1)" TAB(23) "Y(2)"; "{C/DN}"
10045 X=INT(X*100+.5)/100
10050 PRINT "{WHT}"; X; TAB(9) Z(1); TAB(22) Z(2); "{C/DN}{C/DN}"
10060 PRINT "          PRESS A KEY"
10070 GET A$:IF A$="" THEN 10070
10080 RETURN
15000 REM
15010 REM *****
15020 REM *          INPUT          *
15025 REM *****
15027 REM
15030 PRINT "{CLR}"; N$; "{C/DN}{C/DN}"
15040 INPUT "{WHT}ORDER OF THE SYSTEM:"; N
15050 IF N<=0 THEN 15010
15060 INPUT "{C/DN}{C/DN}START VALUE OF INTERGRATION"; XA
15070 INPUT "{C/DN}{C/DN}END VALUE OF INTERGRATION"; XE
15080 IF XE=XA THEN 15070
15090 INPUT "{C/DN}{C/DN}INTERVAL"; IN
15100 IF IN<=0 THEN 15090
15110 NR=(XE-XA)/IN
15120 PRINT "{GRY2}{C/DN}{C/DN}START CONDITIONS:{C/DN}"
15130 PRINT "{WHT}{C/DN}FOR X = "; XA " IST:{C/DN}"
15140 FOR I= 1 TO N
15150 :PRINT "{C/DN}Z("; I; ") ="
15160 : INPUT "{C/UP}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}{C/RT}"; Z(I)
15170 NEXT I
15180 RETURN
50000 REM
50010 REM *****
50020 REM *          TITLE DIFF. SYSTEMS          *
50030 REM *****
50040 REM
50050 POKE 214,11:POKE211,9:SYS 58732
50060 PRINT "{GRY2}NUMERICAL SOLUTION OF"
50070 POKE 214,13:POKE211,5:SYS 58732
50080 PRINT "{GRY2}DIFFERENTIAL EQUATION SYSTEMS"
50100 FOR I= 1 TO 5000:NEXT:PRINT "{CLR}":POKE198,0
50110 RETURN

```

READY.

5.8 Vector calculation

The dot product of two vectors A_1 and A_2 of the n -dimensional real vector space with $A_1=(A_{11},A_{21},\dots,A_{n1})$ and $A_2=(A_{12},A_{22},\dots,A_{n2})$ is defined as:

$$A_{11} * A_{12} + A_{21} * A_{22} + \dots + A_{n1} * A_{n2} = \sum_{i=1}^N A_{i1} * A_{i2} \quad (1)$$

and is usually shortened as $\langle A_1, A_2 \rangle$

The dot product is therefore a real number.

the following control sum is $A_1, A_2, A_3 \in \mathbb{R}^3, c \in \mathbb{R}$

(a) Commutative law: $\langle A_1, A_2 \rangle = \langle A_2, A_1 \rangle$

(b) Associative law for multiplication by a real scalar:

$$c * \langle A_1, A_2 \rangle = \langle c * A_1, A_2 \rangle$$

(c) Distributive law: I $\langle A_1, (A_2 + A_3) \rangle = \langle A_1, A_2 \rangle + \langle A_1, A_3 \rangle$

$$\text{II } \langle (A_1 + A_2), A_3 \rangle = \langle A_1, A_3 \rangle + \langle A_2, A_3 \rangle$$

(d) $\langle A_1, A_2 \rangle = 0$ if A_1 is perpendicular to A_2 .

The dot product can be represented differently:

for A_1, A_2 : $\langle A_1, A_2 \rangle = |A_1| * |A_2| * \cos(w)$

where the vector quantities are obtained through:

$$|A_i| = \sqrt{A_{i1}^2 + A_{i2}^2 + \dots + A_{iN}^2}, \quad i=1,2 \quad (2)$$

w is the angle marked off by A_1 and A_2 .

One can also go the other around and calculate the angle based on the dot product:

$$w = \arccos\left(\frac{\langle A_1, A_2 \rangle}{|A_1| |A_2|}\right) \quad (-\pi/2 \leq w \leq \pi/2) \quad (3)$$

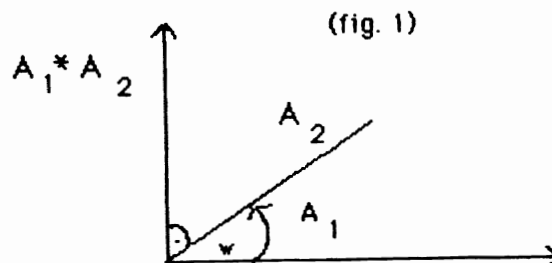
Vector product

For vector (or cross) product the dimension of the real vector space is set $N=3$.

The vector product of any two vectors $A_1=(A_{11}, A_{21}, A_{31})$ and $A_2=(A_{12}, A_{22}, A_{32})$ is defined as:

$$A_1 \times A_2 = (A_{21} \cdot A_{32} - A_{31} \cdot A_{22}, A_{31} \cdot A_{12} - A_{11} \cdot A_{32}, A_{11} \cdot A_{22} - A_{21} \cdot A_{12})$$

This vector $A_1 \times A_2$ is perpendicular to the plane spanned by A_1 and A_2 and forms with this a "right-hand system" (cf. "right-hand rule"):



The following rules apply:

- (a) $A1 \times A2 = -(A2 \times A1)$
- (b) Associative law and
- (c) Distributive law correspond to those for inner product
- (d) $A1 \times A2 = 0$ if $A1$ and $A2$ are parallel to each other.

Furthermore:

$$A1 \times A2 = |A1| * |A2| * \text{ABS}(\sin(w)) \quad (4.1)$$

whereby w is the angle between the two vectors.

From this one recognizes that the amount of the vector product (that is, its length) is equal to the surface area of the parallelogram spanned by $A1$ and $A2$.

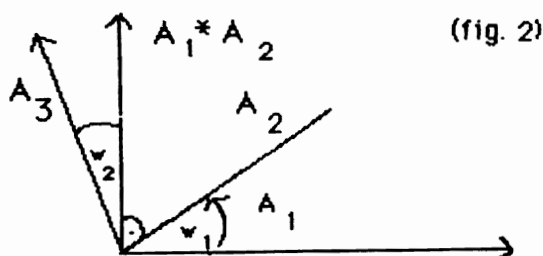
As with the dot product, the angle w can be calculated:

$$w = \arccos \left(\frac{\langle A1, A2 \rangle}{|A1| * |A2|} \right) \quad (3)$$

Triple product

Through the triple product (a combination of the vector and dot products), three vectors are assigned the real number $\langle A1 \times A2, A3 \rangle$.

This number is the volume of the parallelepiped spanned by these vectors.



The angles w_1 and w_2 can be calculated in the same manner as for dot product.

The program

For the calculation of the dot product the components with the same index are multiplied and summed within a loop.

In addition, the amounts of the vectors concerned are calculated. The angle w is calculated in the manner shown above (3) and with the help of the identity:

$$-\arccos(x) = \operatorname{atan}(x / \sqrt{-x^2 + 1}) + \pi/2$$

because the C64 does not have the functions \arcsin and \arccos at its disposal.

The angle is not given in radians but in the usual manner in degrees (hence the factor $180/\pi$).

Example:

dimension: $N = 5$

$A1 = (3, 5, 7, 9, -11)$, $A2 = (2, -4, 6, 8, 10)$

$\langle A1, A2 \rangle = 6 - 20 + 42 + 56 - 110 = -26$

The programn calculates the components of the vector product by means of (4).

The angle w is calculated through equation (3) and with the help of (6) after the necessary vector quantities are known.

Example:

$A1 = (1, 0, 0,)$, $A2 = (0, 1, 0)$

$E = (0, 0, 1,)$ $w = 90$ degrees

In order to calculate the triple product, the vector product is first formed from the first two vectors entered, after which the dot product is calculated based on this result vector and the third vector entered.

Here too the program returns the quantities of the vectors concerned and the angles $w1$ and $w2$ as per figure 2 in addition to the value of the triple product.

```

10 REM *****
11 REM *
12 REM * VECTOR - CALCULATION *
13 REM *
14 REM *
15 REM *****
40 :
50 PRINTCHR$(147):CLR
60 :
200 GOSUB500
210 ONZ%GOSUB 3000,4000,5000,6000,20000
220 GOSUB350:GOTO50
300 FORI=1TO10:GETC$:NEXT
301 GETC$:IFC$THENPRINTTAB(KP%);" {C/LF}";:C=ASC(C$):RETUR
N
310 PRINTTAB(KP%);"_ {C/LF}";:GOTO301
320 :
340 :
350 POKE211,16:POKE214,24:SYS58732
360 PRINT"<KEY>";:FORI=1TO10:GETC$:NEXT
370 WAIT198,1:PRINTCHR$(147):RETURN
380 REM DIMENSIONING SUBROUTINE
390 :
400 DIMA(N,M),W(3)
410 RETURN
420 :
450 PRINT"{RVON}ALL INPUT CORRECT?{RVDF} (Y/N)";:KP%=31
460 GOSUB300:IFC<>83ANDC<>78THEN460
470 RETURN
500 REM MENU 1
600 Z$(1)="-1- DOT PRODUCT"
601 Z$(2)="-2- VECTOR PRODUCT"
602 Z$(3)="-3- TRIPLE PRODUCT"
603 Z$(4)="-4- START PROGRAM "
604 Z$(5)="-5- END PROGRAM "
610 PRINTCHR$(5);TAB(10);"SELECT ...";CHR$(154)
620 PRINT
630 FORI=1TO5:PRINTTAB(10);Z$(I);TAB(30);I:NEXT
640 KP%=31:GOSUB300:IFC<49ORC>53THEN640
650 Z%=VAL(C$)
700 RETURN
940 :
950 RETURN
960 :
969 :
990 REM

```

```

1000 REM DIMENSION      INITIALIZE
1010 REM
1020 IF Z%=1THENGOSUB15500:GOTO1040
1030 IF Z%=2ORZ%=3THENN=3:PRINT:PRINT"-->DIMENSION: ";N:GOSU
B350
1040 IF Z%=1ORZ%=2THENM=2:PRINT"-->NUMBER OF VECTORS: ";M:GO
SUB350
1050 IF Z%=3THENM=3:PRINT"-->NUMBER OF VECTORS: ";M:GOSUB350
1060 :
1100 GOSUB400
1150 PRINTCHR$(147);TAB(8);"INPUT OF  COMPONENTS  ":PRINT
:PRINT
1180 GOSUB16000
1190 GOSUB450
1200 IFC$="N"THENCLR:GOTO10
1300 PRINTCHR$(147);TAB(4);"-----RESULTS-----
--":PRINT:PRINT
1320 RETURN
2510 REM
3000 REM DOT    PRODUCT
3010 REM
3020 :
3030 PRINTCHR$(147);TAB(7);"    D O T    P R O D U C T"
3040 PRINT:PRINT:PRINT
3050 :
3070 GOSUB1000:GOSUB3100:GOSUB11300
3080 X=S/M/(B(1)*B(2)):IFX=1THENW=0:GOSUB11320:RETURN
3085 IFX=-1THEN W=180 :GOSUB11320:RETURN
3090 W=(-ATN(X/SQR(-X*X+1))+ $\pi$ /2)*180/ $\pi$ :GOSUB11320:RETURN
3100 REM
3110 REM DOT.-PROD. COMPONENTS CALCULATIONS
3120 REM
3130 FORJ=1TOM
3140 :FORI=1TON
3150 ::B(J)=B(J)+A(I,J) $\uparrow$ 2
3160 ::S=S+A(I,1)*A(I,2)
3170 :NEXTI
3180 :B(J)=B(J) $\uparrow$ 0.5
3190 :GOSUB11200
3200 NEXTJ
3210 PRINT
3220 :
3230 RETURN
3240 :
3980 REM
3990 REM
4000 REM VECTOR PRODUCT
4010 REM
4020 :

```

```
4030 PRINTCHR$(147);TAB(7);"V E C T O R   P R O D U C T"
4040 PRINT:PRINT:PRINT
4050 :
4070 GOSUB1000
4080 GOSUB4200
4090 RETURN
4200 REM
4210 REM VEK.-PROD. COMPONENT CALCULATIONS
4220 REM
4230 REM
4250 E(1)=A(2,1)*A(3,2)-A(3,1)*A(2,2)
4260 E(2)=A(3,1)*A(1,2)-A(1,1)*A(3,2)
4270 E(3)=A(1,1)*A(2,2)-A(2,1)*A(1,2)
4300 GOSUB11400
4310 :
4320 GOSUB3100
4330 :
4400 FORI=1TON
4410 :BE=BE+E(I)2
4420 NEXTI
4430 :
4440 BE=BE↑0.5
4450 :
4460 GOSUB11460
4470 IFZ%=3THEN GOSUB 4550:RETURN
4480 :
4490 GOSUB 3080
4530 :
4540 RETURN
4550 FORI=1TON
4560 :E=E+E(I)*A(I,3)
4570 NEXTI
4580 RETURN
5000 REM TRIPLE PRODUCT
5010 REM
5020 :
5030 PRINTCHR$(147);TAB(9);"T R I P L E   P R O D U C T"
5040 PRINT:PRINT:PRINT
5050 :
5100 GOSUB 1000
5150 GOSUB4200
5200 X(1)=S/(B(1)*B(2))/3
5210 X(2)=E/(B(3)*BE)
5220 FOR I=1 TO 2
5230 IF X(I)=1 THEN W(I)=0:GOSUB 11350 :GOTO 5300
5240 IF X(I)=-1 THEN W(I)=180:GOSUB 11350 :GOTO 5300
5260 W(I)=(-ATN(X(I)/SQR(-X(I)*X(I)+1))+ $\pi$ /2)*180/ $\pi$  :GOSUB
11350
5300 NEXT I :GOSUB 11500:RETURN
```

```
9990 REM
10000 EM
10010 REM
10090 :
11100 PRINT"COMPONENTS";I;"OF THE VECTOR SUMS:";E(I):PRINT
11110 IF I=NTHEN GOSUB 350:RETURN
11120 RETURN
11130 :
11200 PRINT"QUANTITY OF ";J;"THE VECTORS:";B(J)
11210 PRINT
11220 IF J=MTHEN 350:RETURN
11230 RETURN
11240 :
11300 PRINT
11310 PRINT"DOT PRODUCT: ";S/M : GOSUB 350:RETURN
11320 PRINT
11330 PRINT"ANGLE : ";W;"DEGREES":RETURN
11350 PRINT I"TH ANGLE:"W(I)
11360 PRINT:RETURN
11400 FOR I=1TON
11410 :PRINT"COMPONENTS";I;"OF THE PRODUCT VECTORS: ";E(I)
11420 NEXT I
11430 PRINT:RETURN
11440 :
11460 PRINT"QUANTITY AT RESULT VECTORS: ";BE:RETURN
11500 PRINT "TRIPLE PRODUCT:";E
12000 RETURN
15500 INPUT" {RVON}          DIMENSION ";N:PRINT
15520 RETURN
15600 :
15610 INPUT" {RVON}          DIMENSION ";N:PRINT
16000 FOR J=1TO M:PRINT"ENTER COMPONENT OF THE ";J;"TH VEC
TOR!"
16010 :PRINT
16020 :FOR I=1TON
16030 ::PRINT I"TH COMPONENT OF ";J;"TH VECTOR";
16040 ::INPUT A(I,J)
16050 :NEXT I
16060 :PRINT
16070 NEXT J
16080 RETURN
20000 SYS 64738
```

READY.

5.9 Matrix calculations

An $m \times n$ matrix is a rectangular matrix with m rows and n columns. In this section we will consider the special case when $n=m$, or the matrix is square. An identity matrix is one in which the entries in the principle diagonal of a square matrix are 1 and all other entries are 0. There are various methods which allow one to perform elementary calculations with matrices.

Adding matrices:

The addition of matrices of the same size is associative, commutative, and reversible. The corresponding elements in matrix A and B are added together to form the sum matrix.

Subtracting matrices:

The commutative law does not apply here. Otherwise this operation corresponds to the operation for matrix addition.

Multiplying matrices:

The multiplication of matrices is associative, but not commutative. The distributive law also applies. Multiplication is done as follows: The element in the i th row and k th column of the product matrix results from the dot product of the i th row vector of matrix A and the k th column vector of matrix B.

Inverse of a square matrix:

The inverse of a square matrix A is defined as the matrix which when multiplied by A yields the identity matrix. The method most commonly used for calculating the inverse of a matrix is the Gauss-Jordan elimination procedure with row pivoting.

A few comments should now be made about dealing with matrices which will ease the calculations for you.

Enter the values of a matrix as a data block.

```
2000 REM HERE IS THE DATA FOR THE MATRIX
2010 DATA ORDER, 3
2020 DATA 3,9,12
2030 DATA 5,-3,1
2040 DATA 4,0,-9
```

Column numbering:

```
2100 FOR J=1 TO N
2110 A(N+1,J)=J
2120 NEXT J
2130 RETURN
```

Line exchange:

```
2200 FOR J=1 TO N+1
2210 Z=A(I,J)
2220 A(I,J)=A(K,J)
2230 A(K,J)=Z
2240 NEXT J
2250 RETURN
```

Column exchange:

```
2300 FOR I=1 TO N+1
2310 Z=A(I,J)
2320 A(I,J)=A(I,K)
2330 A(I,K)=Z
2340 NEXT I
2350 RETURN
```

Matrix output:

```
2400 PRINT
2410 FOR J=1 TO N
2420 PRINT A(N+1,J)
2430 NEXT J
2440 PRINT
2450 FOR I=1 TO N
2460 FOR J=1 TO N+1
2470 PRINT A(I,J);
2480 NEXT J
2490 PRINT
2500 NEXT I
2510 RETURN
```

No pivoting:

```
3010 IF A(J,J) <> 0 THEN 3070
3020 FOR J=J+1 TO N
3030 IF A(I,J) <> 0 THEN 3060
3040 NEXT I
3050 PRINT "UNSOLVABLE!" : STOP
3060 GOSUB 2200
3070 RETURN
```

Pivoting:

```
3110 FOR I=J+1 TO N
3120 IF ABS (A(I,J)) > ABS (A(J,J)) THEN 3140
3130 GOSUB 2200
3140 NEXT I
3150 RETURN
```

Total pivoting:

```
3210 AM=0
3220 I=K
3230 J=K
3240 FOR I1=K TO N
3250 FOR J1=K TO N
3260 IF ABS(A(I1,J1)) < ABS(AM) THEN AM=A(I1,J1)
3270 NEXT I1
3280 NEXT J1
3290 IF J=K THEN 3310
3300 GOSUB 2200
3310 IF J=K THEN 3330
3320 GOSUB 2300
3330 RETURN
```

```

1000 REM *****
1010 REM *
1020 REM * ROUTINE FOR
1030 REM *
1040 REM * MATRIX CALCULATION *
1050 REM *
1060 REM *****
1070 REM
1080 REM
1090 REM
1100 REM
1110 REM *****
1120 REM * ADDITION OF MATRICIES *
1130 REM *****
1140 REM
1150 FOR I = 1 TO N
1160 ::FOR J=1 TO N
1170 ::S(I,J)=A(I,J)+B(I,J)
1180 ::NEXT J
1190 NEXT I
1195 END
1200 REM
1210 REM *****
1220 REM * SUBTRACTION OF MATRICIES *
1230 REM *****
1240 REM
1250 FOR I = 1 TO N
1260 ::FOR J=1 TO N
1270 ::S(I,J)=A(I,J)-B(I,J)
1280 ::NEXT J
1290 NEXT I
1295 END
1300 REM
1310 REM *****
1320 REM * MATRIX PRODUCT *
1330 REM *****
1340 REM
1350 FOR I = 1 TO N
1360 ::FOR J=1 TO N
1370 ::::P(I,I)=P(I,I)+A(I,J)*B(I,J)
1380 ::::FOR K= I+1 TO N
1390 :::::P(I,K)=P(I,K)+A(I,J)*B(J,K)
1400 ::::NEXT K
1410 ::::FOR L= I+1 TO N
1420 ::::: IF L=I THEN 1450

```

```
1430 :::::P(L,I)=P(L,I)+A(L,J)*B(J,I)
1440 ::::NEXT L
1450 ::NEXT J
1460 NEXT I
1470 END
1500 REM
1510 REM *****
1520 REM *      INVERT A MATRIX      *
1530 REM *****
1540 REM
1550 FOR I= 1 TO N
1560 A(I,N+1)=1
1570 NEXT I
1580 FOR K=1 TO N
1590 :: FOR I=K TO N
1600 :::: IF ABS(A(I,K))<=ABS(A(K,K)) THEN 1640
1610 :::: FOR J=K TO N*2
1620 :::::Z=A(K,J):A(K,J)=A(I,J):A(I,J)=Z
1630 :::: NEXT J
1640 :: NEXT I
1650 ::IF ABS(A(K,K))<1E-10 THEN END
1660 :: FOR L=1 TO N
1670 :::: IF L=K THEN 1730
1680 :::::Q=A(L,K)/A(K,K)
1690 ::::A(L,K)=0
1700 :::::FOR M=K+1 TO N*2
1710 :::::A(L,M)=A(L,M)-A(K,M)*Q
1720 :::: NEXT M
1730 :: NEXT L
1740 NEXT K
1750 FOR I= 1 TO N
1760 :: FOR J=1 TO N
1770 :::: A(I,N+J)=A(I,N+J)/A(I,I)
1780 ::NEXT J
1790 NEXT I
1800 FOR I=1 TO N
1810 ::A(I,I)=1
1820 NEXT I
1830 END
```

READY.

Chapter 6 : Programs for Chemistry

6.1 Periodic system file program

In section 3.5 you read about relative files and how they are managed. Now you will be able to use the knowledge you gained there. In the sciences one works with data bases quite often. These data bases might contain information about specific molecules, for instance. Literature searches for bibliographical references and citations are also made in this manner. Naturally, professional installed data bases enter into this game. But even the Commodore 64 can do some of this.

Here we will show how a small data base with fast access can be established in principle. For this purpose we want to store the elements of the periodic system as a relative file. Unfortunately, you must first type in all of the known elements with their most important physical attributes. The program "ESTABLISHING A RELATIVE FILE PERIODIC SYSTEM" is used for this.

If you make any errors when entering the data, they can be corrected later.

The storage program is set up such that all of the exactly defined elements will be entered sequentially. If you want to enter only a few elements or perhaps a few from time to time, you can achieve this by changing the input loop in line 2030. Otherwise all of the elements will be asked for from 1 to 105.

If you read your file with the program "READ PERIODIC SYSTEM FILE" after you have entered the data, you will see that the access time is quite fast, an advantage of relative files.

The program for reading the data in contrast to the writing program is not limited to one purpose (displaying the information which you entered previously). It would be possible, for instance, to have this program give additional information obtained through calculation. One such example is the electron composition. If you are acquainted with the basics of chemistry, it would be no problem to realize this for the main group elements.


```

2000 REM
2001 REM *** ESTABLISHING A RELATIVE FILE*
2002 REM *** PERIODIC SYSTEM ***
2005 REM
2010 OPEN 2,8,2,"PERIODIC SYS,L,"+CHR$(59):CLOSE 2
2020 OPEN 15,8,15:REM COMMAND AND ERROR CHANNEL
2030 FOR I=1 TO 105:GOSUB 15000:REM DATA INPUT
2040 GOSUB 10000:REM WRITE DTAT TO FLOOPY DISK
2050 NEXT I:CLOSE 2:CLOSE 15:END
10000 REM
10001 REM *** OUTPUT TO FLOPPY ***
10002 REM
10010 OPEN 2,8,2,"PERIODIC SYS"
10020 PRINT#15,"P"+CHR$(2)+CHR$(I AND 255)+CHR$(I/255)+CHR
$(1):REM POSITIONING
10030 INPUT#15,B,B$:REM READ 'NEW SENTENCE' MESSAGE
10040 CR$=CHR$(13):PRINT#2,N$:CR$;K$;CR$;M;CR$;D;CR$;SM;CR
$;SD;CR$;G
10050 CLOSE 2:RETURN
15000 REM
15001 REM *** INPUT ***
15002 REM
15020 PRINT"{CLR}{WHT}ELEMENT NUMBER";I
15030 PRINT"{C/DN}{C/DN}{GRY2}NAME =";:INPUT N$:N$=LEFT$(N
$,15)
15040 PRINT"{C/DN}SYMBOL =";:INPUT K$:K$=LEFT$(K$,2)
15050 PRINT"{C/DN}MASS =";:INPUT M:M=INT(M*100000)/100000
15060 PRINT"{C/DN}DENSITY=";:INPUT D:D=INT(D*10000)/10000
15070 PRINT"{C/DN}MELTING POINT=";:INPUT SM:SM=INT(SM*10)/
10
15080 PRINT"{C/DN}BOILING POINT=";:INPUT SD:SD=INT(SD*10)/
10
15090 PRINT"{C/DN}GROUP =";:INPUT G$:G$=LEFT$(G$,2)
15100 Z=2:GOSUB 19000:PRINT:PRINT
15110 PRINT "{GRY2}1 NAME          ";N$
15120 PRINT "2 SYMBOL              ";K$
15130 PRINT "3 MASS                ";M
15140 PRINT "4 DENSITY              ";D
15150 PRINT "5 MELTING POINT";SM
15160 PRINT "6 BOILING POINT";SD
15170 PRINT "7 GROUP                ";G$
15180 PRINT "{C/DN}EVERYTHING O.K. ? {WHT} (Y/N) "
15190 GET X$:IF X$="" THEN 15190
15200 IF X$="N" THEN GOSUB 18000:GOTO 15100
15210 RETURN

```

```
18000 REM
18001 REM *** CORRECTION***
18002 REM
18010 PRINT "{C/UP}{GRY2}WHICH ITEM ? {WHT}(1 - 7)  "
18020 GET X$:IF X$="" THEN 18020
18030 IF ASC(X$)<48 OR ASC(X$)>57 THEN 18020
18040 X=ASC(X$)-48
18050 IF X=1 THEN PRINT "{GRY2}NAME";:INPUT N$:N$=LEFT$(N$,15):RETURN
18060 IF X=2 THEN PRINT "{GRY2}SYMBOL ";:INPUT K$:K$=LEFT$(K$,2):RETURN
18070 IF X=3 THEN PRINT "{GRY2}MASS ";:INPUT M:M=INT(M*100000)/100000:RETURN
18080 IF X=4 THEN PRINT "{GRY2}DENSITY";:INPUT D:D=INT(D*100)/100
18090 IF X=5 THEN PRINT "{GRY2}MELTING POINT";:INPUT SM:SM=INT(SM*10)/10:RETURN
18100 IF X=6 THEN PRINT "{GRY2}BOILING POINT";:INPUT SD:SD=INT(SM*10)/10:RETURN
18110 IF X=7 THEN PRINT "{GRY2}GROUP ";:INPUT G$:G$=LEFT$(G$,2):RETURN
18120 GOTO 18020
19000 REM
19001 REM *** ERASE SCRREN AT LINE Z ***
19002 REM
19010 PRINT "{HOME}":FOR ZZ= 1 TO Z-2:PRINT:NEXT:FOR ZZ=Z TO 23
19020 PRINT"                                     ";:NEXT ZZ
19030 POKE 211,0:POKE 214,0:SYS 58732:RETURN
50000 OPEN#13,8,15:INPUT#13,A,B$:CLOSE 13:PRINTA,B$:END
```

READY.

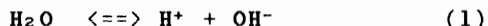
```
2000 REM
2001 REM *****
2002 REM *
2003 REM *      READ PERIODIC SYSTEM      *
2004 REM *
2005 REM *      FILE      *
2006 REM *
2007 REM *****
2010 OPEN 2,8,2,"PERIODIC SYS"
2020 OPEN 15,8,15
2030 PRINT"{CLR}WHICH ELEMENT? ENTER NUMBER";:INPUT I
2040 PRINT#15,"P"+CHR$(2)+CHR$(I AND 255)+CHR$(I/256)+CHR$(
(1)
2050 INPUT#2,N$:INPUT#2,K$:INPUT#2,M
2060 INPUT#2,D:INPUT#2,SM:INPUT#2,SD
2070 INPUT#2,G$
2080 GOSUB 10000:REM OUTPUT DATA
2090 WAIT 198,255:REM WAIT FOR A KEY
2100 CLOSE 2:CLOSE 15
2110 PRINT"{CLR} ANOTHER ELEMENT? {WHT} (Y/N) "
2120 GET X$:IF X$="Y" THEN RUN
2130 IF X$="N" THEN END
2140 GOTO 2120
10000 REM
10001 REM *** SCREEN OUTPUT ***
10002 REM
10010 PRINT"{CLR}{WHT}";N$:PRINT"{GRY2}NUMBER",I
10020 PRINT"CHEM. SYMBOL ",K$
10030 PRINT"MASS ",M
10040 PRINT"DENSITY",D
10050 PRINT"MELTING POINT",SM
10060 PRINT"BOILING POINT",SD
10070 PRINT"GROUP ",G$
10080 RETURN
```

READY.

6.2 pH value calculations

The chemical term "pH value" is one that has widespread use outside of the natural sciences. So we encounter it as the "friendly/kind-to-your-skin" pH value in cosmetics as well as in the search for causes of tree death. In order to reach a better understanding of the importance of the pH value in biological, in a broader sense, chemical equilibrium, we want to recall a few things:

The dissociation of water follows from:



whereby the proton in the simplest hydrolized form



occurs as an oxide ion.

The following relationship holds for this equilibrium reaction based on the mass effect law

$$K = \frac{[\text{H}^+] [\text{OH}]^-}{\text{H}_2\text{O}} \quad (3)$$

The square brackets denote the molar concentrations. Since the portion of undissociated water molecules can be seen as the constant 55.4 mol/l, we multiply this factor in the equilibrium constants and get

$$K_w = [\text{H}^+] [\text{OH}]^- \quad (4)$$

the ion product of water.

The equilibrium given in (1) is naturally dependent on temperature and moves somewhat to the right. The concentrations of the H_3O^+ and the OH^- ions amount to 10^{-7} mol/l at 22 degrees Celsius. For K_w one receives 10^{-14} mol²/l². Since this number is a bit unwieldy, the biochemist Soerensen formulated the following simplified representation:

"The pH value is the negative base-10 logarithm of the oxygen ion concentration."

Stated mathematically:

$$[H_3O^+] = 10^{-pH} \quad (5)$$

or

$$pH = -\log [H_3O^+] \quad (6)$$

The obvious advantage of this definition is the consequence that the wide range of over 10 orders of magnitude is reduced to a positive real number between 0 and 14. Special systems outside these boundaries represent a subject of their own.

Subsequently, the following results from 94)

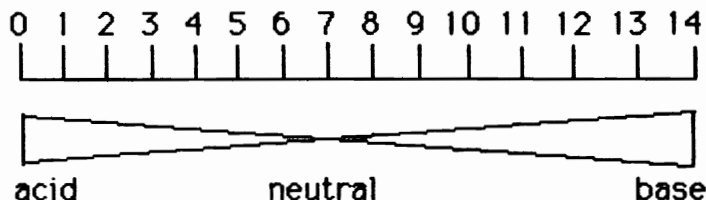
$$pK_w = pH + pOH$$

or

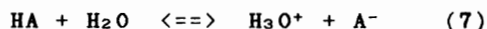
$$pK_w = 7 + 7 \quad \text{for } T = 22 \text{ degrees Celsius}$$

This method of formulation allows us to escape the drudgery of dragging dimensions around.

In this manner we can now arrange a satisfactory pH value scale:



As a result of what has been said, it is only logical that the pH value changes as soon as protonized acids are added to the water. Such an acid in its simplest form obeys the acid equilibrium



with the characteristic acid constants

$$K = \frac{[\text{H}_3\text{O}^+][\text{A}^-]}{[\text{HA}]} \quad (8)$$

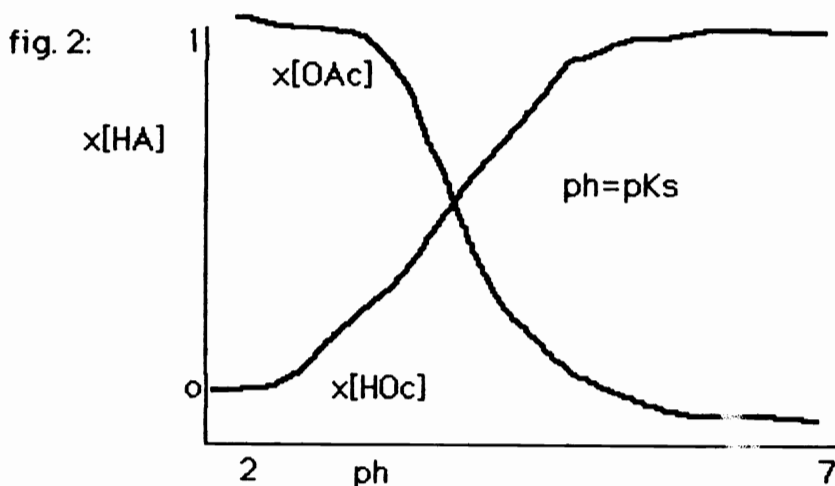
as for (6) we get

$$\text{p}K_s = -\log K_s \quad (9)$$

the acid exponents.

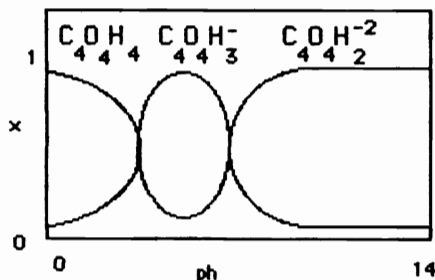
A complicated relationship applies for an n-valent acid, that is, an acid which protonizes in several stages, which will not be discussed in full detail here.

A graph offers the possibility of a visual explanation.



The graph shows the course of the pH value of a single-valent acid against the concentration.

Correspondingly changed, our graph of a multi-valent acid looks like this:



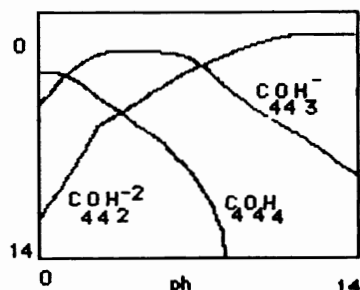
The individual protolysis levels clearly divide the pH graph into many corresponding existence ranges.

In the places where $\text{pH} = \text{pK}_s$ applies, the equilibrium concentrations of the individual parts can be determined.

It is often necessary to select the accuracy in order to match a convenient figure scale. A logarithmic representation is recommended in many cases. Here the pH value is plotted against $-\log c$. If we represent figure 3 logarithmically, we get

$$[\text{H}_{n-1}\text{A}^+] = \frac{[\text{H}_3\text{O}^+]^{n-1} * [\text{H}_n\text{A}] * \prod_{j=1}^i K_{sj}}{\sum_{r=0}^n \left[[\text{H}_3\text{O}^+]^{n-r} * \prod_{j=1}^r K_{sj} \right]} \quad (10)$$

It should be noted that, mathematically, every equilibrium concentration of the divisions within an equation can be written as a function of the hydrogen ion concentration. The solution of the corresponding equation system permits the exact determination of the individual concentrations:



With the introduction of this last monster equation, it is time to let the C64 take over. The following program gives us the necessary values for creating a pH graph. We

can select the acid fraction x (linear representation) or the concentration $-\log c$ (logarithmic representation).

This program offers many interesting possibilities for expansion and extension. Routines can be inserted without difficulty which find the various isoelectric points, for instance, or which simply print the corresponding curves.

```

1250      read the acid exponents  $pK_{S_i}$ 
1300      convert to  $K_{S_i}$ 
1350-1450 calculate  $\prod_{j=1}^i K_{S_j}$ 
1500      select representation type
1720      oxygen ion concentration
1820      calculate  $\left. \begin{array}{l} \\ \\ \end{array} \right\} \text{ of } (10)$ 
1920      calculate  $\left. \begin{array}{l} \\ \\ \end{array} \right\}$ 
1930       $x[H_{n-1}A^+] = \frac{[H_{n-1}A^+]}{CA}$  where CA is the total concentration.
1940       $-\log [H_{n-1}A^+]$ 
1985-2000 output results
15000-      input of CA, NW, diagram type

```

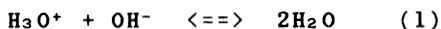
```
10 REM *****
11 REM *
12 REM * PH-CHART
13 REM *
14 REM *
15 REM *****
20 :
30 :
50 PRINTCHR$(147)
440 :
450 DIM PK(10),KS(10),C(10),LC(10)
470 DEF FNR(X)=INT(1000*X+0.5)/1000
520 GOSUB15000:GOSUB 1200:END
600 PRINT:PRINTTAB(16); "<KEY>"
610 GETC$: IF C$ THEN PRINTCHR$(147):RETURN
620 GOTO 610
1200 FOR I=1 TO NW
1250 : GOSUB 15400
1300 : KS(I)=10↑-PK(I)
1350 : P(I)=1
1400 : FOR K=1 TO I
1410 : : P(I)=P(I)*KS(K)
1420 : NEXT K
1450 NEXT I
1500 GOSUB 15430
1520 :
1600 P(0)=1
1700 FOR H=0 TO 14 STEP 0.5
1720 : CH=10↑-H
1740 : NS=0
1800 : FOR L=0 TO NW
1820 : : NS=NS+CH↑(NW-L)*P(L)
1900 : : FOR U=0 TO NW
1920 : : : C(U)=CH↑(NW-U)*CA*P(U)/NS
1930 : : : XC(U)=C(U)/CA:XC(U)=FNR(XC(U))
1940 : : : LC(U)= LOG(C(U))/LOG(10)
1960 : : NEXT U
1980 : NEXT L
1985 PRINT"*****PH-VALUE:";H
1990 : PRINT
2000 : FOR R=0 TO NW
2010 : : IF SV=1 THEN GOTO 2030
2020 : : PRINT"***** X(";R;"{C/LF}) :";XC(R):GOTO2040
2030 : : PRINT"*****-LOG C(";R;"{C/LF}) :";LC(R)
2040 : NEXT R
```

```
2045 PRINT
2048 : IFH/1.5=INT(H/1.5) THEN GOSUB 600
2050 NEXT H
2100 RETURN
15000 REM INPUT SUBROUTINE
15100 INPUT "{RVON}CONCENTRATION OF ACID      :";CA
15150 PRINT
15200 INPUT "{RVON}ACIDITY                      :";NW :PRINTCH
R$(147)
15250 RETURN
15400 PRINT "{RVON}ACID CONSTANT      ";I;"{C/LF}      : ";;INP
UT PK(I):PRINT
15420 RETURN
15430 PRINTCHR$(147):PRINT
15440 PRINTCHR$(5);"WHAT WOULD YOU LIKE TO CALCULATE?"
15445 PRINT
15450 PRINTTAB(7);"{RVON}LINEAR REPRESENTAION?-0- {RVOF}"
15470 PRINT"{RVON}LOGARITHMIC REPRESENTATION ?-1- {RVOF}"
15480 PRINTTAB(27);:INPUT SV:PRINTCHR$(154);CHR$(147)
15500 RETURN

READY.
```

6.3 Titration

Let us now turn from pH value calculations to another important branch of analytic chemistry, the acid-base titrations. A characteristic of such a procedure is the proton transfer which occurs between the principle materials concerned in a diluted medium. This, in regards to the changing pH value as a result of the neutralization reaction, suffices for the general formulation



The goal of such a titration is the volumetric analysis of the reaction partners present. The end of the titration is reached when the substance in question has been completely assimilated in the test solution. This point is generally found with the help of a color indicator.

With the following we want to simulate such a titration. But have no fear, what is quite damp in practice will by a dry run for our C64. In order for the unpracticed to understand the program, a bit of theory is in order.

Let us take a look at the following titration of an acid. The reaction progress can be indicated by the ususally S-shaped linear titration curve.

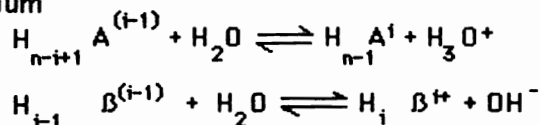
In our case, the degree of the transformation is

$$t = \frac{[\text{B}]_A}{[\text{HA}]_A} \quad (2), \text{ at which } []_A$$

where $[]_A$ indexed the initial concentrations.

IF we look at the general procedure of a titration of an n -valent acid with an m -valent base, we find that the acid-base equilibrium is

acid-base equilibrium



$$\text{bel } i=1,2,3 \dots n(m) \quad (3)$$

for oxygen in concentration, it follows that

$$[H_3O^+] = [OH^-] - \sum_{i=1}^n i * [H_i B^i] + \sum_{i=1}^n i * [H_{n-i} A^i] \quad (4)$$

with the definition equations for the equilibrium concentration

$$[H_i B^i] = \frac{[H_3O^+]^i * [B]_A * K_W^{m-1} * \prod_{j=1}^i K_{B_j}}{\sum_{i=0}^m \left[[H_3O^+]^{m-i} * K_W^i * \prod_{j=1}^{m-i} K_{B_j} \right]} \quad (5)$$

$$[H_{n-i} A^i] = \frac{[H_3O^+]^{n-i} * [H_n A]_A * \prod_{j=1}^i K_{S_j}}{\sum_{i=0}^n \left[[H_3O^+]^{n-i} * \prod_{j=1}^i K_{S_j} \right]} \quad (6)$$

one gets from the 4

$$\begin{aligned}
 [\text{OH}^-] - [\text{H}_3\text{O}^+] - \frac{[\beta]_A * \sum_{i=1}^m \left[[\text{H}_3\text{O}^+]^i * i * K_W^{m-1} * \prod_{j=1}^i K_{B_j} \right]}{\sum_{i=0}^m \left[[\text{H}_3\text{O}^+]^{m-i} * K_W^i * \prod_{j=1}^{m-i} K_{B_j} \right]} \\
 + \frac{[\text{H}_n\text{A}]_A * \sum_{i=1}^n \left[[\text{H}_3\text{O}^+]^{n-i} * i * \prod_{j=1}^i K_{S_j} \right]}{\sum_{i=0}^n \left[[\text{H}_3\text{O}^+]^{n-i} * \prod_{j=1}^i K_{S_j} \right]} = 0
 \end{aligned}$$

division by $[\text{H}_n\text{A}]_A$ yields

$$\begin{aligned}
 \frac{[\text{OH}^-] - [\text{H}_3\text{O}^+]}{[\text{H}_n\text{A}]_A} + \frac{\sum_{i=1}^n \left[[\text{H}_3\text{O}^+]^{n-i} * i * \prod_{j=1}^i K_{S_j} \right]}{\sum_{i=0}^n \left[[\text{H}_3\text{O}^+]^{n-i} * \prod_{j=1}^i K_{S_j} \right]} \\
 = \frac{[\beta]_A * \sum_{i=1}^m \left[[\text{H}_3\text{O}^+]^i * i * K_W^{m-1} * \prod_{j=1}^i K_{B_j} \right]}{[\text{H}_n\text{A}]_A * \sum_{i=0}^m \left[[\text{H}_3\text{O}^+]^{m-i} * K_W^i * \prod_{j=1}^{m-i} K_{B_j} \right]} \quad (8)
 \end{aligned}$$

$$\frac{[\text{OH}^-] - [\text{H}_3\text{O}^+]}{[\text{H}_n\text{A}]} = \frac{K_W / [\text{H}_3\text{O}^+] - [\text{H}_3\text{O}^+]}{[\text{H}_n\text{A}]_A} \quad (9)$$

we get

(10)

$$\begin{aligned} \sum_{i=0}^m [\text{H}_3\text{O}^+]^{m-i} * K_W^i * \prod_{j=1}^{m-i} K_{B_j} &= \sum_{i=0}^m [\text{H}_3\text{O}^+]^i * K_W^{m-i} * \prod_{j=1}^i K_{B_j} \\ t &= \frac{\sum_{i=0}^m [\text{H}_3\text{O}^+]^i * K_W^{m-i} * \prod_{j=1}^i K_{B_j}}{\sum_{i=1}^m [\text{H}_3\text{O}^+]^i * K_W^{m-i} * \prod_{j=1}^i K_{B_j}} * \\ &\left[\frac{K_W / [\text{H}_3\text{O}^+] - [\text{H}_3\text{O}^+]}{[\text{H}_n\text{A}]_A} + \right. \\ &\left. \frac{\sum_{i=1}^n [\text{H}_3\text{O}^+]^{n-i} * \prod_{j=1}^i K_{S_j}}{\sum_{i=0}^n [\text{H}_3\text{O}^+]^{n-i} * \prod_{j=1}^i K_{S_j}} \right] \quad (11) \end{aligned}$$

Program comments:

120 ion product of water
520 call the input routine
1000-1350 calculation of the
1400-1650 calculation of the
1700-2400 calculation of the
1820 hydrogen ion concentration
1920 calculation

1940 calculation

2020 calculation

2080 calculation

2100 calculation of t


```
10 REM *****:****
11 REM * *
12 REM * TITRATION *
13 REM * *
14 REM * *
15 REM *****
20 :
50 PRINTCHR$(147)
60 :
100 DIM PB(10),R(10),KB(10),KS(10)
110 :
120 KW=1E-14
520 GOSUB 15000
530 GOSUB 1200:END
550 :
600 PRINT:PRINTTAB(16);"<KEY>"
605 P=0:FOR E=1TO10:GETC$:NEXT
610 GETC$:IF C$ THEN PRINTCHR$(147):RETURN
620 GOTO 610
1000 REM CALCULATE PROD. ACID
1100 :
1200 FOR I=1 TO NW
1250 : GOSUB 15400
1260 : KS(I)=10-PS(I)
1270 : P(I)=1
1300 : FOR K=1 TO I
1310 : : P(I)=P(I)*KS(K)
1320 : NEXT K
1350 NEXT I
1380 :
1400 REM CALCULATE PROD. BASE
1420 :
1500 FOR I=1 TO MW
1550 : GOSUB 15500
1560 : KB(I)=10-PB(I)
1570 : R(I)=1
1600 : FOR K=1 TO I
1610 : : R(I)=R(I)*KB(K)
1620 : NEXT K
1650 NEXT I
1680 :
1700 REM CALCULATE TITRATION DEGREE
1720 :
1800 FOR H=0 TO 14 STEP 0.1
1810 H=INT(10*H+0.5)/10:P=P+1
```

```

1820 : CH=10↑-H
1840 : A1=KW↑MW
1850 : A2=0
1900 : FOR I=1 TO MW
1920 : : A1=A1+CH↑I*R(I)*KW↑(MW-I)
1940 : : A2=A2+CH↑I*I*R(I)*KW↑(MW-I)
1950 : NEXT I
1960 :
1980 : A3=0
2000 : FOR I=1 TO NW
2020 : : A3=A3+CH↑(NW-I)*I*P(I)
2050 : NEXT I
2080 : A4=CH↑NW+A3
2100 : T=A1/A2*((KW/CH-CH)/CA+A3/A4)
2110 : T=INT(100*T)/100
2200 : GOSUB 10000
2250 IF P=3 THEN GOSUB 600
2300 NEXT H
2400 RETURN
10000 REM OUTPUT
10200 PRINT"*****PH-VALUE:";H
10250 IF T<0 THEN T=0
10260 IF T>2 THEN T=2
10300 PRINT"*****TITRATION DEGREE";T
10350 PRINT:PRINT
10400 RETURN
14900 :
15000 REM INPUT SUBROUTINE
15100 INPUT"{RVON}CONCENTRATION OF ACID      ":";CA
15150 PRINT:PRINT
15200 INPUT"{RVON}WERTIGKEIT OF ACID        ":";NW
15250 PRINTCHR$(145);TAB(15);"{RVON}BASE      ":";
15300 INPUT MW
15320 :
15350 RETURN
15380 :
15400 PRINT"{RVON}ACID EXPONENT";I;": ":";:INPUT PS(I)
15410 PRINT
15420 RETURN
15430 :
15440 :
15500 PRINT"{RVON}BASE EXPONENT  ";I;": ":";:INPUT PB(I)
15510 PRINTCHR$(147)
15520 :
15530 RETURN
22500 : CO=1E7*H/3E6:CP=INT(1E7*H/3E6):IFCO=(CP) THEN GOSUB
600

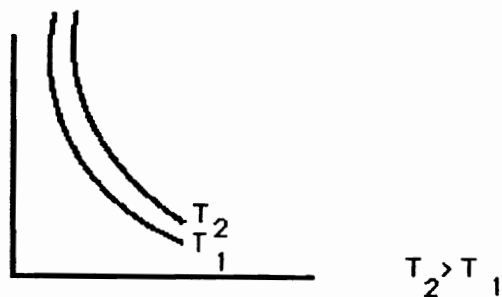
```

6.4 Thermodynamics of real and ideal gases

The thermal behavior of a gas is of great importance for many natural science and technical areas. A comprehensive thermodynamic description of a particular system is often impossible without considering the fundamental gas laws. These are as applicable in complex gaseous interchanges just as in a pressure cooker.

It is no wonder that the first gas law was already formulated in the 17th century. This involved the work of Boyle and Mariotte and the relationship between pressure and volume changes of an ideal gas (1664, 1672).

fig. 1



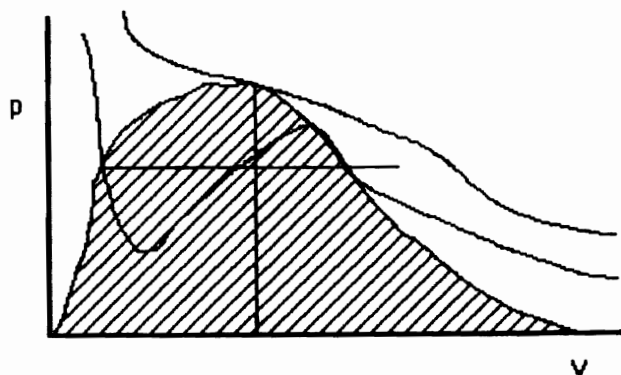
$$P \cdot V = \text{const.}$$

(1)

The condition of an ideal gas is thereby characterized that intermolecular interactions are disregarded and the molecule itself is viewed as the mass product.

The isothermal curve of a real gas (carbon dioxide), however, shows how much more difficult it is to describe a real gas.

fig. 2



In connection with the laws concerning the influence of the size n (number of particles) and T (temperature) lead to the following relationship for ideal gases:

$$PV = nRT \quad (2)$$

where R is the constant relationship PV/T for 1 mole under standard temperature and pressure.

While the exact calculations are possible for ideal gases, this is not the case for real gases because of the "individuality" of the different molecules. For real gases we must take into account phenomenological pieces of knowledge which are built into approximation equations, the most well-known of which is the van der Waal condition equation.

$$\left(p + \frac{an^2}{V^2}\right) (V - nb) = nRT \quad (3)$$

Through the introduction of the material-dependent constants a and b , the volume of the particles, the pertaining repulsion effects, as well as the pressure-reduction-producing attraction of the particles is taken into account.

The condition equation for ideal gases is a marginal law which is never used with real gases and approximated only for very low pressures and high temperatures.

A calculation of the condition sizes from the equation for ideal gases is in general unproblematical. Otherwise these relationships are the same as the corresponding solutions of the van der Waal condition equation. While the pressure can be calculated quite easily through the transformation

$$p = \frac{n \cdot R \cdot T}{V - nb} - a \left(\frac{n}{V} \right)^2 \quad (4)$$

the determination of the volume requires more effort because this calculation leads to a third-degree equation in V :

$$V^3 - \left(b \cdot n + \frac{n \cdot R \cdot T}{p} \right) \cdot V^2 + \frac{a n^2}{p} \cdot V - \frac{a \cdot b \cdot n^3}{p} = 0 \quad (5)$$

We can determine the zeros of this equation with help of the Newton approximation method which was explained earlier in the book.

We proceed in a similar manner to calculate the mole number n , for which we create the equation

$$n^3 - \frac{V \cdot n^2}{b} + \left(\frac{pV^2}{a} + \frac{R \cdot T \cdot V^2}{a \cdot b} \right) \cdot n - \frac{p \cdot V^2}{a \cdot b} = 0 \quad (6)$$

We get the first approximation value for n or V through the solutions of the ideal gas equation. The accuracy of the iteration can be selected as desired.

The knowledge of the constants a and b is presumed for all calculations. An expansion of the program could consist of integrating modules which calculate the covolume based on the molecule radius or both constants from the critical data, for example.

Program explanation

Line

100 rounding function

130 def.: volume function

131 def.: particle function

155 def.: 1st derivative of the volume function

160 def.: 1st derivative of the particle function

180 def.: gas constant in l bar/(k mol)

200-450 menu

1000-2600

calculations

Because the van der Waal equation reduces to the condition equation for ideal gases when $a=b=0$, we will only use the former. With the help of this

equation we determine pressure as well as temperature.

1500-2600

Calculation of the volume and the number of particles using Newtonian zero-approximation. The solutions to the ideal gas equation serve as the initial values. It is possible to control the accuracy of the iteration through the variable S%.

11900-12100

The step-wise approximation can be followed in the output.

17000-17950

Input of parameter and constants. The constants and b are set to zero or input based on the designating variable LN%.

```

10 REM *****
11 REM *
12 REM * REAL AND IDEAL GASES
13 REM *
14 REM *
15 REM *****
30 DIM V(100),NM(100),A(100)
50 PRINTCHR$(147)
95 REM ROUNDING
100 DEF FNR(X)=INT(1E4*X+0.5)/1E4
101 :
119 REM
120 REM FUNCTIONS OF VOLUME, MOLE.
121 REM
125 :
130 DEF FNF(V)=V↑3-(NM*KB+NM*R*GT/GP)*V↑2+KA*NM↑2/GP*V-KA
*KB*NM↑3/GP
131 DEFFNG(NM)=NM↑3-GV/KB*NM↑2+(GP*GV↑2/KA+R*GT*GV↑2/(KA
*KB))*NM-GP*GV↑3/(KA*KB)
149 REM
150 REM 1ST DETERMINATION OF VOL, MOLE. FUNCTIONS
151 REM
155 DEFFNT(NM)=3*NM↑2-GV/KB*2*NM+GP*GV↑2/KA+R*GT*GV↑2/(KA
*KB)
160 DEF FND(V)=3*V↑2-(NM*KB+NM*R*GT/GP*2*V)+KA*NM↑2/GP
165 :
180 R=0.083144
185 :
200 REM PARAMETER, CONSTANTS
201 Z$(1)="COEFFICIENT A "
202 Z$(2)="COEFFICIENT B "
203 Z$(3)="PRESSURE (P) "
204 Z$(4)="VOLUME (V) "
205 Z$(5)="TEMPERATURE (T) "
206 Z$(6)="MOLE NUMBER (N) "
299 REM
300 REM MENUE
301 REM
320 PRINT
325 PRINT
330 PRINT
350 PRINTTAB(15);" "+Z$(3)+"-1-"
355 PRINTTAB(15);" "+Z$(4)+"-2-"
360 PRINTTAB(15);Z$(5)+"-3-"
365 PRINTTAB(15);" "+Z$(6)+"-4-"

```



```
370 PRINT
380 PRINT" {RVON}   WHAT WOULD YOU LIKE TO CALCULATE";
390 INPUTK%
400 IFK%<10RK%>4THEN50
405 Z%=K%+2
410 PRINT" {RVON} IDEAL OR REAL GAS (I/R)";
420 INPUT LN$
430 IF LN$<>"I" AND LN$<>"R" THEN 50
440 IF LN$="I" THEN LN%=2: GOTO 500
450 LN%=1
499 REM
500 REM CALL INPUT SUBROUTINE
501 REM
510 GOSUB15000
519 REM
520 REM ENRTY POINT OF PRESSURE SUBROUTINE
521 REM
529 REM
530 REM CALL OUTPUT SUBROUTINE
531 REM
540 GOSUB 10000
550 :
600 GOSUB 14000
700 :
999 REM
1000 REM CALCULATIONS
1001 REM
1010 :
1099 REM
1100 REM PRESSURE CALCULATIONS
1101 REM
1110 :
1200  $GP = NM * R * GT / (GV - NM * KB) - KA * (NM / GV) \uparrow 2$ 
1210 GP=FNR (GP)
1230 :
1250 RETURN
1280 :
1299 REM
1300 REM TEMPERATURE CALCULATIONS
1301 REM
1350 :
1400  $GT = 1 / (NM * R) * (GP + KA * (NM / GV) \uparrow 2) * (GV - NM * KB)$ 
1410 GT=FNR (GT)
1430 :
1450 RETURN
1480 :
1499 REM
1500 REM VOLUME CALCULATIONS
```

```
1501 REM
1550 :
1590 REM IDEAL GAS
1600 GV=NM*R*GT/GP:GV=FNR(GV)
1610 IF LN%=2 THEN RETURN
1640 REM REAL GAS
1650 REM NEWTON APPROXIMATION
1660 REM
1670 :
1680 V(0)=GV
1690 I=0
1700 PRINT" (RVON) ACCURACY TO HOW MANY PLACES ";
1710 INPUTS%
1790 REM TERTATION LOOP
1800 X=V(I):A(I)=V(I)
1820 DK=(-1)*FNF(X)/FND(X)
1850 :
1900 V(I+1)=V(I)+DK
1930 C=ABS(V(I+1)-V(I))
1950 IF C<10-S% THEN K=I+1:RETURN
1980 I=I+1
2000 GOTO 1800
2099 REM
2100 REM MOLE NUMBER CALCULATION
2101 REM
2150 :
2200 NM= GP*GV/(R*GT):NM=FNR(NM)
2210 IF LN%=2 THEN RETURN
2230 :
2240 REM REAL GAS
2250 REM USING NEWTON APPROXIMATION
2260 REM
2270 :
2280 NM(0)=NM
2290 I=0
2310 S%=4
2390 REM ITERATION LOOP
2400 X=NM(I):A(I)=NM(I)
2420 DK= (-1)*FNG(X)/FNT(X)
2450 :
2500 NM(I+1)=NM(I)+DK
2530 C=ABS(NM(I+1)-NM(I))
2550 IF C<10-S% THEN K=I+1:RETURN
2580 I=I+1
2600 GOTO 2400
2650 :
9999 REM
10000 REM OUTPUT
```

```
10001 REM
10010 :
10050 PRINTTAB(15);"RESULTS "
10060 PRINT:PRINT
11000 PRINT"*****"+Z$(3)+"      : ";GP
11050 PRINT
11100 PRINT"*****"+Z$(5)+" : ";GT
11150 PRINT
11180 IF Z%=4 AND LN%=1 THEN A(I)=NM(I):GOSUB 12000:GOTO 11300
11200 PRINT"*****"+Z$(4)+"      : ";GV
11250 PRINT
11280 IF Z%=6 AND LN%=1 THEN A(I)=NM(I):GOSUB 12010:GOTO 1
1350
11300 PRINT"*****"+Z$(6)+"      : ";NM
11350 PRINT
11400 IF LN%=2 THEN RETURN
11450 PRINT"*****COEFFICIENTES";SPC(6);"A: ";KA
11500 PRINTTAB(24);"B: ";KB:RETURN
11600 GOSUB 14600
11899 REM
11900 REM OUTPUT ITERATION STEPS
11901 REM
12000 PRINT"*****"+Z$(4)+" : ";GOTO 12020
12010 PRINT"*****"+Z$(6)+" : "
12020 FOR I=0 TO K STEP 2
12050 : PRINTTAB(8);A(I);TAB(25);A(I+1)
12080 IF PEEK(214)>15 THEN GOSUB 14600
12100 NEXT
12150 FOR I=0 TO 39:PRINT"—";:NEXT:PRINT
12200 RETURN
13000 :
13100 :
13499 REM
13500 REM END PROGRAMM ?
13501 REM
13600 :
14000 INPUT"(RVON)MORE CALCULATIONS? (Y/N) ";C$
14010 IF C$<>"Y" AND C$<>"N" THEN PRINTCHR$(147):GOTO 1400
0
14030 IF C$="Y" THEN CLR: GOTO 10
14500 END
14600 POKE 211,15:POKE 214,22:SYS56732:PRINT"<KEY>":WAIT 1
98,1:PRINTCHR$(147)
14610 RETURN
14999 REM
15000 REM INPUT PARAMETERS AND OTHERS
15001 REM
15050 :
```

```
15100 GOSUB17000:RETURN
17000 PRINTCHR$(147):PRINT:PRINT
17100 IFLN%=2THENP(1)=0:P(2)=0:L=3:GOTO17500
17150 :
17200 L=1
17250 :
17500 FORI=LTO6
17600 :IFI=Z%THEN17900
17700 :PRINT" {RVON} ";Z$(I);
17800 :INPUTP(I)
17850 PRINT
17900 NEXT
17920 :
17950 KA=P(1):KB=P(2):GP=P(3):GV=P(4):GT=P(5):NM=P(6)
18000 PRINTCHR$(147)
18100 ONK%GOSUB1100,1500,1300,2100
18190 :
18200 RETURN
```

READY.

6.5 The HMO procedure

When is a bond between atoms achieved? Why does a molecule appear in this or that geometric structure? In short, what really happens between the electrons, goes the question which chemists as well as physicists have not allowed to die since the start of this century.

After it was taken for granted that energetic criteria are concerned in the forming of a bond, it wasn't long before valuable suppositions could be checked with the help of quantum mechanics.

Based on the present state of affairs, we can conclude that based on the connection to the knowledge that an exact mathematical calculation is not possible for multiple electron systems, all approximation procedures can be placed in one of two categories.

While the valence bond or VB method finds consideration chiefly in practical deliberations, the molecule orbital or MO method dominates in most areas of chemistry. Its success is founded on the systematics of describing a complex molecular orbital without excessive consumption of time with a combination of atom orbitals similar to oxygen.

In view of a numerical approximation for atom wave functions it becomes clearer that more and more efficient computers have sped up this development significantly. The computer is probably the most important tool for the chemist in search of the last secrets of matter.

In the following we will show that even the C64 can be of service for quantum mechanical calculations.

In the judgement of π -electron systems of conjugated molecules, an approximation procedure originating from E. Hückel has proven to be useful, the HMO procedure, in order to describe intramolecular interactions sufficiently accurately. The concept of this procedure provides for two essential simplifications within the MO model:

1. The π -electrons are handled separately from the σ electrons; so-called $\pi\sigma$ separation. This separation is only valid for plain π -systems.
2. The overlapping-integral for non-adjacent centers remains unconsidered. In general, the following applies for a π -electron system with n centers

$$\psi = \sum_{\mu=1}^n c_{\mu} \cdot \phi_{\mu} \quad (1)$$

in which ϕ stands for the function of the atom orbitals, from which the multi-centered molecular orbitals will be obtained through linear combinations.

We get the approximation value E for the energy value required by the Schroedinger equation.

The coefficients c_{μ} are searched for, for which $E = E(c_{\mu})$ achieves a minimum. That is,

$$\frac{\partial E}{\partial c_v} = 0 \quad (v=1,2\dots N) \quad (2)$$

Calculating the derivatives yields the secular equation

$$\sum_{u=1}^n c_{\mu} (H_{\mu\phi} - E \delta_{\mu\phi}) = 0 \quad (3)$$

The coefficients c_{μ} are the only unknown, while the parenthesized expressions represent constant sizes.

For such an equation system, non-trivial solutions are only possible if the secular determinant is equal to zero:

$$\left| H_{\mu\phi} - E \delta_{\mu\phi} \right| = 0 \quad (4)$$

The n zero points of the corresponding equation of n th degree are the sought-after energy approximation values or the eigen values.

By the simplifications given above

$$\delta_{\mu\phi} = 0 \quad (\mu \neq \phi) \quad (5)$$

$$\delta_{\mu\phi} = 1 \quad (\mu = \phi)$$

and

$$H_{\mu\phi} = 0 \quad (\text{are not adjacent}) \quad (6)$$

Further, we set

$$\alpha_{\mu} = H_{\mu\mu} \quad (7a)$$

and

$$\beta_{\mu\phi} = H_{\mu\phi} \quad (7b)$$

With these suppositions the secular determinate is

$$\begin{vmatrix} (\alpha_1 - E) & \beta_{12} & \beta_{13} & \dots & \beta_{1N} \\ \beta_{21} & \alpha_2 - E & \beta_{23} & \dots & \beta_{2N} \\ \beta_{31} & \beta_{32} & \alpha_3 - E & \dots & \beta_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta_{N1} & \beta_{N2} & \beta_{N3} & \dots & \alpha_n - E \end{vmatrix} \quad (8)$$

$\beta_{\mu\phi}$ has a value other than zero if μ and ϕ denote neighboring atoms.

The values for ψ_i determined for (1) are orthogonal, so that other quantities, such as the charge densities, can be calculated from ψ_i alone.

The additional relationships should not be explained in any greater detail here; you can get that information from various other sources.

Let us now turn to the program operation.

The program in question is composed of several subroutines, each of which executes a portion of the necessary equations. The set of operations is controlled by the main program.

The program is set up such that a whole series of pi-electron systems can be calculated without anything additional. In addition, it is also possible to describe a desired π -System from the given isoconjugated model through a interference calculation. The isoconjugated reference system is specified at the start of the program by means of its incidence matrix. Within this matrix, adjacent atoms are designated with a 1 and all others with a 0.

Let us take a closer look at the subroutines used:

The input routine

First the number of the isoconjugated models is asked for. Then the name of the isoconjugated system and the number of centers contained in it and the doubly-occupied orbitals must be entered.

The incidence matrix is assigned columnwise in triangular form.

The matrix for benzene is:

```
0 1 0 0 0 1
0 1 0 0 0
0 1 0 0
0 1 0
0 1
0
```

Finally, the computer asks for the number of interfered systems to be calculated, the derivatives.

The matrix so generated on the monitor can be compared with the original as a check.

The routine for creating the Hueckel matrix.

The task of this routine is to duplicate the values in the upper triangular form in the lower half. The diagonal elements are also stored elsewhere.

The routine for calculating interference systems.

At the start of this routine the number of elements which must be exchanged is determined. The positions of the heteroatoms and the values to be substituted must also be entered.

The routine for diagonalization of real-symmetric matrices

One encounters so-called eigenvalue problems in many areas of engineering and natural sciences. For the case in which a linear equation system obeys the equation

$$A * X = \lambda X \quad (9.1)$$

it can also be written as

$$(A - \lambda E) X = 0 \quad (9.2)$$

where E represents the identity matrix. This system has non-trivial solutions only when

$$\begin{vmatrix} a_{11} - \lambda & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} - \lambda \end{vmatrix} = 0 \quad (9.3)$$

This leads to an equation of n th degree in λ , which is known as a characteristic equation. The solutions λ_i represent the eigenvalues. By inserting all λ_i in the solution system we get the column vectors x and thereby the eigenvectors.

Furthermore, if A is a symmetric matrix, all eigenvalues are real and the eigenvalues are perpendicular to the various λ_j ; they are orthogonal.

Since matrices of higher order and thereby polynomials of n th degree often exist in this relationship, there are already a large number of solution methods with which results can be approximated with the help of a diagonalization method. Since we will be working with real, symmetric matrices, we will be using Jacobi's method.

If there exists such a matrix A , then there is a matrix Q for which

$$Q^{-1} A Q = \text{diag}(\lambda) = D \quad (9.5)$$

It is the choice of Q which determines the actual difficulty, since one has no a prior knowledge of this matrix.

The Jacobi method consists of a set of elementary orthogonal transformations, which, like a planar revolution, in each pass eliminate a element off the diagonal. This element will again be changed in the next transformation, but it can be shown that the sum of the squares of the non-diagonal elements decreases and converges to zero.

The transformation matrix is an orthogonal matrix T which differs from the identity matrix in only four places, ii , ik , ki , and kk according to the sub-matrix

$$\begin{pmatrix} \cos \varphi & \sin \varphi \\ \sin \varphi & -\cos \varphi \end{pmatrix} \quad (9.6)$$

which results in a planar revolution about the angle.

The following apply for the new matrix

$$\begin{aligned} A_{ii}^x &= \cos^2 \varphi a_{ii} + \sin^2 \varphi a_{jj} + 2 \sin \varphi \cos \varphi a_{ij} \\ A_{jj}^x &= \sin^2 \varphi a_{ii} + \cos^2 \varphi a_{jj} - 2 \sin \varphi \cos \varphi a_{ij} \\ A_{ik}^x &= \cos \varphi a_{ik} + \sin \varphi a_{jk} \end{aligned} \quad (9.7)$$

with $k \neq i, j$

$$A_{jk}^x = \cos \varphi a_{jk} - \sin \varphi a_{ik}$$

From these relationships for the coordinate transformation one gets

$$\tan 2\varphi = \frac{2 a_{ij}}{a_{ii} - a_{jj}} \quad (9.8)$$

For the case where $a_{ii} < a_{jj}$ the angle can be determined with

$$\tan 2\varphi = \frac{-2 a_{ij}}{|a_{ii} - a_{jj}| + \sqrt{(a_{ii} - a_{jj})^2 + 4 a_{ij}^2}} \quad (9.9)$$

$$\tan 2\varphi = \frac{2 a_{ij}}{|a_{ii} - a_{jj}| + \sqrt{(a_{ii} - a_{jj})^2 + 4 a_{ij}^2}} \quad (9.10)$$

One can choose the angle $\{\{\}\}$ in the interval $(-\pi/4, \pi/4)$ such that in the newly formed matrix

$$A^X = T^T A T \quad (9.11)$$

the elements $a_{ij}=a_{ji}$ equal zero.

If there are no more elements lying off the diagonal which are under a preset limit, the diagonal elements can be viewed as an approximation for the eigenvalues.

In this connection, the eigenvectors can be calculated from the product matrix of all used T :

$$U = T_1 T_2 T_3 \dots T_n \quad (9.12)$$

As is also evident from our program, the calculation of this modal matrix is done level by level, starting with the identity matrix

$$U = I \quad (9.13)$$

and in each iteration proceeding on the provision that

$$U = U T \quad (9.14)$$

In general, the amount of work which Jacobi's method requires depends on the method which is used to find the elements to be eliminated.

One technique involves rotating the pivots which exceed a preset limit, until all of the non-diagonal elements are under this limit. Following this the limit is decremented and the whole procedure is repeated.

A less time-consuming procedure which will suffice for us is the following: The limit which we set at the start of the program serves as to determine when the approximation is reached as well as does the selection of the pivot element. A pivot is only rotated if the following is true:

$$|A_{ij}| < 10^{-8} \quad (9.15)$$

The sorting routine

Because the eigenvalues and their eigenvectors are not in any particular order, this subroutine sorts them according to size. Negative eigenvalues stand for binding and positive eigenvalues for non-binding conditions.

The bond pattern matrix routine

Charge pattern and bond pattern reflect the dynamic behavior of a simple derived pi system. The sum of the charge positions must return the number of pi-electrons. The created bond pattern matrix yields the general bond patterns and along the diagonals the charge patterns.

The routine for calculating free valences

The free valence is always viewed as a reaction criterium on a center. Certain reactions take place only on those centers whose free valences demonstrate the greatest values.

The output routine

The quantity of data dictates that the output go to the printer rather than the screen. The use of the print-using routine given elsewhere in this book makes it possible to format the results in a more readable manner. Modifications will have to be made to the program for systems with more than 9 centers.

The program as it stands here can be easily modified and enhanced. Program modules for calculation polarizeability and drawing the level pattern can be easily added.

The following is an example of using the program to make some calculations with benzene.

In response to the prompt for number and name of the isoconjugated model we enter:

1, BENZENE

We answer "Number of centers" and "Doubly-occupied orbitals" with

6 and 3, respectively

The matrix to be entered was given previously in the text.

"Derivatives": 2

1. For benzene itself:

variations: 0

2. For pyridin:

variations: 1

positions

row: 1

column: 1

value: -0.5

```
10 REM *****
11 REM *
12 REM * HUECKEL *
13 REM *
14 REM *CALCULATION*
15 REM *****
50 PRINTCHR$(147)
100 DIM A(30,30),B(30,30),OM(30,30),UM(30,30)
110 DIM AD(30),E(30),UT(30),U(30,30)
200 EP=1E-16
210 DEF FNR(X)=INT(1000*X+0.5)/1000
390 REM
395 REM *****
400 REM * START MAIN PROGRAM *
405 REM *****
410 REM
450 GOSUB 15000
460 PRINT
470 FOR Y=1 TO MM
480 : GOSUB 15300:GOSUB 16050:GOSUB 10000
490 PRINT:PRINTCHR$(13);TAB(10);"ALL INPUT O.K. (Y/N)";
500 : INPUT C$
510 :IF C$="N" THEN RUN
520 : PRINT
530 : FOR D=1 TO DD
535 : : PRINT
540 : : GOSUB 3000
550 : : GOSUB 6000
560 : : PV=0
570 : : PRINT:PRINT"CALCULATING..."
580 : : GOSUB 1000
590 : : GOSUB 4000
600 : : GOSUB 4600
610 : : GOSUB 5000
620 : : PRINTCHR$(147)
625 POKE211,5:POKE214,12:SYS 58732:PRINT"*****RESULTS*****"
630 : : GOSUB 11000:CLOSE1
640 : NEXT D
650 NEXT Y
700 END
800 : : POKE211,16:POKE214,23:SYS 58732:PRINT"( KEY )":FOR
I=1 TO 10:GETC$:NEXT
810 : : WAIT198,255:PRINTCHR$(147):RETURN
990 REM
```

```
995 REM *****
1000 REM * JACOBS ROUTINE *
1005 REM *****
1010 REM
1100 IF PV<>0 THEN 1250
1150 FOR J=1 TO N
1160 : FOR I=1 TO N
1180 : :U(I,J)=0
1190 NEXT I
1200 : U(J,J)=1
1210 NEXT J
1250 AM=0
1300 FOR I=2 TO N
1310 : PI=I-1
1320 : FOR J=1 TO PI
1330 : : AI=AD(I)
1340 : : AJ=AD(J)
1350 : : AP=A(I,J)
1360 : : AS=AP*AP
1370 :
1380 : : IF AS<=AM THEN 1400
1390 : : AM=AS
1400 : : IF AS<=EP THEN 1900
1410 : : DI=AI-AJ
1420 : : IF DI>=0 THEN 1450
1430 : : VZ=-2
1440 : : DI=(-1)*DI:GOTO 1460
1450 : : VZ=2
1460 : : VI=DI+SQR(DI*DI+4*AS)
1470 : : VN=VZ*AP/VI
1480 : : W=1/(SQR(1+VN*VN))
1490 : : S=W*VN
1500 : : FOR K=1 TO N
1510 : : : XJ=W*U(K,J)-S*U(K,I)
1520 : : : U(K,I)=S*U(K,J)+W*U(K,I)
1530 : : : U(K,J)=XJ
1540 : : : IF K=J THEN 1800
1550 : : : IF K>J THEN 1600
1560 :
1570 : : : XJ=W*A(J,K)-S*A(I,K)
1580 : : : A(I,K)=S*A(J,K)+W*A(I,K)
1590 : : : A(J,K)=XJ:GOTO 1800
1600 : : : IF K=I THEN 1800
1610 : : : IF K>I THEN 1680
1620 :
1630 : : : XJ=W*A(K,J)-S*A(I,K)
1640 : : : A(I,K)=S*A(K,J)+W*A(I,K)
1650 : : : A(K,J)=XJ:GOTO 1800
```

```

1670 :
1680 : : : XJ=W*A(K,J)-S*A(K,I)
1690 : : : A(K,I)=S*A(K,J)+W*A(K,I)
1700 : : : A(K,J)=XJ
1800 : : NEXT K
1810 :
1820 : : AD(I)=W*W*AI+S*S*AJ+2*S*W*AP
1830 : : AD(J)=W*W*AJ+S*S*AI-2*S*W*AP
1840 : : A(I,J)=0
1900 : NEXT J
1910 NEXT I
1950 IF AM>EP THEN 1250
2000 RETURN
2110 REM
2200 REM *****
3000 REM *   LOWER TRIANGULAR FORM *
3010 REM *****
3020 REM
3100 FOR J=1 TO N
3110 : FOR I=1 TO J
3150 : : A(J,I)=A(I,J)
3160 : NEXT I
3170 : Z(J)=1
3180 : AD(J)=A(J,J)
3200 NEXT J
3210 RETURN
3990 REM
3995 REM *****
4000 REM *   RIGHT SORTING   *
4005 REM *****
4010 REM
4099 FOR K=1 TO N:AD(K)=FNR(AD(K)):NEXT
4100 FOR K=1 TO N
4110 PA=AD(K)
4120 PJ=K
4150 FOR J=K TO N
4160 IF AD(J)>=PA THEN 4200
4170 PA=AD(J)
4180 PJ=J
4200 NEXT J
4250 AD(PJ)=AD(K)
4260 AD(K)=PA
4270 FOR I=1 TO N
4280 UT=U(I,PJ)
4290 U(I,PJ)=U(I,K)
4300 U(I,K)=UT
4310 NEXT I
4400 NEXT K

```

```
4500 RETURN
4590 REM
4595 REM *****
4600 REM * RIGHT HAND ORDER *
4605 REM *****
4610 REM
4620 :
4700 FOR BO=1 TO N
4710 : FOR BP=1 TO BO
4720 : : BS=0
4750 : : FOR J=1 TO M
4760 : : : BS=BS+U(BO,J)*U(BP,J)
4770 : : NEXT J
4780 : : B(BO,BP)=2*BS
4790 : NEXT BP
4800 NEXT BO
4850 :
4900 RETURN
4990 REM
4999 REM *****
5000 REM * DETERMINE FREE VALUES *
5005 REM *****
5010 REM
5100 FOR J=1 TO N
5110 :
5120 : NB=0
5130 : FOR I=1 TO N
5150 : : IF I=J THEN 5300
5160 : : IF I>J THEN 5200
5170 : : IF (A(I,J)+.1)>0 THEN 5300
5180 : : NB=NB+B(J,I):GOTO 5300
5200 : : IF (A(J,I)+0.1)>0 THEN 5300
5220 :
5250 : : NB=NB+B(I,J)
5300 : NEXT I
5310 : E(J)=1.732-NB
5320 NEXT J
5340 :
5350 RETURN
6000 REM
6005 REM *****
6010 REM * I'LL BE YOUR SUBSTITUTE *
6015 REM *****
6016 REM
6020 IF DD=0 THEN RETURN
6100 PRINTCHR$(147):PRINT:PRINT
6200 INPUT"(RVON)HOW MANY VARIABLES ARE NECESSARY";MO
6205 IF MO<1 THEN RETURN
```

```

6210 PRINT:PRINT
6220 PRINTCHR$(5);"POSITION OF THE HETETOATOM ?"
6230 PRINT:PRINTCHR$(154)
6280 :
6300 FOR K=1 TO MD
6320 : PRINTCHR$(13);TAB(14);K;"TH. ELEMENT"
6340 : INPUT"{RVON}ROW : ";I
6350 : PRINT
6360 : INPUT"{RVON}COLUMN: ";J
6370 : PRINT
6380 : INPUT"{RVON}VALUE TO BE SUBSTITUTED: ";X:PRINTCHR$(1
47):PRINT
6390 :
6400 : IF I<J THEN 6460
6410 : IF I>J THEN 6470
6420 :
6430 : AD(J)=X:GOTO 6480
6450 :
6460 : A(J,I)=X:GOTO 6480
6470 : A(I,J)=X
6480 NEXT
6500 RETURN
9990 REM
9995 REM *****
10000 REM *          OUTPUT          *
10005 REM *****
10010 REM
10050 REM *****
10100 REM *          SEMIMATRIX          *
10105 REM *****
10110 REM
10120 PRINTCHR$(147)
10130 PRINT"THIS IS YOUR MATRIX!"
10140 PRINT:PRINT
10160 FOR I=1 TO N
10170 : PRINT:PRINT:PRINTTAB(3*I+6);
10180 : FOR J=I TO N
10190 : : PRINTOM(I,J);
10200 : NEXT J
10210 NEXT I
10230 PRINT:PRINT
10240 RETURN
10500 REM
11000 REM *****
11001 REM * ** PRINTER OUTPUT ** *
11002 REM *****
11003 REM
11100 S1=3:S2=3

```

```

11110 OPEN1,4:PRINT#1,CHR$(14)"ISOCONJUGATED MODEL:";NA$;C
HR$(10)
11115 PRINT#1,"MOLECULE NR. ";Y;CHR$(15):PRINT#1:PRINT#1,"
QZ=";
11120 FOR I=1 TO N
11130 PX=AD(I):GOSUB 20000:PRINT#1,PX$;
11140 NEXT I:PRINT#1:PRINT#1
11200 PRINT#1,"HUECKEL-COEFFICIENTS "
11210 FOR I=1 TO N:PRINT#1," ";
11220 FOR J=1 TO N
11230 PX=U(I,J):GOSUB 20000:PRINT#1,PX$;
11240 NEXT J:PRINT#1:NEXT I
11250 PRINT#1:PRINT#1,"TOTAL--ELECTRON ENERGY: ";
11260 GE=0:FORJ=1 TO M:GE=GE+AD(J): NEXT:GE=2*GE:PX=GE:GOS
UB20000:PRINT#1,PX$
11270 PRINT#1:PRINT#1,"CHARGE DENSITIES"
11280 FORJ=1TON:FOR I=JTOJ:PX=B(I,J):GOSUB20000:PRINT#1,PX
$;:NEXTI:NEXTJ
11290 PRINT#1:PRINT#1
11300 PRINT#1,"FREE VALENCES:"
11310 FOR I=1 TO N:PX=E(I):GOSUB 20000:PRINT#1,PX$;:NEXT
11320 PRINT#1:PRINT#1
11330 PRINT#1,"GENERAL BOND ORDERS:"
11340 FORI=1 TO N:FORJ=1 TO I:PX=B(I,J):GOSUB 20000:PRINT#
1,PX$;:NEXT J:PRINT#1
11350 NEXT I
11360 PRINT#1:PRINT#1:IF D>1 THEN 11430
11365 PRINT#1,"HUECKEL MATRIX"
11370 FOR I=1 TO N
11380 PRINT#1:CMD 1:PRINTTAB(3*I+6);
11390 : FOR J=I TO N
11400 : : PRINT#1,OM(I,J);
11410 : NEXT J
11420 NEXT I
11430 PRINT#1:PRINT#1
11450 PRINT#1,"THAT WAS THE";D;"TH. MODIFICATION!"
11460 PRINT#1:PRINT#1,"*****"
*****
11900 PRINT#1:PRINT#1:PRINT#1:PRINT#1:CLOSE 1:RETURN
14990 REM
14995 REM *****
15000 REM * RIGHT INPUT *
15005 REM *****
15010 REM
15100 PRINTCHR$(5):POKE53280,0:POKE 53281,0

```

```

15140 PRINT:PRINT
15150 INPUT"HOW MANY ISOCONJUGATED MODELS ";MM
15160 PRINT"{CLR}";:FOR I=0 TO 39:PRINT " ";:NEXT
15200 RETURN
15300 PRINT"NAME OF THE";Y;"MATRIX STRUCTURE ":":PRINT:PRIN
T:INPUT NA$:PRINT
15500 PRINTTAB(10);Y;"{C/LF}. MOLECULE":PRINT
15520 INPUT"NUMBER OF CENTERS ";N
15550 PRINT
15600 INPUT"DOUBLY OCCUPIED ORBITALS ";M
15610 PRINT
15650 PRINTCHR$(147)
15700 PRINTTAB(9);"TIRANGULAR MATRIX INCOLUMNS"
15750 FOR J=1 TO N
15760 : FOR I=1 TO J
15770 : : PRINT"#";I;"{C/LF} ELEMENT OF THE #";J;"{C/LF} C
OLUMN";
15780 : : INPUT OM(I,J)
15790 : : A(I,J)=(-1)*OM(I,J)
15830 : NEXT I
15840 NEXT J
15850 RETURN
15870 :
16020 :
16050 INPUT"HOW MANY 'DERIVATIVES'";DD
16060 RETURN
20000 REM SUBROUTINE PRINT-USING-COMMANDPRINTER
20005 REM CREATES PX$ FROM PX WITH S1 THEN PLACES AFTER DE
CIMAL POINT
20010 REM AND S2 BEFORE DECIMAL POINT
20015 IF ABS(PX)<1E-8 THEN PX=0
20020 PX$=STR$(PX):PP=LEN(PX$):DP=PP+1:NU$=".000000000"
20025 IF ABS(PX)<1 AND PX<>0 THEN PX$="0"+RIGHT$(PX$,PP-1)
20027 IF -1<PX AND PX<0 THEN PX$="-"+PX$:PP=PP+1
20030 FOR II=1 TO PP:IF MID$(PX$,II,1)=". "THEN DP=II:NU$="0
0000000":II=PP
20040 NEXT:PX$=PX$+NU$:PX$=LEFT$(PX$,DP+S1)
20050 FOR II=1 TO S2-DP+2:PX$=" "+PX$:NEXT II
20060 RETURN

READY.

```


ISOCONJUGATED MODEL: BENZENE

MOLECULE NR. 1

QZ= -2.000 -1.000 -1.000 1.000 1.000 2.000

HUECKEL-COEFFICIENTS

0.408	-0.547	0.183	-0.561	-0.135	-0.408
0.408	-0.432	-0.382	0.397	-0.418	0.408
0.408	0.114	-0.565	0.163	0.553	-0.408
0.408	0.547	-0.183	-0.561	-0.135	0.408
0.408	0.432	0.382	0.397	-0.418	-0.408
0.408	-0.114	0.565	0.163	0.553	0.408

TOTAL- π -ELECTRON ENERGY: -8.000

CHARGE DENSITIES

0.999	0.999	0.999	0.999	0.999	0.999
-------	-------	-------	-------	-------	-------

FREE VALENCES:

0.398	0.398	0.398	0.398	0.398	0.398
-------	-------	-------	-------	-------	-------

GENERAL BOND ORDERS:

0.999					
0.666	0.999				
0.000	0.666	0.999			
-0.333	0.000	0.666	0.999		
0.000	-0.333	0.000	0.666	0.999	
0.666	0.000	-0.333	0.000	0.666	0.999

HUECKEL MATRIX

0	1	0	0	0	1
	0	1	0	0	0
		0	1	0	0
			0	1	0
				0	1
					0

THAT WAS THE 1 TH. MODIFICATION!

Program explanation

Line

100-110	dimension variables
200	stop critierium
210	rounding function
450	call input
540-610	call calculation routines
630	call printer output
1000-2000	Jacobi method
1100-1210	construct identity matrix
1300-1400	select the pivot element to rotate
1410-1490	calculate $\cos \{\{\}\}$, $\sin \{\{\}\}$
1500-1800	coordinate transformation
1510-1530	construct modal matrix
1540-1700	calculate the new matrix
1820-1830	
	The eigenvalues or diagonal elements are stored in this one-dimensional array.
1950	stop when value goes under limit
3000-3210	Hueckel matrix
3150	reflection of the array elements on the main diagonal.
3180	place diagonal elements in $A(I,J)$
4000-4500	
	Sorting the eigenvalues $AD(J)$ and the eigenvectors $U(I,K)$ according to size.
4600-4900	General bond pattern
4700-4800	Calculation of the bond pattern $P\{\{\mu v\}\}$ by

$$p_{\mu\nu} = \sum_{j=1}^n b_j c_{j\mu} c_{j\nu} \quad (10)$$

The charge patterns $q\{\mu\}=p\{\mu\mu\}$ lie on the main diagonals of the matrix.

5000-5350

Free valences

The free valence is calculated by means of

$$F_{\mu} = \sqrt{3} - \sum_{\nu=1}^n p_{\mu\nu} \quad (11)$$

6000-6500

Calculating interrupted systems

If a carbon atom is replaced by a heteratom within a pi-electron system, the coulomb integral $\{\alpha\}$; if it is an atom which is involved in the bond, a change in the resonance integral is also involved. This subroutine takes care of this condition and makes sure that it is taken into account in further calculations.

10000-11900

Output routines

10000-10240

Display the entered triangular matrix on the screen.

11000-11900

Formatted output of the collected results on the printer, with the help of the PRINT USING in 20000-20060.

15000-16100

Input routines

Chapter 7: Applications in Physics

7.1 Measurement of three overlapping times

You can easily measure times or time differences with the built-in software clock TI\$ or TI. If you must measure several times which may overlap each other, however, it becomes somewhat more complicated to make the measurements without losing accuracy.

If you the assign various times to various keys, you must be certain that reading the keys does not take so long that the clock has run a few tenths of a second past the actual time before it is stopped.

The program "Three Times" in this section illustrates one solution of this problem. The keys 1, 2, and 3 are used to start the timings and the keys 8, 9, and 0 to stop them. It is automatically ensured that a time will not be stopped or started twice, or that an unstarted time can be stopped. Because the algorithm is somewhat complicated, the corresponding structogram is shown in figure 7.1a.

In addition, the PRINT USING subroutine is used in order to output the times to a given number of places of accuracy. Please note that the time can only be measured to 1/60 ths of a second accuracy and that even this accuracy is itself a bit utopian because the software clock TI is not quartz-stabilized and also does not depend on the frequency of the household current. If you require a higher accuracy over a long period of time, you must use the clock in CIA 1 or CIA 2. These are coupled to the alternating current of

the power line and are therefore extremely accurate. The only measure in tenths of a second, however. The use of the CIA clocks is explained below.

If you want to measure the times with external contacts, which is indispensable for physics applications, the joystick ports can be used. You need only change the numbers in lines 15010 and 15020 in order to read the joysticks. Because of the superimposition of the joystick and keyboard, you may not use the keyboard during the measurements, otherwise you could start or stop a time or inhibit the registering of a closed joystick contact.

It is naturally impossible to start two times simultaneously because only one contact can be read and worked with at a time. On the other hand, if you know beforehand that times 1 and 2 start together, for instance, then in line 26010 you can set not only $S1=T$, but also $S2=T$ and use the contact for starting time 1 as the starter for both. In order to be able to stop the second time (which was not started by itself), you must also insert $X2=E2$ in line 16010.

You can do that same sort of thing for common stop times. The structogram will help you see how to do this.

The determining factor for the accuracy of this program is the registration of the time immediately after the keypress. The temporarily-stored time is assigned. The assignment algorithm does not affect the time measurement. Once a time is registered, the registration of another time is inhibited.

Do not disable the RUN/STOP key in the program with POKE 788,52! This causes the software clock to stop and all of the times you try to measure will be zero.

You will no doubt think of many applications for this program.

Using the CIA clocks

In contrast to TI\$ or TI, you must enable these clocks before they will run. Once they start running, only loss of power will stop them again. A soft reset through SYS 64738 stops both clocks for about 1 1/2 seconds, but they continue running. Each SYS 64738 causes a loss of about 1 1/2 seconds.

You can set and enable the clocks with the program "Time input." For our application of measuring time intervals, the actual time we assign to clock is irrelevant since we need only the differences. The program "Real time" reads the clock time. It outputs the time in 24-hour format. More information about the clocks in CIA 1 and CIA 2 can be found in the book The Anatomy of the Commodore 64, or The Advanced Machine Language Book for the Commodore 64.

We made an accuracy comparison between TI\$ and the CIA timers. After 9 1/2 hours, there was no measureable deviation in the CIA clock, but the TI\$ clock had a deviation of about 3 seconds.

```

1 REM *****
2 REM *
3 REM *   MEASURING THREE TIMES   *
4 REM *
5 REM *****
6 :
10 A1=49:A2=50:A3=51
20 E1=56:E2=57:E3=48
30 SA=3:SB=15
1000 X1=A1:X2=A2:X3=A3
1010 REM BEGIN
1020 GETX$:IFX$="" THEN 1020
1030 T=TI: REM NOTE TIME
1040 X=ASC(X$):GOSUB 2000
1050 IF X1<>0 OR X2<>0 OR X3<>0 THEN 1020
1100 REM OUTPUT
1110 PRINT"FIRST  TIME";:PX=(T1-S1)/60:GOSUB10000:PRINT
1120 PRINT"SECOND TIME";:PX=(T2-S2)/60:GOSUB10000:PRINT
1130 PRINT"THIRD  TIME";:PX=(T3-S3)/60:GOSUB10000:PRINT
1140 END
2000 REM *   BRANCH *
2010 IF X=X1 THEN GOSUB 3000:RETURN
2020 IF X=X2 THEN GOSUB 4000:RETURN
2030 IF X=X3 THEN GOSUB 5000:RETURN
2040 RETURN
3000 REM ***   ASSIGN TIME   ***
3010 IF X1=A1 THEN S1=T:X1=E1:PRINT"1 START":RETURN
3020 IF X1=E1 THEN T1=T:X1=0 :PRINT"1 STOP ":RETURN
3030 RETURN
4000 REM ***   ASSIGN TIME   ***
4010 IF X2=A2 THEN S2=T:X2=E2:PRINT"2 START":RETURN
4020 IF X2=E2 THEN T2=T:X2=0 :PRINT"2 STOP ":RETURN
4030 RETURN
5000 REM ***   ASSIGN TIME   ***
5010 IF X3=A3 THEN S3=T:X3=E3:PRINT"3 START":RETURN
5020 IF X3=E3 THEN T3=T:X3=0 :PRINT"3 STOP ":RETURN
5030 RETURN
10000 REM * PRINT USING SUBROUTINE *
10010 REM PRINT PX WITH SA POSITIONS AFTER DECIMAL AT THE
SA-TH PLACE
10020 PX$=STR$(PX):PF=LEN(PX$):DP=PF+1:NU$=".000000000"
10030 FOR II=1 TO PF:IF MID$(PX$,II,1)=". " THEN DP=II:NU$=
"000000000"
10040 NEXT:PX$=PX$+NU$:DL$=LEFT$(PX$,DP):DR$=MID$(PX$,DP+1
,SA)

10045 IF SB-DP<1 THEN SB=DP+1
10050 PRINTTAB(SB-DP);DL$;DR$
10060 RETURN

```

READY.

```

1 REM *****
2 REM * TIME INPUT *
3 REM *****
4 :
10 C=56328:REM BASIC ADDRESS CIA 1
20 REM C=56584 FOR CIA 2
30 POKE C+7,PEEK(C+7) AND 127
35 POKE C+6,PEEK(C+6) OR 128
40 INPUT "ENTER TIME IN FORMAT HHMMSS";A$
50 IF LEN(A$)<>6 THEN 40
60 H=VAL(LEFT$(A$,2))
70 M=VAL(MID$(A$,3,2))
80 S=VAL(RIGHT$(A$,2))
90 IF H>23 THEN 40
100 IF H>11 THEN H=H+60
110 POKE C+3,16*INT(H/10)+H-INT(H/10)*10
120 IF M>59 THEN 40
130 POKE C+2,16*INT(M/10)+M-INT(M/10)*10
140 IF S>59 THEN 40
150 POKE C+1,16*INT(S/10)+S-INT(S/10)*10
160 POKE C,0

```

READY.

```

1 REM *****
2 REM * REAL TIME*
3 REM *****
4 :
10 C=56328:REM BASIC ADDR. OF THE CLOCK IN CIA 1
20 PRINT"{CLR}":REM C=56584 CLOCK IN CIA 2
30 H=PEEK(C+3):M=PEEK(C+2):S=PEEK(C+1):T=PEEK(C)
40 FL=1
50 IF H>23 THEN H=H-128:FL=0
60 H=INT(H/16)*10+H-INT(H/16)*16:ON FL GOTO 80
65 IF H=12 THEN 85
70 H=H+12
80 IF H=12 THEN H=0
85 M=INT(M/16)*10+M-INT(M/16)*16
90 S=INT(S/16)*10+S-INT(S/16)*16
100 T$=STR$(T)
110 H$=STR$(H):IF LEN(H$)=2 THEN H$=" 0"+RIGHT$(H$,1)
120 M$=STR$(M):IF LEN(M$)=2 THEN M$=" 0"+RIGHT$(M$,1)
130 S$=STR$(S):IF LEN(S$)=2 THEN S$=" 0"+RIGHT$(S$,1)
140 PRINT"(HOME)";
150 PRINT RIGHT$(H$,2):"RIGHT$(M$,2)": "RIGHT$(S$,2)":0";
160 PRINT RIGHT$(T$,1)
170 GOTO 30

```

READY.

7.2 Determining an isolation error in an underground cable

We will write a program to determine the transmission resistance in an isolation error occurring in an underground cable. We will also determine where the isolation error occurs in the cable.

The following information is necessary:

Length of the cable	L (m)
Cross-section	A (mm ²)
Conductivity	S (1/Ohm*meter)
Resistance to earth at both ends	RA (Ohm) RB (Ohm)

The program yields as a result the location of the isolation error and its transmission resistance. The following mathematical relationships are used:

$$(1) \quad R_A = R_1 + R_2$$

$$(2) \quad R_B = R_2 + R_3$$

$$(3) \quad R_G = R_1 + R_3 = \frac{L}{S \cdot A}$$

$$(4) \quad R_1 = \frac{R_G + R_A - R_B}{2} = \frac{(R_1 + R_3) + (R_1 + R_2) - (R_2 + R_3)}{2}$$

$$(5) \quad R_2 = R_A - R_1$$

$$(6) \quad X = \frac{R_1}{R_G} \cdot L$$

After you have started the program, graphics which will clarify the procedure will appear on the screen. After pressing a key, you will be asked for the values L and A. The program has the values for pure and wire aluminum, pure and wire copper, konstantan, silver, and tungsten so that you do not have to keep looking up the resistance. If you use one of these substances as the conductor, simply press the corresponding function key. Otherwise press F8 and enter the conductivity.

The last value which you must enter is the measured value of the conductor-earth resistance. The resistance between two arbitrary ground points is taken to be zero and is ignored.

The location of the isolation error is given in m, measured from point A. The program can print all relevant data and the graphics on the printer on command.

In clarification of the program: The graphics subroutine (at line 35000) is called in line 1000. The CHR\$(15) and CHR\$(8) have no effect on the screen output; they are required only for the printer output. The graphics clarify the view of the problem, illustrating the three resistances, the line resistance from A and from B to the isolation error and the transmission resistance at the isolation error.

At the end of the line 1000, the input subroutine (15000-15340) is called. This is the most complicated part of the program because the input should be especially easy to do. After the input of the length and cross-section of the underground cable, the screen is erased and the values

are printed in the first two lines. Underneath a dividing line appear the 7 choices of conductor material, the conductivities of which are stored in the DATA line 25010. One of these conductors or some other material can be selected with the help of a function key. The screen is then erased from line 3 on by a subroutine (at line 16000) and the conductivity requested, if necessary. This is written in the fourth line of the screen, after which the screen is again erased at line 3 in order to remove the input.

Then the measured conductor-end-earth resistances at the points A and B are requested. With this the input routine ends and the calculation at line 2000 can begin.

The location and transmission resistance of the isolation error are calculated as per the above formulas. The message "Data error" appears if nonsensical values have been entered and the program begins again after a key has been pressed. The loop to wait for the keypress is implemented with a WAIT command.

All the screen lines below the fourth are erased before the output by the subroutine call in line 10010. The cursor positioning in lines 10020 and 10030 is done with POKes because the TAB command no longer functions in a line where the cursor position has been changed with a cursor control key in a printed line. More about this can be found in the section on screen input and output.

The values for distance and transmission resistance are formatted with the help of the PRINT USING subroutine. Note that the subroutine presented here is somewhat different than usual because it is also used for formatting the

printer output. Instead of the print location on the screen, we give here the number of places before the decimal in order to line the data up nicely. The TAB command does not work on the VIC-1525 or GP 100 VIC printer.

At 11000 the question about printer output is asked. The answer to this question controls the output in the subroutine at line 30000. All of the data is formatted with PRINT USING. Graphics can also be printed if desired. In order to do this the output is directed to the printer (CMD 1) and the graphics subroutine is called. Now the CHR\$(15) and CHR\$(8) are required. In the graphics mode, enables with CHR\$(8), the line feeds are smaller so that the lines are printed closer together. It is only in this manner that uninterrupted vertical lines are possible. The normal print mode is switched back on with CHR\$(15) because the the graphics symbols are printed in this mode. The CHR\$(8) is only for single-pin graphics. Because of the different dot matrices of the screen (8x8) and the printer (5x7), the vertical lines appear transposed one point. This unfortunately not be avoided if one uses the same commands from both printer and screen output.

The "output only to printer mode" is switched off by the PRINT#1 command before the CLOSE 1.

Once again in the output portion, the user is asked in line 14000 if more calculations are desired.

```

10 REM *****
20 REM *
30 REM * ISOLATION ERROR LOCATION *
40 REM * & RESISTANCE DETERMINATION*
50 REM *
60 REM *****
70 REM
100 POKE53280,0:POKE53281,0:PRINT "{GRY2}"
1000 GOSUB 35000:WAIT 198,255:POKE 198,0:GOSUB 15000
2000 REM
2001 REM *****
2002 REM * CALCULATION *
2003 REM *****
2004 REM
2010 RG=L/(S*A/1E6):REM * TOTAL LINE RESISTANCE *
2020 R1=(RG+RA-RB)/2
2030 R2=RA-R1:REM * CROSS RESISTANCE *
2040 IF R1>RG OR R3>RG THEN PRINT "{C/DN}{C/DN}{C/RT}{C/RT}"
DATA ERROR":WAIT 198,255:POKE198,0:RUN
2050 X=(R1/RG)*L:REM * DISTANCE TO A *
2060 IF R1<0 THEN PRINT "{C/DN}{C/DN}{C/RT}{C/RT}DATA ERRO
R":WAIT 198,255:POKE 198,0:RUN
10000 REM
10001 REM *****
10002 REM * OUTPUT *
10003 REM *****
10004 REM
10010 Z=4:GOSUB 16000: REM * ERASE *
10020 POKE211,0:POKE214,3:SYS 58732:PRINT "{GRY2}RA";
10030 POKE 211,18:SYS 58732:PRINT RA;:POKE 211,28:SYS 5873
2:PRINT "OHMS"
10040 PRINT"RB";TAB(18);RB;TAB(28);"OHMS"
10050 PRINT"{C/DN}{C/DN}{WHT}CROSS RESISTANCE/OHMS=";:PX=R
2:S1=1:S2=5:GOSUB 20000:PRINT PX$
10060 PRINT"{C/DN}DISTANCE FROM A (M) =";:PX=X:S1=1:S2
=5:GOSUB 20000:PRINT PX$
11000 REM
11001 REM *****
11002 REM * PRINTER ? *
11003 REM *****
11004 REM
11010 POKE 211,5:POKE 214,20:SYS 58732:PRINT "{GRY2}OUTPUT
TO PRINTER {WHT}(Y/N)"
11020 GET X$
11030 IF X$="Y" THEN GOSUB 30000:GOTO 11050

```

```

11040 IF X$<>"N" THEN 11020
11050 REM
14000 REM
14001 REM *****
14002 REM * MORE CALCULATIONS ? *
14003 REM *****
14005 REM
14010 PRINT "{CLR}":POKE 211,5:POKE 214,20:SYS 58732:PRINT"
{GRY2}MORE CALCULATIONS {WHT}(Y/N)
14020 GET X$
14030 IF X$="Y" THEN RUN
14040 IF X$="N" THEN PRINT "{CLR}":END
14050 GOTO 14020
15000 REM
15001 REM *****
15002 REM * INPUT *
15003 REM *****
15004 REM
15010 PRINT "{CLR}{GRY2}{C/DN}{C/DN}{C/RT}DETERMINING RESIS
TANCE AND PLACE OF AN"
15020 PRINT "{C/RT}{C/RT}{C/RT}ISOLATION ERROR IN AN EARTH
CABLE"
15030 PRINT
15040 PRINT "{C/DN}{C/DN}{C/DN}LENGTH OF THE CABLE IN METER
S";:INPUT L
15050 PRINT "{C/DN}{GRY2}LINE CROSS SECTION MM↑2{WHT}";:
INPUT A
15060 PRINT "{CLR}{GRY2}LENGTH";TAB(18);L;TAB(28);"M"
15070 PRINT "CROSS SECTION";TAB(18);A;TAB(28);"MM↑2
15080 PRINT"
"
15090 PRINT "{C/DN}CONDUCTOR MATERIAL FUNCTIONS-":
PRINTTAB(30);"KEY"
15100 PRINT "{C/DN}{GRY2}WIRE -CU";TAB(32);"{WHT}F1"
15110 PRINT "{C/DN}{GRY2}PURE CU";TAB(32);"{WHT}F2"
15120 PRINT "{C/DN}{GRY2}WIRE -AL";TAB(32);"{WHT}F3"
15130 PRINT "{C/DN}{GRY2}PURE AL";TAB(32);"{WHT}F4"
15140 PRINT "{C/DN}{GRY2}KONSTANTAN";TAB(32);"{WHT}F5"
15150 PRINT "{C/DN}{GRY2}SILVER";TAB(32);"{WHT}F6"
15160 PRINT "{C/DN}{GRY2}TUNGSTEN";TAB(32);"{WHT}F7"
15170 PRINT "{C/DN}{GRY2}OTHER MATERIAL";TAB(32);"{WHT}F8"
"
15180 GET X$:IF X$=""THEN 15180
15190 X=ASC(X$):IF X<133 OR X>140 THEN 15180
15200 IF X<140THEN FOR I=133 TO X:READ S:NEXT I:S=S*1E7
15210 Z=3:GOSUB 16000:REM * CLEAR *
15220 IF X=140 THEN PRINT "{HOME}{C/DN}":PRINT "{GRY2}CONDUCT
IVITY{WHT}";:INPUT S:Z=3:GOSUB 16000

```

```

15290 Z=3:GOSUB 16000:REM CLEAR
15300 PRINT"(HOME){C/DN}{C/DN}{GRY2}CONDUCTIVITY";TAB(18);
S;TAB(28);"1/OHMMETER"
15310 PRINT"{C/DN}{C/DN}RESISTANCE TO EARTH IN OHMS"
15320 PRINT"{GRY2}AT END A:{WHT}";:INPUT RA
15330 PRINT"{GRY2}AT END B:{WHT}";:INPUT RB
15340 RETURN
16000 REM
16001 REM *****
16002 REM *   CLEAR SCREEN AT LINE   *
16003 REM *                               *
16004 REM *****
16005 REM
16010 PRINT"(HOME)":FOR I=1 TO Z-2:PRINT:NEXT:FOR I=Z TO 2
3
16020 PRINT"                                ";:NEX
T I
16030 RETURN
20000 REM *****
20001 REM *SUBROUTINE                               *           *PRINT-U
SING-COMMAND-PRINTER *
20005 REM *CREATE PX$ FROM PX IF *           *S1 PLACES AFT
ER DX *
20010 REM *AND S2 BEFORE *
20011 REM *****
20015 REM
20020 PX=PX+.5:PX$=STR$(PX):PP=LEN(PX$):DP=PP+1:NU$=".0000
0000"
20030 FOR II=1 TO PP:IF MID$(PX$,II,1)=". "THEN DP=II-2:NU$
="00000000":II=PP
20040 NEXT:PX$=PX$+NU$:PX$=LEFT$(PX$,DP+S1+2)
20050 FOR II=1 TO S2-DP:PX$=" "+PX$:NEXT II
20060 RETURN
25000 REM
25001 REM *****
25002 REM *   CONDUCTIVITY DATA   *
25003 REM *****
25005 REM
25010 DATA 5.618,5.814,3.484,3.704,.2,6.25,1,818
30000 REM
30001 REM *****
30002 REM *   PRINTER OUTPUT   *
30003 REM *****
30004 REM
30010 PRINT"{CLR}{C/DN}{C/DN}{GRY2}NUMBER OF CABLES 1{C/LF
}{C/LF}{C/LF}";:INPUT DR
30020 FOR I=1 TO DR
30030 OPEN 1,4

```

```

30040 PRINT#1,"DETERMINATION OF RESISTANCE AND PLACE OF"
30050 PRINT#1,"ISOLATION ERROR IN AN EARTH CABLE  ":S2=12
30060 PRINT#1,"
-----"
30070 PRINT#1,"*   INPUTS   *"
30080 PX=L:S1=0:GOSUB 20000:PRINT#1,"LENGHT OF CABLE  (M)
.....";PX$
30090 PX=A:S1=1:GOSUB 20000:PRINT#1,"CROSS SECTION(QMM)...
.....";PX$
30100 PX=S:S1=0:GOSUB 20000:PRINT#1,"CONDUCTIVITY  (1/OHM
METER)..."PX$
30110 PX=RA:S1=1:GOSUB 20000:PRINT#1,"RESISTANCE AT END  A
(OHM)..."PX$
30120 PX=RB:S1=1:GOSUB 20000:PRINT#1,"RESISTANCE AT END  B
(OHM)..."PX$
30130 PRINT#1:PRINT#1,"*   RESULTS   *"
30140 PX=R2:S1=1:GOSUB 20000:PRINT#1,"CROSS RESISTANCE
(OHM)..."PX$
30150 PX=X:S1=0:GOSUB 20000:PRINT#1,"DISTANCE ERROR  -  A
(M)..."PX$
30160 FOR I=1 TO 3:PRINT#1:NEXT I
30170 PRINT "{C/DN}{GRY2}PRINT GRAPHICS  {WHT}(Y/N)"
30180 GET X$:IF X$<>"Y" THEN 30180
30190 PRINT#1,CHR$(8):CMD1:GOSUB 35000:PRINT#1,CHR$(15):CL
OSE1:RETURN
35000 REM
35001 REM *****
35002 REM *   CLARIFACTION GRAPHICS   *
35003 REM *****
35004 REM
35010 PRINT "{CLR}";
35020 PRINTCHR$(15);"          "          "          ";CHR$(
8)
35030 PRINTCHR$(15);"          +-----+          +-----+";
CHR$(8)
35040 PRINTCHR$(15);"          |          |          |          ";C
HR$(8)
35050 PRINTCHR$(15);"          |          R1          |          R3          |";CHR$(
8)
35060 PRINTCHR$(15);"          |          |          |          |";CHR$(
8)
35070 PRINTCHR$(15);"          |          +-----+          |";CHR$(
8)
35080 PRINTCHR$(15);"          |          |          |          |";CHR$(
8)
35090 PRINTCHR$(15);"          RA          R2          |          |          RB";CHR$(
8)
35100 PRINTCHR$(15);"          |          |          |          |";CHR$(

```



```

(8)
35110 PRINTCHR$(15); " | MEASURED | MEASURED | "; CHR$(
(8)
35120 PRINTCHR$(15); " | | | "; CHR$(
(8)
35130 PRINTCHR$(15); " | | | "; CHR$(
(8)
35140 PRINTCHR$(15); " .....+.....
"; CHR$(8)
35150 PRINTCHR$(15); " | "; CHR$(8)
35160 PRINTCHR$(15); " .....+....."; CHR$(8)
35170 PRINTCHR$(15); " ERDE ....."; CHR$(8)
35180 PRINTCHR$(15); " ....."
35190 PRINT
35200 PRINT" R1 = LINE RESISTANCE UP TO ...."
35210 PRINT" R2 = CROSS RESISTANCE OF THE ...."
35220 PRINT" R3 = LINE RESISTANCE AT THE ...."
35230 PRINT" ISOLATION ERROR "
35240 RETURN

```

READY.

7.3 Optical geometry

The task of optic geometry is to examine the behavior of a single ray of light. The light source is considered to be a point source.

Such an elementary outlook simplifies an exact pursuit of the light ray's path.

1. Divergent rays -- rays which stream out radially from a common point.
2. Convergent rays -- rays which run to a common point.
3. Diffuse rays -- have neither a common output point nor a common point of intersection.

If when parallel rays are reflected by a smooth surface, the reflected rays are also parallel to each other, one calls this surface a mirror.

It is different for a spherical mirror; with this type of concave mirror, all of the rays parallel to the axis are collected to a focal point. If we also want to work with rays near the axis, we get the following formula for the focal length f :

$$f = r/2$$

In addition we get

$$P/O = -p/o$$

for the figure scale, or in words:

"The size of the picture and object is related to their distances from the mirror."

For a formulation of the mathematical relationships, it is important to pay attention to the correct choice of sign because this obeys a special agreement. The distances on the light side are positive and the distances lying on the other side of the mirror have a negative sign.

For the representation equation of the mirror one gets

$$1-f = 1-o+1-p$$

where f =focal length, p =object length, and b =picture length.

Analog relationships apply for spherical lenses. The focal length can be determined from the radii of the lenses and their refractive index.

The indispensable equation for the optician is

$$1/f = (n-1)(1/r_1-1/r_2)$$

with $D=1-f$. D is the refracting power of the lense.

Here too the applicable sign rules must be noted:

The object distance o is positive on the side of the light incidence; the picture distance has a negative value on this side. In addition, the curve radius has a negative sign on the side to the light, in the case that it is a

convave surface. With this short refresher, you can now proceed to the program which calculated the representation sizes of mirrors or thin lenses.

Knowledge of the focal length is not absolutely necessary, as long as the radii and refractive index are known.

```
10 REM *****
11 REM *
12 REM * OPTICS
13 REM *
14 REM *
15 REM *****
20 :
50 PRINTCHR$(147):POKE 53280,0:POKE53281,0
60 :
200 DEF FNR(X)=INT(100*X+0.5)/100
300 :
400 PRINTTAB(15);"O P T I C S":PRINT:PRINT
500 :
510 PRINT:PRINT"{RVON}M{RVOF}ERROR OR {RVON}L{RVOF}ENS";
520 INPUT A$
530 IF A$="L" THEN GOSUB 15000:GOTO 560
540 PRINT
550 GOSUB 15200
560 GOSUB 1100
600 GOTO 20000
750 :
800 POKE211,5:POKE214,20:SYS 58732:PRINT"N O T E S I G N !!
!!";"{HOME}"
810 PRINT:RETURN
850 :
990 REM
1000 REM CALCULATIONS
1010 REM
1100 PRINTCHR$(147):PRINT:PRINT
1120 PRINTTAB(5);"OBJECT WIDTH: ";
1130 INPUT G
1140 PRINT
1200 PRINT"HEIGHT OF OBJECTS: ";
1210 INPUT GH
1300 IF G=F THEN 1100
1400 B=1/(1/F-1/G)
1450 AM=-B/G
1500 BH=GH*AM
1540 GOSUB 10000
1550 PRINT:PRINTTAB(12)"NEW FIGURE? (Y/N)"
1560 GETC$:IF C$="" THEN 1560
1570 IF C$="N" THEN RETURN
1600 GOTO 1100
9970 +0
9980 REM OUTPUT
```

```
9990 REM
10000 PRINTCHR$(147):PRINT:PRINTTAB(9);"R E S U L T S   "
10010 PRINT:PRINT
10020 PRINT"*****PICTURE WIDTH   : ";FNR(B)
10030 PRINT
10040 PRINT"*****PICTURE HEIGHT  : ";FNR(BH)
10050 PRINT:PRINT
10060 PRINT"*****FOCAL LENGTH"; FNR(F)
10100 RETURN
14990 REM
15000 REM INPUT
15010 REM
15090 PRINTCHR$(147):PRINT
15100 GOSUB 800:PRINT
15110 INPUT"FOCAL LENGTH: ";F
15120 IF F<>0 THEN RETURN
15130 INPUT"{C/UP}REFRACTIVE INDEX: ";N
15135 PRINT"{C/UP}";:FOR I=0 TO 39:PRINT" ";:NEXT
15140 PRINT"{C/UP}CURVATURE RADIUS 1: ";:INPUT KR
15150 PRINT
15160 PRINT"CURVATURE RADIUS 2: ";:INPUT RK
15165 IF N=1 OR KR=0 OR RK=0 THEN 15000
15170  $F=1/((N-1)*(1/KR-1/RK))$ 
15180 RETURN
15190 :
15200 PRINTCHR$(147):PRINT
15210 GOSUB 800:PRINT
15220 PRINTTAB(6);"FOCAL LENGTH: ";:INPUT F
15230 IF F<>0 THEN RETURN
15240 PRINT
15250 PRINT"CURVATURE RADIUS: ";:INPUT RS
15260  $F=RS/2$ 
15270 RETURN
20000 END
```

READY.

7.4 Planetary orbit calculation

The following little program calculates the positions of planetary orbits using the Kepler Laws. Planets orbit the sun in elliptical orbits. The third of Kepler's laws can be written as follows:

$$(A) \quad M = \left(\frac{M}{A} \right)^{1/2} * (t-T) = E - e \sin E$$

The perihelion distortion is ignored and the mass of the planet is viewed as a point.

The following data are required as input:

1. The eccentricity of the planetary orbit
2. The period of the orbit
3. The value of the greater half-axis of the orbit
4. The time

Current values can be obtained from an astronomical calendar.

The program can then calculate the situation of the planet regarding its perihelion.

```
10 REM *****
20 REM *
30 REM * CALCULATIONS OF *
40 REM * POSITIONS OF *
50 REM * THE PLANETS *
60 REM * USING KEPLER'S *
70 REM * LAWS *
90 REM *****
99 REM
100 POKE 53280,0:POKE 53281,0
110 REM
200 REM VARIABLE LIST
210 REM -----
220 REM
230 REM MA = AVERAGE ANOMALY
240 REM EX = ECCENTRICITY
250 REM RM = DISTANCE BETWEEN
255 REM THE TWO MASSES
260 REM (SUN AND PLANET)
270 REM PU = TIME FOR AN ORBIT
280 REM GH = LARGE HALF-AXIS OF THE ORBIT
290 REM T = TIME
300 REM EA = ECCENTRIC ANOMALY
310 REM
320 REM -----
333 REM
1000 REM *****
1010 REM * MAIN PROGRAM *
1020 REM *****
1025 REM
1030 GOSUB 10000: REM INPUT
1040 MA=2*PI*T/PU
1050 GOSUB 5000 : REM ECCENTRIC ANOMALY
1060 RM=GH*(1-EX*COS(EA))
1070 CA=(COS(EA)-EX)/(1-EX*COS(EA))
1080 AT=ATN(SQR((1-CA^2)/CA^2))
1090 AT=AT*180/3.1415927
1100 WE=AT
```



```
1110 IF CA<0 THEN WE=180-AT
1120 IF CA>0 AND SIN(EA)<0 THEN WE=360-AT
1130 IF SIN(EA)<0 THEN WE=180+AT
1140 WE=INT(WE*100+.5)/100
1150 GOSUB 15000: REM OUTPUT THE RESULTS
1200 PRINT:PRINT "{C/DN}{C/DN}{C/DN}{C/DN}{C/DN}{C/DN}{C/DN}
}DO YOU WANT A NEW"
1210 PRINT"CALCULATION Y/N?"
1220 GET A$:IF A$="" THEN 1220
1230 IF A$="Y" THEN RUN
1240 IF A$="N" THEN END
1250 GOTO 1220
1299 REM
5000 REM *****
5010 REM *      CALCULATE THE ANOMALY *
5020 REM *****
5030 REM
5040 DIMF(33)
5050 PRINT:PRINT "PLEASE WAIT"
5060 P=1
5070 FOR I= 1 TO 33
5080 : P=P*I
5090 : F(I)=P
5100 NEXT
5110 F(0)=1
5120 REM *****
5200 REM * FOURIER-TRANSFORMATION *
5210 REM *      USING THE *
5220 REM * BESSEL FUNCTION *
5230 REM *****
5240 REM
5250 SUM=0
5260 FOR J=1 TO 33
5270 : G=EXJ
5280 : GOSUB 6000: REM BESSEL FUNCTION
5290 : B=(.5*G)↑J*S1
5300 : S2=SIN(J*MA)/J*B
5310 : IF ABS(S2)<=1E-20 THEN 5340
5320 : SUM=SUM+S2
5330 NEXT J
5340 EA=MA+2*SUM
5350 RETURN
6000 REM *****
6010 REM * BESSEL FUNCTION *
6020 REM *****
6030 REM
6040 S1=0
6050 FOR K=0 TO 30
```

```

6060 : BB=(-G*G/4)↑K
6070 : S3=BB/F(K)/F(J+K)
6080 : IF ABS(S3)<=1E-20 THEN 7000
6090 : S1=S1+S3
6100 NEXT K
7000 RETURN
10000 REM *****
10010 REM * INPUT THE VALUES *
10020 REM *****
10030 REM
10040 PRINTCHR$(147)
10050 INPUT"ECCENTRICITY          ";EX
10060 PRINT:INPUT"PERIOD FOR AN ORBIT      ";PU
10070 PRINT:INPUT"HALF-AXIS (LARGE)        ";GN
10080 PRINT:INPUT"TIME VALUE               ";T
10100 PRINT:PRINT"EVERYTHING CORRECT Y/N?"
10110 GET B$: IF B$="" THEN 10110
10120 IF B$="Y" THEN 10200
10130 IF B$="N" THEN 10040
10140 GOTO 10110
10200 RETURN
15000 REM *****
15010 REM * OUTPUT THE RESULTS *
15020 REM *****
15030 REM
15040 FOR I= 1 TO 24
15050 PRINT"
15060 NEXT I
15070 PRINT"ECCENTRICITY          ";EX
15080 PRINT
15090 PRINT"PERIOD OF AN ORBIT      ";PU
15100 PRINT
15110 PRINT"HALF AXIS              ";GH
15120 PRINT
15130 PRINT"TIME (DAYS)            ";T
15140 PRINT
15150 PRINT"ECCENTRICTY ANOMALIE        ";EA
15160 PRINT
15170 PRINT"RADIUS                  ";RM
15180 PRINT
15190 PRINT"ANGLE ETA                ";WE
15200 RETURN

```

READY.

Chapter 8: Computers in Biology

Two main areas of use for computers in biology are the processing of measured data and performing simulations. Just like the much-abused malingerer who tries to simulate the symptoms of an illness as accurately as possible, one tries to imitate real events with simulations on the computer. The closer one comes to the real relationships, the better conclusions can be drawn about the exact conditions. We now want to concern ourselves a bit with the nature of simulation techniques.

Why are simulations performed? The reasons are quite varied. In engineering, for example, one uses them in order to find ways of optimizing production methods, but simulations are also common in the sciences. Simulations are, as a general rule, used to save money, in order to prevent humans from being exposed to danger, or to investigate very comprehensive systems. It is naturally much less expensive to first try out a space ship on earth in a simulator, for example. All activities which will later be carried out during the flight can be tested individually and in the safety of the simulator. In biology, simulations in the form of mathematical models provide a view of larger systems, such as the symbioses of an area.

Simulations can be placed into certain categories. There are no sharp boundaries; many simulations are mixed types. One distinction can be made in regards to the goal of the model. There are mathematical models the purpose of which is to get the best possible yield from a chemical procedure by varying the initial parameters. A procedure

like this is called a "simulation with static results." In contrast to this there are simulations with "dynamic results" which attempt to make changes in the system dependent on time. The type of initial data used is also important in the classification. Systems which are performed with random values are designated as "accidental systems." If the border conditions of a model are planned, it is called a "specific simulation."

Simulation systems are usually very complex today and require quite powerful computers to run them. Naturally, we cannot approximate such a systems with the Commodore 64. Despite this, we can at least become better acquainted with the techniques of simulation. Several preconditions are given. With the RND(X) function we have the ability to create equally distributed random numbers between 0 and 1. Because of the internal accuracy of the Commodore 64 to 10 decimal places, it is of course impossible to represent all of the numbers in the interval (0,1). For that reason we do not get "real" random numebrs but so-called "pseudo-random" numbers. This does not affect their usability in our case.

In the following section you will find a simulation. It deals with the model of two animal populations in a closed living space (island) and the influence that the populations of both species have on each other.

8.1 The predator-prey model

A current branch of biology today is ecology,. Ecology deals with the changing relationships between the whole of life and the environment. Simulations are often used here to explore the relationships. Again we are dealing with a mathematical model. A fundamental model is the predator-prey relationship. The simplest model of this type presumes the following conditions: There are two different populations in a bounded living space. One type is designated as predator, the other as prey. The food source for the prey is assumed to be unlimited while the amount of nourishment available for the predators depends on the size of the prey population. One also assumes that the prey multiply without limit in the absence of the predators. Conversely, the predator population will decrease to the point of dying out in the absence of any prey.

The model cannot be found in nature of course. There are no completely closed systems in nature. This model simply serves as a basis for more refined simulations.

The Italian mathematician Volterra represented the model as a system of two differential equations of first order. The differential equations can be solved numerically.

The basic equations which describe a relationship of two things to each other are:

$$dN_1 = f(N_1, N_2) dt \quad (1)$$

$$dN_2 = g(N_1, N_2) dt \quad (2)$$

N_1 stands for the size of the prey population and N_2 represents the number of predators. From the conditions of the above explained model it follows that

$$dN_1 = A*N_1 dt \quad (3)$$

or

$$dN_2 = -D*N_2 dt \quad (4)$$

The condition between the predators and prey is assumed through the term N_1*N_2 with a positive sign for the predators and negative for the prey. Finally we get the Volterran equations

$$dN_1/dt = A*N_1 - B*N_2*N_1 \text{ or } dN_1/dt = (A - B*N_2)*N_1 \quad (5)$$

$$dN_2/dt = -D*N_2 + C*N_2*N_1 \text{ or } dN_2/dt = (-D + C*N_1)*N_2 \quad (6)$$

The A, B, C , and D are positive constants, the values of which one can vary and solve for certain reactions of the model. But what is hiding behind these constants? In principle one can say that they allow us to introduce the birth and death rates of the populations concerned into the model. The constants are assumed as follows:

A = prey birth rate

B = prey death rate

C = predator birth rate

D = predator death rate

It is first assumed that there is no change in the populations over time. For this case the constants B and C can be obtained from the following expressions:

$$B = A/N_2 \quad (7) \quad \text{and} \quad C = D/N_1 \quad (8)$$

N1 and N2 are our initial populations. We thereby get an equilibrium condition for the system which we will then proceed from. As soon as we change one of the constants, our mini-ecosystem goes out of balance. The results of our interventions can be seen in the result of the program operation.

The complete mathematical exploration of the Volterra equations is relatively comprehensive and cannot be repeated here. Here we want to limit ourselves to the best-known representation of the predator-prey model, namely the relationship of the populations to time. Charts which reflect such relationships graphically can be found in many biology books, usually without mathematical description.

In order to create such a chart, we must know N1 and N2 for various points in time. For this it is necessary to obtain the solutions to the Volterrann equations. The following program uses a method which you may already familiar with, the Runge-Kutta method. The numerical solution is taken here since the equation system does not have an analytical solution.

After all of this theory, we now want to present our program and execute a test run.

In addition to the Runge-Kutta routine from section 5.7, the program has an input block as well as two output subroutines. You can specify that your results go either to screen or to the printer. The high-resolution representation on the screen is done using a Simons' BASIC routine. An MPS-801 is used for the printer output.

If you do not have either Simons' BASIC or a printer, then you still have the option of inserting the two differential equations into the program DGL.SYS from section 5.7 and view the results as numbers. You then give the initial populations when asked for the initial conditions. You can write values you get down on paper. It is a good idea to determine and enter the values logarithmically.

Program construction:

100-350	Dimensioning the data arrays
	Set color
	New simulation, end
	Subroutine call
3000-3290	Runge-Kutta routine (diff-eq. systems)
5000-5050	Equation system
at 15000	Input
at 30000	Screen graphics routine
30010-30060	Determine maximum and minimum populations
30070-30080	Conversion to screen size
31010-31070	Text and coordinate axes
32010-32060	Calculate print positions
	Plot the curve with LINE (smoothing)
32070	High-res off, return to main program
40000-46040	Printer subroutine (MPS-801)

A test run was executed with the following input:

A = 2.2, B = 0.00563, C = 0.0004, D = 1.2

Initial prey population: 4000

Initial predator population: 500

Start time: 0, end time: 14 (arbitrary units)

Time interval: 0.2 units

You can find the result of this example on the next page (figure 8.1a). It is quite easy to identify the relationship of the populations to each other. The predator population reacts about a fourth of a phase shifted behind the growth of the prey. The whole system maintains a so-called dynamic equilibrium. You can identify this even more easily if you take time of 0 to 70 instead of 0 to 14 and select a time interval of 1.

You can choose any desired values for the various input parameters, but it is possible that extreme values will not fit on the screen, although this should not happen as a general rule. If it does occur, you must restart the program and change the parameters.

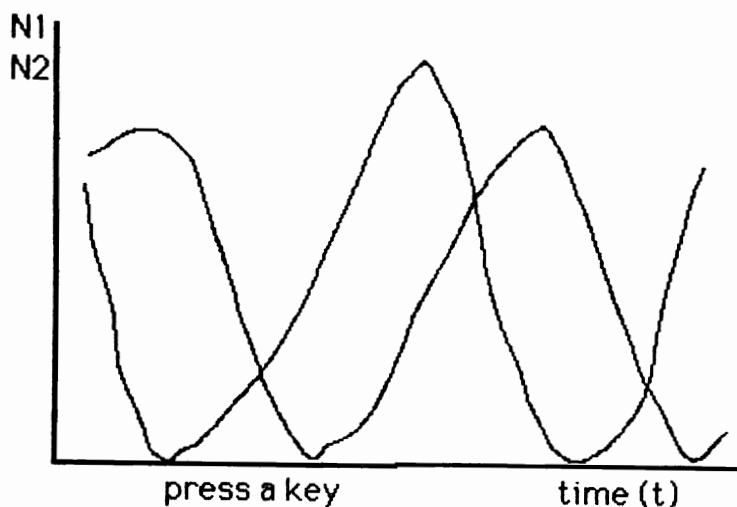


fig. 8.1a

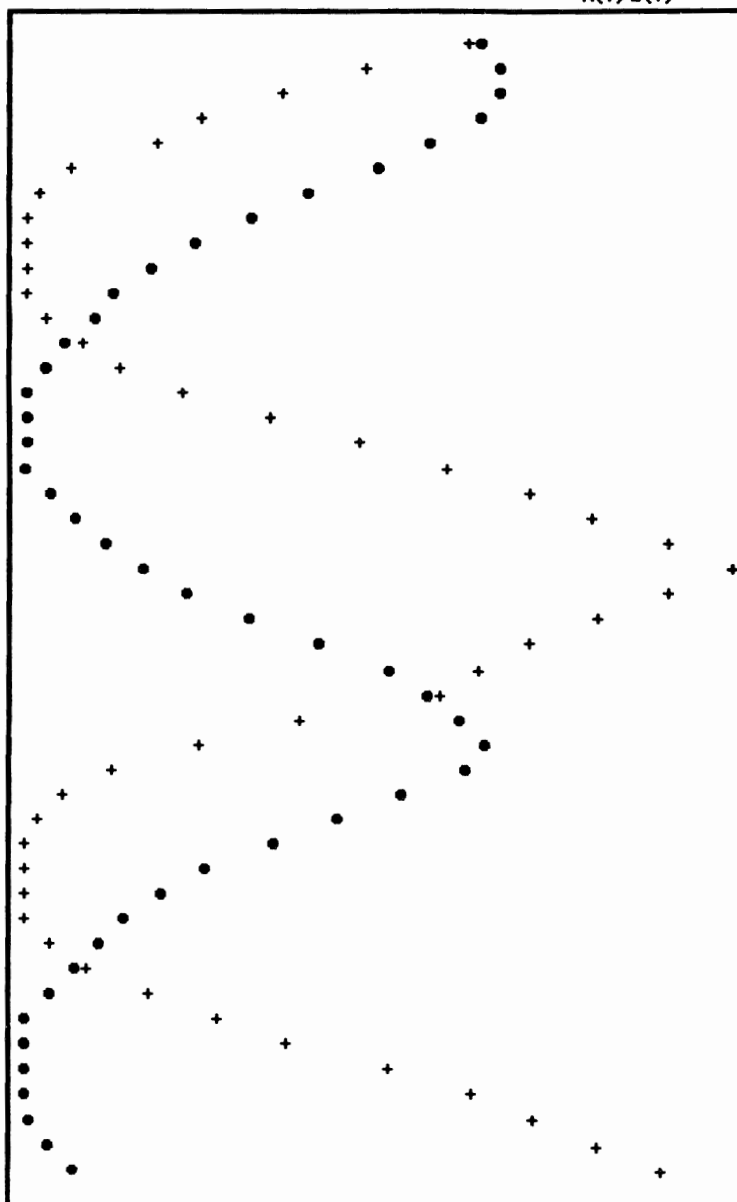
Predator (MAX.) = 539

Predator (MIN.) = 272

Prey (MAX.) = 4600

Prey (MIN.) = 1822 R(T) B(T)

.15
.3
.45
.6
.75
.9
1.05
1.2
1.35
1.5
1.65
1.8
1.95
2.1
2.25
2.4
2.55
2.7
2.85
3
3.15
3.3
3.45
3.6
3.75
3.9
4.05
4.2
4.35
4.5
4.65
4.8
4.95
5.1
5.25
5.4
5.55
5.7
5.85
6
6.15
6.3
6.45
6.6
6.75
6.9
time



```

10 REM *****
20 REM *
30 REM *      DYNAMIC POPULATION      *
40 REM *
50 REM *      MODEL                    *
60 REM *
70 REM *      USING VOLTERRA/LOTKA    *
80 REM *
90 REM *****
100 CLR
110 DIM Y(500),Z(500),K1(50),K2(50),K3(50),K4(50),RZ(50),B
    (100),R(100),T(100)
120 POKE 53280,0:POKE 53281,0
130 PRINT"{CLR}"
150 REM
160 GOSUB 20000:REM *      TITLE      *
200 REM
210 GOSUB 15000:REM *      INPUT      *
212 REM
220 GOSUB 3000:REM *      RUNGE/KUTTA *
222 REM
225 POKE 214,12:POKE 211,0:SYS 58732
230 PRINT"{WHT}OUTPUT TO PRINTER      (Y/N)"
232 REM
235 GET A$:IF A$="" THEN 235
236 REM
240 IF A$="Y" THEN GOSUB 40000:GOTO 310
242 REM
245 IF A$="N" THEN GOSUB 30000:GOTO 310
246 REM
250 REM *      PRINTER: GOSUB 40000 *
255 REM
260 REM *      GRAPHICS:GOSUB 30000 *
280 GOTO 225
300 REM
310 PRINT"{CLR}NEW SIMULATION (Y/N)"
315 POKE 198,0
320 GET A$:IF A$="" THEN 320
330 IF A$="Y" THEN RUN
340 IF A$="N" THEN PRINT"{CLR}BYE!!!!!! ":END
350 GOTO 310
3000 REM
3001 REM *****
3002 REM *      RUNGE-KUTTA-PROCEDURE  *
3003 REM *****

```

```

3004 REM
3006 X=XA
3010 FOR J=1 TO N
3020 : Y(J)=Z(J)
3030 NEXT J
3050 FOR I=0 TO NR-1
3060 :GOSUB5000
3070 : FOR J=1 TO N
3080 : : RZ(J)=Z(J)
3090 : : K1(J)=IN*ZP(J)
3100 : : Z(J)=RZ(J)+K1(J)/2
3110 : NEXT J
3120 : X=X+IN/2
3130 :GOSUB5000
3140 : FOR J=1 TO N
3150 : : K2(J)=IN*ZP(J)
3160 : : Z(J)=RZ(J)+K2(J)/2
3170 : NEXT J
3180 :GOSUB5000
3190 : FOR J=1 TO N
3200 : : : K3(J)=IN*ZP(J)
3210 : : : Z(J)=RZ(J)+K3(J)
3220 : NEXT J
3225 : X=X+IN/2
3227 :GOSUB5000
3228 : FOR J=1 TO N
3230 : : K4(J)=IN*ZP(J)
3240 : : Z(J)=RZ(J)+(K1(J)+2*(K2(J)+K3(J))+K4(J))/6
3250 : NEXT J
3270 : T(I)=T(I)+X:B(I)=B(I)+Z(1):R(I)=R(I)+Z(2)
3280 NEXT I
3290 RETURN
5000 REM
5001 REM *****
5003 REM * EQUATION SYSTEM *
5004 REM *****
5020 REM
5030 ZP(1)=(A-B*Z(2))*Z(1)
5040 ZP(2)=(-D+C*Z(1))*Z(2)
5050 RETURN
15000 REM
15010 REM *****
15020 REM * INPUT *
15025 REM *****
15027 REM
15030 PRINT"PLEASE ENTER:"
15040 N=2
15060 INPUT"(WHT){C/DN}{C/DN}START TIME: ";XA

```

```

15070 INPUT "{C/DN}{C/DN}END TIME: ";XE
15075 IF XA<0 THEN 15060
15080 IF XE<=XATHEN15070
15090 INPUT "{C/DN}{C/DN}TIME INTERVALL: ";IN
15100 IF IN<=0 THEN15090
15110 NR=(XE-XA)/IN
15120 PRINT "{GRY2}{C/DN}{C/DN}INITIAL SIZE: {C/DN}"
15130 PRINT "{WHT}{C/DN}{C/DN}FOR T = ";XA;" IS : {C/DN}"
15150 INPUT "{C/DN}{C/DN}PREY POPULATION: ";Z(1)
15160 INPUT "{C/DN}{C/DN}PREDATOR POPULATION: ";Z(2)
15170 PRINT "{CLR}"
15180 REM
15190 PRINT "{GRY2}ENTER THE R CONSTANTS A,B,C,D
15200 INPUT "{WHT}{C/DN}{C/DN}CONSTANT A: ";A
15210 INPUT "{C/DN}{C/DN}CONSTANT B: ";B
15220 INPUT "{C/DN}{C/DN}CONSTANT C: ";C
15230 INPUT "{C/DN}{C/DN}CONSTANT D: ";D
15240 POKE214,12:POKE211,12:SYS58732
15250 PRINT "{GRY2}PLEASE WAIT!!!"
15300 RETURN
20000 REM
20010 REM *****
20020 REM * TILTE VOLTERRA/LOTKA *
20030 REM *****
20040 REM
20050 POKE 214,11:POKE 211,9 :SYS 58732
20060 PRINT "{GRY2}POPULATION SIMULATION"
20070 POKE 214,13:POKE 211,5:SYS 58732
20080 PRINT "{GRY2}WITH THE VOLTERRA/LOTKA-MODEL"
20100 FOR I=1TO2000:NEXT:PRINT "{CLR}":POKE198,0
20110 RETURN
30000 REM
30001 REM *****
30002 REM * GRAPHICS-ROUTINE (SIMONS) *
30003 REM *****
30005 REM
30010 BM=B(0):BN=BM:RM=R(0):RN=RM
30020 FOR I=1 TO NR-1
30030 : IF BM<B(I) THEN BM=B(I)
30040 : IF BN>B(I) THEN BN=B(I)
30050 : IF RN>R(I) THEN RN=R(I)
30055 : IF RM<R(I) THEN RM=R(I)
30060 NEXT I
30070 KB=(BM-BN)/150
30080 KR=(RM-RN)/120
31010 HIRES 1,0
31020 TEXT 3,3,"N1",1,1,9

```

```

31025 TEXT 3,12,"N2",1,1,9
31030 TEXT 50,190,"PRESS A KEY",1,1,9
31040 LINE 20, 8,20,190,1
31050 LINE 10,180,320,180,1
31060 TEXT 250,185,"TIME (T)",1,1,9
31070 TEXT 60,3,"N1 (PREY) = UPPER CURVE",1,1,8
32010 FOR I=0 TO NR-1
32020 T(I)=300/(XE-XA)*T(I)+20:B(I)=180-(B(I)-BN)/KB
32030 R(I)=180-(R(I)-RN)/KR
32040 IF I<>0 THEN LINE T(I-1),B(I-1),T(I),B(I),1
32050 IF I<>0 THEN LINE T(I-1),R(I-1),T(I),R(I),1
32060 NEXT I:GOTO PRINT
32070 WAIT198,255:CSET 0: RETURN
40000 REM
40001 REM *****
40002 REM * PRINTER OUTPUT LOW RES *
40003 REM *****
40005 REM
40010 BM=B(0):BN=BM:RM=R(0):RN=RM
40020 FOR I=1 TO NR-1
40030 : IF B(I)>BM THEN BM=B(I)
40040 : IF B(I)<BN THEN BN=B(I)
40050 : IF R(I)>RM THEN RM=R(I)
40060 : IF R(I)<RN THEN RN=R(I)
40070 NEXT I
40090 :
40100 KB=(BM-BN)/65
40110 KR=(RM-RN)/45
40200 OD$="
40210 OPEN 1,4:PRINT#1,"PREDATOR(MAX.)=";INT(RM+.5);
40220 PRINT#1,CHR$(16)"40PREDATOR(MIN.)=";INT(RN+.5)
40230 PRINT#1,"PREY (MAX.)=";INT(BM+.5);
40240 PRINT#1,CHR$(16)"40PREY (MIN.)=";INT(BN+.5)
40250 PRINT#1,CHR$(16)"60R(T) B(T)"CHR$(16)"77\"CHR$(8):
PRINT#1,CHR$(15);
40260 PRINT#1," {RVON} {RVOF} "+OD$+OD$+"\"CHR$(8)
):PRINT#1,CHR$(15);
40300 FOR I=0 TO NR-1
40310 : WB=(B(I)-BN)/KB+10
40320 : WR=(R(I)-RN)/KR+10
40330 : BZ=INT(WB/10):BE=INT(WB-10*BZ)
40340 : RZ=INT(WR/10):RE=INT(WR-10*RZ)
40350 :
40360 : PRINT#1,INT(T(I)*100+.5)/100;
40370 PRINT#1,CHR$(16)"10 " ;
40380 : IF WR>WB THEN GOSUB 45000
40390 : IF WR<=WB THEN GOSUB 46000
40400 PRINT#1,CHR$(8):PRINT#1,CHR$(15);:
40410 PRINT#1,CHR$(16)"10 "CHR$(8):PRINT#1,CHR$(15);

```

```
40420 NEXT I:
40430 PRINT#1,"  TIME  \/"
40440 PRINT#1:PRINT#1:PRINT#1:PRINT#1:RETURN
45000 REM
45001 REM
45002 REM
45010 PRINT#1,CHR$(16)MID$(STR$(BZ),2,1)MID$(STR$(BE),2,1)
"+";
45020 PRINT#1,CHR$(16)MID$(STR$(RZ),2,1)MID$(STR$(RE),2,1)
"@";
45030 RETURN
46000 REM
46001 REM
46010 PRINT#1,CHR$(16)MID$(STR$(RZ),2,1)MID$(STR$(RE),2,1)
"@";
46020 PRINT#1,CHR$(16)MID$(STR$(BZ),2,1)MID$(STR$(BE),2,1)
"+";
46030 RETURN
```

READY.

Chapter 9: The IEEE Bus and its Possibilities

Commodore gave its '64 several possibilities for interfacing with the rest of the world. One such possibility is the user port, onto which one can connect a parallel printer with the appropriate software. Many others applications are also possible here.

In addition to the user port there is the expansion port which accepts cartridges and can serve as the input/output for disk drives, datasette, joysticks, etc.

The various interface options of this port have been explored in many publications. Especially important for scientific purposes is the ability to connect something called an IEEE interface. There are many different devices which use this interface. One speaks of IEEE 488, HP-IB, or even IEC 625. The most prevalent interface standard is the IEEE standard from 1978.

The most important characteristic of the IEEE bus is the capability to attach many external devices simultaneously. The bus was designed for extensive measurement systems. These peripherals can consist of volt meters, pH-measuring devices, plotters, and so on. This is one great advantage of the IEEE bus compared to other bus systems. The IEEE bus transmits data in parallel.

The devices on an IEC/IEEE 488 bus system are placed into one of the following three categories:

1. Sender of data (talker)
2. Receiver of data (listener)
3. Control device (controller)

Concerning the first group, there are devices on one side which can practice only one of the functions while others serve both (talk/listen). The controller function is generally assumed by only one device. In our case, the controller will be the computer itself. The controller is the "brains" of the whole system. It decides which device may send or receive at any given time. Otherwise all of the components would start to transmit through each other. The coding of the addresses is done with the ASCII code, also known as the ISO 7-bit code.

The bus is quite fast (parallel!), and one can increase this speed by installing tri-state drivers!

All devices are simply connected together in parallel by piggy-backing the connectors ontop of each other. The IEC/IEEE bus can be connected to the expansion port or to the user port. The former has the advantage that it can execute all of the functions of the Commodore 64's CPU from "outside."

The bus connector of the IEC bus has 25 lines while the IEEE 488 standard uses 24-pin connectors. The list below illustrates the different pin-outs of the two standards.

Pin	IEC 625	IEEE 488
1	DATA I/O 1	DATA I/O 1
2	DATA I/O 2	DATA I/O 2
3	DATA I/O 3	DATA I/O 3
4	DATA I/O 4	DATA I/O 4
13	GND	DATA I/O 5
14	DATA I/O 5	DATA I/O 6
15	DATA I/O 6	DATA I/O 7
16	DATA I/O 7	DATA I/O 8
17	DATA I/O 8	REN
18 to 25/24	earth/logic gnd	
6	EOI	DAV
7	DAV	NRFD
8	NRFD	NDAC
9	NDAC	IFC
10	IFC	SRQ
11	SRQ	ATN
12	ATN	shield
13	shield	
5	REN	EOI

In closing we might remind you that all of the peripherals from the Hewlett-Packard company are available to you over the IEEE bus. The same also applies to the Commodore PET line of peripherals (disk drives, printers, etc.).

Chapter 10: Technical Programs

10.1 Heat transmission through a multi-layer plate

The program calculates the loss of heat through a one or more layer plate and the layer border temperatures between the layers.

Even a window is such a plate. A simple window is a single-layer glass plate; an isolation glass window is usually a three-layer plate of glass, a layer of rarefied gas and an additional layer of glass.

The program clears the screen and asks for the surface area, the number of plates, their thickness and heat conduction constants.

Enter the corresponding values. If you enter a nonsensical value, such as a surface less than or equal to zero, the program will not do anything. You can then enter the correct value.

If you should enter a letter instead of a digit, the message "redo from start" will appear. The message comes from the operating system and means that the input should be repeated. Here again you continue with the correct input of the value.

The variables used in the program are:

F is the surface
 N is the number of plates
 WV is the heat transmission constant
 K(N) is the dimensioned variable with N+1 values (0 to N), heat transmission constants
 T(N) are the border temperatures
 D(N) are the thicknesses of the layers

The heat transmission constants are calculated first. Because the denominator is a sum, it is formed in a loop.

$$(1) WV = \frac{T(0) - T(N)}{\sum \frac{D(I)}{K(I) * F}}$$

The individual border temperatures are then calculated.

$$(2) T(I) = T(I-1) - \frac{WV * D(I)}{K(I) * F}$$

T(0) is the given temperature of the upper surface. All of the input must be made in SI units, that is, the surfaces must be given in square meters, thicknesses in meters, heat transmission constant in joules per meter, and temperatures in Kelvin. You can also use Celsius instead of Kelvin, but the results will then also be in Celsius. The same is true if you use kilocalories instead of joules.

The program outputs the heat transmission constants in joules per hour and the border temperatures. The maximum number of layers is not limited by the program. If only this program is in memory, you can enter several thousand layers.

```

10 REM *****
20 REM *
30 REM * HEAT TRANSMISSION *
40 REM *
50 REM *****
60 :
100 POKE53280,0:POKE53281,0
1000 GOSUB 15000
2000 REM
2001 REM CALCULATIONS
2010 NE=0
2020 FOR I=1 TO N
2030 NE=NE+D(I)/(K(I)*F)
2040 NEXT I
2050 REM LOSS OF HEAT CALCULATION
2060 WV=(T(0)-T(N))/NE
2070 REM CALCULATION BORDER LAYER TEMPERATURES
2080 FOR I= 1 TO N
2090 T(I)=T(I-1)-(WV*D(I))/(K(I)*F)
2100 NEXT I
10000 REM
10001 REM OUTPUT
10002 REM
10010 PRINT"{CLR}{C/DN}{WHT}HEAT LOSS IN JOULES/HPOUR: ":P
RINT,WV
10020 PRINT"{C/DN}{WHT}FOR LAYER #","TEMPERATURE"
10030 FOR I=1 TO N
10040 PRINTI,,T(I)
10050 NEXT
14000 REM
14001 REM ** MORE CALCULATIONS **
14002 REM
14010 POKE211,5:POKE214,22:SYS58732
14020 PRINT"{GRY2}MORE CALCULATIONS {WHT}{Y/N}?"
14030 GETX$
14040 IF X$="Y" THEN RUN
14050 IF X$="N" THEN END
14060 GOTO 14030
15000 REM
15001 REM **INPUT **
15002 REM
15010 PRINT"{CLR}{GRY2}{C/DN}{C/DN}{C/RT}{C/RT}{C/RT}HEAT
TRANSMISSION THROUGH A"
15020 PRINT"{C/DN}{C/RT}{C/RT}{C/RT}{C/RT}MULTI-LAYER META
L PLATE"

```

```

15030 PRINT"{C/DN}{C/DN}SURFACE AREA (SA)":INPUT F
15040 IF F<=0 THEN PRINT "{C/UP}{C/UP}{C/UP}{C/UP}";:GOTO1
5030
15050 PRINT"{C/DN}{PURP}NUMBER OF LAYERS(WHT)":INPUT N:IF
N<1 THEN PRINT"{C/UP}{C/UP}{C/UP}{C/UP}";:GOTO15050
15060 DIMD(N): REM DIMENSION FOR N LAYERS
15070 DIMK(N): REM DIMENSION FOR N LAYERS
15080 FOR I= 1 TO N: REM LOOP FOR THE N LAYERS
15090 PRINT"{C/DN}{PURP}THICKNESS OF ";I;"TH LAYER":INPUTD
(I)
15100 IF D(I)<=0 THEN PRINT"{C/UP}{C/UP}{C/UP}";:GOTO15090
15110 PRINT"{C/DN}{PURP}HEAT CONDUCTION NUMBER OF ";I;"TH
LAYER"
15120 PRINT"IN JOULE/M * H *K ":INPUT K(I)
15130 IF K(I)<=0 THEN PRINT"{C/UP}{C/UP}{C/UP}{C/UP}";:GOT
O15110
15140 PRINT:NEXT I
15150 PRINT"{C/DN}{C/DN}{GRY2}TEMP IN FRONT OF THE 1ST LAY
ER (K OR C)":INPUT T(0)
15160 PRINT"{C/DN}{PURP}TEMP BEHIND THE LAST LAYER":INPUT
T(N)
15170 IF T(N)>T(0) THEN PRINT"{C/UP}{C/UP}{C/UP}";:GOTO 15
160
15180 RETURN

```

READY.

10.2 Pulley belt length calculation

The length of a belt which runs around two or three pulleys will be calculated here.

A) Program for two pulleys

Variables

R1 radius of the larger pulley
R2 radius of the smaller pulley
D distance between the pulley centers
D1 "hypotenuse" of pulley
T angle T between the tangents of the pulleys and
 the lines connecting the pulley centers

A parallel to the tangents through the center of the smaller pulley is used to graphically represent the angle. The angle is given in radians because in BASIC the angle functions recognize only radians.

The input is accomplished in the lines 15000 to 15080. Please enter the radius of the larger pulley first so that the angle T is not negative. If you enter a value which turns out to be smaller than that of the smaller pulley, it will not be accepted by the program. This condition is checked in line 15050. Nonsensical cases are also singled out, cases in which the distance between the centers is less than the sum of radii. If this is the case then the pulleys will overlap.

The length D1 is calculated with the help of the Pythagorean theorem. The following relationship applies:

$$(1) \quad D1 = \sqrt{D^2 - (R1-R2)^2}$$

This length D1 is contained twice in the total belt length. The circumference portions of the pulleys must also be taken into account. The exponential operator is not used in line 1010 of the program because it is both slower and less accurate than explicit multiplication.

$$(2) \quad \frac{(R1-R2)}{D} = \sin T$$

Is the relationship between the opposite leg and the hypotenuse of the angle T. The arcsine of this value returns the angle in radians. Since the arcsine function is not implemented in BASIC, it must be defined. This is done with the help of a derivative of the arctangent (ATN) in line 410.

The following applies for the angles W2 and W1:

$$(3) \quad W2 = 2 * \frac{\pi}{2T} \quad W1 = 2\pi - W2$$

The lengths L1 and L2 can be calculated as

$$(4) \quad L1 = W1 * R1 \quad L2 = W2 * R2$$

The total belt length is

$$(5) \quad L = L1 + L2 + 2 * D1$$

B) Program for three pulleys

In principle this program is not much different from the previous one for two pulleys, but additional problems occur which will be explained here.

First the variable allocations:

R1	radius of pulley 1
R2	radius of pulley 2
R3	radius of pulley 3
D1	distance between pulleys 2 and 3
D2	distance between pulleys 1 and 3
D3	distance between pulleys 1 and 2
W1	angle between the two radii at pulley 1
W2	radii angle at pulley 2
W3	radii angle at pulley 3
A1	angle between the two connecting lines which meet at pulley 1
A2	angle at pulley 2
A3	angle at pulley 3
T1	angle between line connecting 2 - 3 and tangents 2 - 3
T2	angle between line connecting 1 - 3 and tangents 1 - 3
T3	angle between line connecting 2 - 3 and tangents 1 - 2

Here all three distances are taken into consideration. Together they form a triangle. In addition, the pulleys must be sorted according to size so that the angle T does not become negative. Here the sorting is carried out in the program. The order of entry is therefore irrelevant.

During the input the distances are checked to see if they are larger than the sums of the radii of the appropriate pulleys and if the three distances given can be the sides of a triangle.

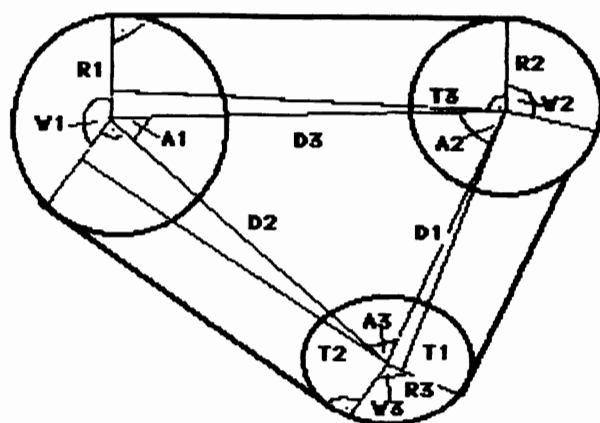
After the input in lines 15000-15080, the pulleys are sorted in decreasing order according to their radii. The distances must be exchanged at the same time.

Finally the tangent lengths must be calculated using the Pythagorean theorem. The T angles are always positive, that is, always given as in figure 10.2 a. One side of the angle is parallel to the appropriate tangent, the other is the connecting line between the centers of the pulleys.

The cosine law is used in order to calculate the A angles of the triangle. Since only segment lengths and no angles are known, this relatively complicated procedure is necessary. The arc cosine function, also not found in BASIC, is defined in line 440 as a user function as a derivative of the arctangent function.

Addition or subtraction of angles yields the angle W between the radii with which the circumference portions are then calculated.

The total length is added up in line 1270 and printed in line 10010.



```

10 REM *****
20 REM *
30 REM * BELT LENGTH *
40 REM * FOR TWO PULLEYS *
50 REM *
60 REM *****
70 :
100 POKE 53280,0:POKE53281,0
110 POKE 788,52:REM INHIBIT STOP
400 REM
401 REM *** DEFINE FUNCTIONS ***
402 REM
410 REM ARCSIN
420 DEF FN AS(X)=ATN(X/SQR(1-X*X))
500 GOSUB 15000
1000 REM
1001 REM *** CALCULATIONS**
1002 REM
1010 D1=SQR(D*D-(R1-R2)*(R1-R2))
1020 T=FN AS((R1-R2)/D)
1030 W2=2*(PI/2-T):W1=2*PI-W2
1040 L1=W1*R1:L2=W2*R2
1050 L=L1+L2+2*D1
10000 REM
10001 REM *** OUTPUT ***
10002 REM
10010 PRINT"{WHT}{C/DN}{C/DN}NECESSARY BELT LENGTH=";L
14000 PRINT"{C/DN}{C/DN}{C/DN}{C/RT}{C/RT}{C/RT}{C/RT}{C/R
T}{C/RT}{GRY2}ADDITIONAL CALCULATIONS (Y/N)"
14010 GETX$:IF X$="Y" THEN RUN
14020 IF X$="N" THEN POKE 788,49:END
14030 GOTO 14010
14999 END
15000 REM
15001 REM *** INPUT ***
15002 REM
15010 PRINT"{CLR}{GRY2} BELT LENGTH CALCULATIONS"
15020 PRINT"{GRY2} FOR TWO PULLEYS"
15030 PRINT"{GRY2}{C/DN}RADIUS OF LARGER PULLEY{WHT}";:INP
UT R1
15040 PRINT"{GRY2}{C/DN}RADIUS OF SMALLER PULLEY{WHT}";:IN
PUT R2
15050 IF R2>R1 THEN PRINT"{C/UP}{C/UP}{C/UP}":GOTO 15040
15060 PRINT"{GRY2}{C/DN}DISTANCE OF THE MIDPOINT{WHT}";:IN
PUT D

15070 IF D<=R1+R2 THEN PRINT"{C/UP}{C/UP}{C/UP}":GOTO 1506
0
15080 RETURN

READY.

```

```

10 REM *****
20 REM *
30 REM * BELT LENGTH *
40 REM * FOR THREE PULLIES *
50 REM *
60 REM *****
70 :
100 POKE 53280,0:POKE 53281,0
110 POKE 788,52:REM * STOP KEY BLOCK *
400 REM
401 REM *****
402 REM * DEFINE FUNCTION *
403 REM *****
404 REM
410 REM * ARCSIN *
411 REM
420 DEF FN AS(X)=ATN(X/SQR(1-X*X))
421 REM
430 REM *ARCCOS *
431 REM
440 DEF FN AC(X)=-ATN(X/SQR(1-X*X))+PI/2
500 GOSUB 15000
1000 REM
1001 REM *****
1002 REM * CALCULATIONS *
1003 REM *****
1004 REM
1010 IF R3>R2 THEN X=R3:R3=R2:R2=X:X=D1:D1=D2:D2=X
1020 IF R2<=R1 THEN 1050
1030 X=R2:R2=R1:R1=X:X=D2:D2=D3:D3=X
1040 IF R3>R2 THEN X=R3:R3=R2:R2=X:X=D1:D1=D2:D2=X
1050 REM * RADII SORTED IN DEC. ORDER *
1060 :
1070 E1=SQR(D1*D1-(R2-R3)*(R2-R3)):REM* TANGENT 1 *
1080 E2=SQR(D2*D2-(R1-R3)*(R1-R3)):REM* TANGENT 2 *
1090 E3=SQR(D3*D3-(R1-R2)*(R1-R2)):REM* TANGENT 3 *
1100 :
1110 T1=FN AS((R2-R3)/D1):REM*THETA 1*
1120 T2=FN AS((R1-R3)/D2):REM*THETA 2*
1130 T3=FN AS((R1-R2)/D3):REM*THETA 3*
1140 :
1150 A1=FN AC((D1*D1-D2*D2-D3*D3)/(-2*D2*D3)):REM*ANGLE AL
PHA 1, COSINE RULE*
1160 A2=FN AC((D2*D2-D3*D3-D1*D1)/(2*D3*D1)):REM*ANGLE ALP
HA 2, COSINE RULE *

```

```

1170 A3=PI-A1-A2:REM*ANGLE ALPHA3 *
1180 :
1190 W1=PI-A1+T2+T3:REM*ANGLE BETWEEN RADII AT 1 *
1200 A2=FN AC((D2*D2-D3*D3-D1*D1)/(2*D3*D1)):REM*ANGLE AL
PHA 2, COSINE RULE*
1210 W2=PI-A2-T3+T1:REM*ANGLE BEI 2*
1220 W3=PI-A3-T1-T2:REM*ANGLE BEI 3*
1230 L1=W1*R1:REM*CIRCUMFERANCE PORTION 1*
1240 L2=W2*R2:REM*CIRCUMFERANCE PORTION 2*
1250 L3=W3*R3:REM*CIRCUMFERANCE PORTION 3*
1260 :
1270 L=L1+L2+L3+E1+E2+E3:REM*TOTAL LENGTH*
1280 :
10000 REM
10001 REM *****
10002 REM * OUTPUT *
10003 REM *****
10004 REM
10010 PRINT "{WHT}{C/DN}{C/DN}NECESSARY BELT LENGTH=";CHR#
(5);L
14000 PRINT"{GRY2}{C/DN}{C/DN}{C/DN}{C/RT}{C/RT}{C/RT}{C/R
T}{C/RT}{C/RT}ADDITIONAL CALCULATIONS? (Y/N)"
14010 GET X$:IF X$="Y" THEN RUN
14020 IF X$="N" THEN POKE788,49:END
14030 GOTO 14010
14999 END
15000 REM
15001 REM *****
15002 REM * INPUT *
15003 REM *****
15010 PRINT "{GRY2}{CLR} BELT LENGTH CALCULATION"
15020 PRINT " FOR THREE PULLIES"
15030 PRINT "{C/DN}RADIUS OF PULLEY 1{WHT}";:INPUT R1
15040 PRINT "{GRY2}{C/DN}RADIUS OF PULLEY 2{WHT}";:INPUT R
2
15050 PRINT "{GRY2}{C/DN}RADIUS OF PULLEY 3{WHT}";:INPUT R
3
15060 PRINT "{GRY2}{C/DN}DISTANCE OF MIDPOINT 1 - 2{WHT}";
:INPUT D3
15070 IF D3<=R1+R2 THEN PRINT"{C/UP}{C/UP}{C/UP}":GOTO 150
60
15080 PRINT "{GRY2}{C/DN}DISTANCE OF MIDPOINT 2 - 3{WHT}";
:INPUT D1
15090 IF D1<=R2+R3 THEN PRINT"{C/UP}{C/UP}{C/UP}":GOTO 150
80
15100 PRINT "{GRY2}{C/DN}DISTANCE OF MIDPOINT 3 - 1{WHT}";
:INPUT D2
15110 IF D2<=R1+R3 THEN PRINT"{C/UP}{C/UP}{C/UP}":GOTO 151

```


10.3 Analysis of complex networks

10.3.1 Complex numbers (currents, potentials, resistances)

The occurrence of complex quantities, contingent on functions of time, cannot be explained here. We want to concern ourselves with complex numbers only as far as is necessary for using the programs for network current and node potential analyses.

A complex number is shown represented in figure 10.3.11. The two most important forms for us and the mathematical connection are shown. The required quantities for the two analysis procedures are asked for in real and imaginary parts, so a program to convert between the two methods of representation of complex numbers is necessary.

If instead of real and imaginary portions the length and angle for the complex resistance of the component (inductance--Henry, capacitance--farad) is given, an additional program is necessary which calculates complex resistance of the combination circuit.

Naturally, networks with purely Ohmic components and which contain only direct-current sources can also be handled by the analysis procedures presented in sections 10.3.3 and 10.3.4. Here the imaginary portion can be ignored, that is, set to zero.

Now we come to the program:

Conversion of complex numbers

The program is general enough so that the values (length and angle or real and imaginary parts) can be entered without regard to the quantities, that is, current, potential, or resistance. You receive as output the other method of representing a complex number. Several numbers can be entered consecutively, the numbers are numbered according to order.

1100 calculation of real and imaginary portion

1300 calculation of length and length

10100 output of real and imaginary portion

10300 output of length and angle

15200 input of length and angle

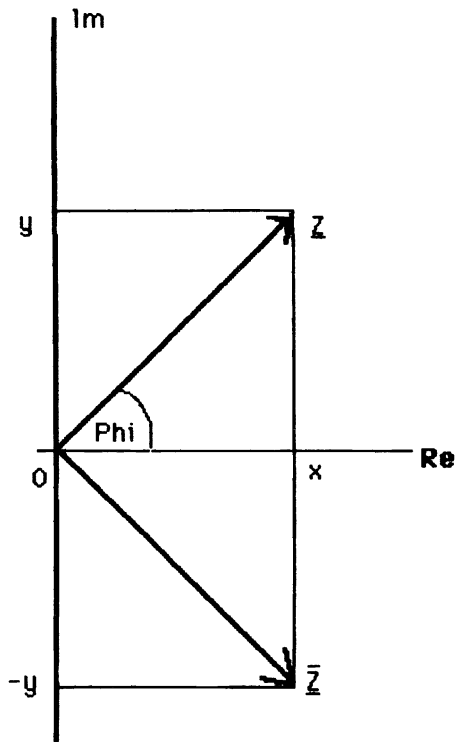
15300 input of real and imaginary parts

Calculation complex resistances

The components and two of the most common types of combination circuits are presented to you here. After selecting the desired resistance form (in ohms, millihenrys, or microfarads), the values are entered. Please note that a calculation is not possible without the frequency. This is given in hertz.

You receive as output the real and imaginary portions of the complex resistance in ohms.

1100 calculation of the ohmic resistance (R)
1200 calculation of the inductance (L)
1300 calculation of the capacitance (G)
1400 calculation of the serial circuit (R+L)
1500 calculation of the parallel circuit (R//G)
10100 output of the complex resistance in real, imag.
portions
15400 input resistance
15500 input inductance
15600 input capacitance
15700 input for parallel circuit
15700 input for serial circuit



representing a complex number Z

$$Z = x + jy$$

$$\underline{Z} = |\underline{Z}| e^{j\text{Phi}}$$

$$\text{with } |\underline{Z}| = \sqrt{x^2 + y^2}, \text{Phi} = \arctan y/x$$

```

10 REM *****
20 REM *
30 REM *
40 REM *CONVERTING COMPLEX NUMBERS *
50 REM *
60 REM *
70 REM *****
80 REM
90 REM
100 POKE 53280,0:POKE53281,0:PRINT" (GRY2) "
500 GOSUB 15000
510 GOSUB10000
520 END
1000 REM
1010 REM *****
1020 REM * MATHEMATICAL ROUTINES *
1030 REM *****
1040 REM
1100 REM
1110 REM *****
1120 REM * CALC. REAL IMAGINARY *
1130 REM *****
1140 REM
1150 FOR I=1 TO N
1160 :X(I)=Z(I)*COS(P(I)* $\pi$ /180)
1170 :Y(I)=Z(I)*SIN(P(I)* $\pi$ /180)
1180 :IFP(I)>-90 AND P(I)<90 THEN X(I)=-X(I)
1190 NEXT I
1200 RETURN
1300 REM
1310 REM *****
1320 REM * CALC. LENGTH-ANGLE *
1330 REM *****
1340 REM
1350 FOR I= 1 TO N
1360 :Z(I)=SQR(X(I)*X(I)+Y(I)*Y(I))
1370 IF X(I)=0 AND Y(I)>0 THEN P(I)= 90
1380 IF X(I)=0 AND Y(I)<0 THEN P(I)=-90
1390 : IF X(I)=0 THEN 1440
1400 : O(I)=ATN(Y(I)/X(I))*180/ $\pi$ 
1410 : IF X(I)>0 THEN P(I)=O(I)
1420 : IF X(I)>0 AND Y(I)>0 THEN P(I)=O(I)+180
1430 : IF X(I)<0 AND Y(I)<0 THEN P(I)=O(I)-180
1440 NEXT I
1450 RETURN

```

```

10000 REM
10010 REM *****
10020 REM *      OUTPUT ROUTINE      *
10030 REM *****
10100 REM
10101 REM *****
10102 REM *      OUTPUT REAL IMGINARY  *
10103 REM *****
10104 REM
10107 IF M=2 THEN 10200
10110 FOR I= 1 TO N
10120 :PRINT"{CLR}":Z=4:GOSUB 17000
10130 :PRINT"OUTPUT";I;"NUMBER":Z=8:GOSUB17000
10140 :PRINT"REAL PORTION";X(I);"NUMBER":Z=10:GOSUB17000
10150 :PRINT"IMAGINARY PORTION";Y(I);"NUMBER":Z=23:GOSUB17
000:GOSUB17100
10160 NEXT I
10170 GOTO 14000
10200 REM
10201 REM *****
10202 REM *      OUTPUT LENGTH ANGLE  *
10203 REM *****
10204 REM
10207 FOR I= 1 TO N
10210 :PRINT"{CLR}":Z=4:GOSUB17000
10220 :PRINT"OUTPUT";I;"NUMBER":Z=8:GOSUB17000
10230 :PRINT"LENGTH";Z(I):Z=10:GOSUB17000
10240 :PRINT"ANGLE ";P(I):Z=10:GOSUB17000 :GOSUB 17100
10250 NEXT I
14000 :PRINT"{CLR}":Z=4:GOSUB 17000
14010 PRINT"MORE CALCULATIONS?":Z=8:GOSUB17000
14020 PRINT"(Y/N);:?"
14030 GET T$:IF T$="" THEN 14030
14040 IF T$="Y" THEN 10
14050 IF T$="N" THEN RETURN
15000 REM
15010 REM *****
15020 REM *      INPUT ROUTINE      *
15030 REM *****
15040 REM
15050 :PRINT"{CLR}":S=4: Z=4:GOSUB 17000
15060 PRINT"*****":Z=9:GOSUB
17000
15070 PRINT"*
17000 *":Z=10:GOSUB
15080 PRINT"*      CONVERTING COMPLEX NUMBERS *":Z=10:GOSUB
17000
15090 PRINT"*
*":Z=12:GOSUB

```

```

17000
15100 PRINT"*****":Z=9:GOSUB
17000
15110 FOR T=1 TO 1000:NEXT:PRINT"{CLR}":Z=4:GOSUB 17000
15120 INPUT"{WHT}OPERATION NUMBER:";N:Z=10:GOSUB17000
15130 PRINT"{GRY2}INPUT METHOD:";Z=12:GOSUB17000
15140 PRINT"{GRY2}LENGTH & ANGLE: 1":Z=14:GOSUB17000
15150 PRINT"{GRY2}REAL/IMAGINARY PARTS: 2":Z=18:GOSUB17000
15160 INPUT"{WHT}SELECT 1-2 ";M:PRINT"{GRY2}"
15170 IF M<1 OR M>2 THEN 15130
15180 IF M=1 THEN 15200
15190 IF M=2 THEN 15280
15200 REM
15201 REM *****
15202 REM * INPUT LENGTH - ANGLE *
15203 REM *****
15204 REM
15207 FOR I= 1 TO N
15210 :PRINT"{CLR}":Z=4:GOSUB 17000
15220 :PRINT"INPUT";I;"NUMBER":Z=8:GOSUB 17000
15230 :INPUT"{WHT}LENGTH";Z(I):Z=10:GOSUB 17000
15240 :INPUT"ANGLE";P(I):PRINT"{PURP}"
15250 NEXT I
15260 GOSUB 1100
15270 RETURN
15280 FOR I =1 TO N
15290 PRINT"{CLR}":Z=4:GOSUB 17000
15300 REM
15301 REM *****
15302 REM * INPUT REAL IMAGINARY*
15303 REM *****
15304 REM
15307 FOR I= 1 TO N
15310 :PRINT"{CLR}":Z=4:GOSUB 17000
15320 :PRINT"INPUT";I;"NUMBER":Z=8:GOSUB 17000
15330 :INPUT"{WHT}REAL PORTION";X(I):Z=10:GOSUB 17000
15340 :INPUT"IMAGINARY PEORTION";Y(I):PRINT"{PURP}"
15350 NEXT I
15360 GOSUB 1300
15370 RETURN
17000 REM
17010 REM *****
17020 REM * PROGRAM CONTROL ROUTINES *
17030 REM *****
17040 REM
17050 POKE211,S:POKE314,Z:SYS 58640:RETURN
17100 PRINT"{WHT}PRESS RETURN{GRY2}"
17110 GET T$:IF T$="" THEN 17110
17120 IF T$=CHR$(13) THEN RETURN

```

READY.

```

10 REM *****
20 REM *
30 REM *      CALCULATING      *
40 REM *
50 REM *      COMPLEX RESISTANCES      *
60 REM *
70 REM *****
80 REM
90 REM
100 POKE 53280,0:POKE53281,0:PRINT" (GRY2)"
500 GOSUB 15000
510 GOSUB10000
520 END
1000 REM
1010 REM *****
1020 REM * MATHEMATICAL ROUTINES      *
1030 REM *****
1040 REM
1100 REM
1101 REM *****
1102 REM *      CALCULATE R      *
1103 REM *****
1104 REM
1110 X=R
1120 Y=0
1130 RETURN
1200 REM
1201 REM *****
1202 REM *      CALCULATE L      *
1203 REM *****
1204 REM
1210 X=0
1220 Y=2*PI*F*L/1000
1230 RETURN
1300 REM
1301 REM *****
1302 REM *      CALCULATE C      *
1303 REM *****
1304 REM
1310 X=0
1320 Y=(1000000/(2*PI*F*C))
1330 RETURN
1400 REM
1401 REM *****
1402 REM *      CALCULATE R+L      *

```



```

1403 REM *****
1404 REM
1410 X=R
1420 Y=Y
1430 RETURN
1500 REM
1501 REM *****
1502 REM *          CALCULATE R // C          *
1503 REM *****
1504 REM
1510 X=R
1520 A=1/X
1530 B=1/Y
1540 NE=A*A+B*B
1550 X=A/NE
1560 Y=B/NE
1570 RETURN
10000 REM
10010 REM *****
10020 REM *          OUTPUT ROUTINES          *
10030 REM *****
10040 REM
10100 PRINT "{CLR}":S=4:Z=4:GOSUB 17000
10110 PRINT"COMPLEX RESISTANCE IN OHMS:":Z=8:GOSUB17000
10120 PRINT"REAL PART:":X:Z=10:GOSUB 17000
10130 PRINT"IMAGINARY PART:":Y:Z=23:GOSUB 17000:GOSUB17100
14000 PRINT "{CLR}":S=4:Z=4:GOSUB 17000
14010 PRINT"MORE CALCULATIONS?":Z=8:GOSUB 17000
14020 PRINT"(Y/N)":PRINT"{GRY23}"
14030 GET T$: IF T$="" THEN 14030
14040 IF T$="N" THEN RETURN
14050 IF T$="Y" THEN 10
15000 REM
15010 REM *****
15020 REM *          INPUT ROUTINES          *
15030 REM *****
15040 REM
15050 PRINT "{CLR}":S=4:Z=8:GOSUB 17000
15060 PRINT"*****":Z= 9:GOSUB17000
15070 PRINT"*":Z=10:GOSUB17000
15080 PRINT"*          CALCULATING          *":Z=11:GOSUB17000
15090 PRINT"*":Z=12:GOSUB17000
15100 PRINT"*    COMPLEX RESISTANCES    *":Z=13:GOSUB17000
15110 PRINT"*":Z=14:GOSUB17000
15120 PRINT"*****"
15130 FOR T= 1 TO 1000:NEXT:PRINT "{CLR}"
15140 Z=4:GOSUB 17000
15150 PRINT"SELECT RESISTANCE FORM:"

```

```

15160 PRINT
15170 PRINT
15180 PRINT"
15190 PRINT"  -----|----- R : 1 -----|-----"
15200 PRINT"  -----|----- |-----|-----"
15210 PRINT"  -----|----- |-----"
15220 PRINT"  -----{RVON} {RVOF}----- L : 2 -----|-----
|-----|-----"
15230 PRINT
15240 PRINT
15250 PRINT"  -----|----- C : 3 R || C : 4 "
15270 PRINT
15280 PRINT
15290 PRINT"
15300 PRINT"  -----|-----{RVON} {RVOF}-----
R + L : 5 "
15310 PRINT"  ----- ":Z=23:GOSUB17000
15320 INPUT"SELECT 1 - 5 ";N:PRINT"{GRY2}"
15330 ON N GOSUB 15400,15500,15600,15700,15800
15340 RETURN
15350 REM
15360 REM *****
15370 REM * INPUT RESISTANCE VALUES *
15380 REM *****
15390 REM
15400 REM
15401 REM *****
15402 REM * INPUTS *
15403 REM *****
15404 REM
15407 PRINT"{CLR}":Z=4:GOSUB 17000
15410 PRINT"RESISTANCE IN OHMS":Z=6:GOSUB 17000
15420 INPUT"{WHT}R= ";R:PRINT"{GRY2}"
15430 GOSUB1100
15440 REM
15500 REM
15501 REM *****
15502 REM * INPUTS L *
15503 REM *****
15504 REM
15507 PRINT"{CLR}":Z=4:GOSUB 17000
15510 PRINT"INDUCTION IN MILLIHENRIES":Z=6:GOSUB 17000
15520 INPUT"{WHT}L= ";L:Z=10: GOSUB 17000
15530 PRINT"{GRY2}FREQUENCY IN HERTZ":Z=12:GOSUB 17000
15540 INPUT"{WHT}F= ";F:PRINT"{GRY2}"
15550 GOSUB 1200
15560 RETURN
15600 REM

```

```

15601 REM *****
15602 REM *          INPUTS      C          *
15603 REM *****
15604 REM
15607 PRINT "{CLR}":Z=4:GOSUB 17000
15610 PRINT"CAPACITANCE IN MICROFARADS":Z=6:GOSUB 17000
15620 INPUT"{WHT}C= ";C:Z=10:GOSUB17000
15630 PRINT"{GRY2}FREQUENCY IN HERTZ":Z=12:GOSUB 17000
15640 INPUT"{WHT}F= ";F:PRINT
15650 GOSUB 1300
15660 RETURN
15700 REM
15701 REM *****
15702 REM *          INPUTS      R / / C      *
15703 REM *****
15704 REM
15707 PRINT "{CLR}":Z=4:GOSUB 17000
15710 PRINT"RESISTANCE IN OHMS ":Z=6:GOSUB 17000
15720 INPUT"{WHT}R= ";R:Z=10:GOSUB17000
15730 PRINT"{GRY2}CAPACITY IN MICROFARRADS":Z=12:GOSUB 17
000
15740 INPUT"{WHT}C= ";C:PRINT
15750 PRINT"{GRY2}FREQUENCY IN HERTZ":Z=18:GOSUB 17000
15760 INPUT"{WHT}F= ";F:PRINT
15770 GOSUB 1300
15780 GOSUB 1500
15790 RETURN
15800 REM
15801 REM *****
15802 REM *          INPUTS      R + L      *
15803 REM *****
15804 REM
15807 PRINT "{CLR}":Z=4:GOSUB 17000
15810 PRINT"RESISTANCE IN OHMS ":Z=6:GOSUB 17000
15820 INPUT"{WHT}R= ";R:Z=10:GOSUB17000
15830 PRINT"{GRY2}INDUCTION IN MILLIHENRIES":Z=12:GOSUB 1
7000
15840 INPUT"{WHT}L= ";L:Z=16:GOSUB17000
15850 PRINT"{GRY2}FREQUENCY IN HERTZ":Z=18:GOSUB 17000
15860 INPUT"{WHT}F= ";F:PRINT
15870 GOSUB 1200
15880 GOSUB 1400
15890 RETURN
17000 REM
17001 REM *****
17002 REM * PROGRAM CONTROL ROUTINES *
17003 REM *****
17004 REM

17050 POKE211,S:POKE214,Z:SYS 58640:RETURN
17100 PRINT"{WHT}PRESS RETURN{GRY2}";
17110 GET T$: IF T$= "" THEN 17110
17120 IF T$=CHR$(13) THEN RETURN

```

10.3.2 Current and potential sources

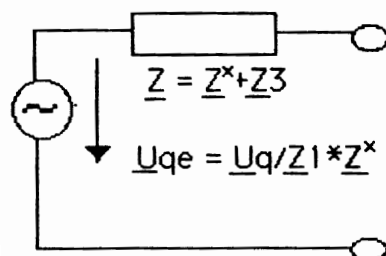
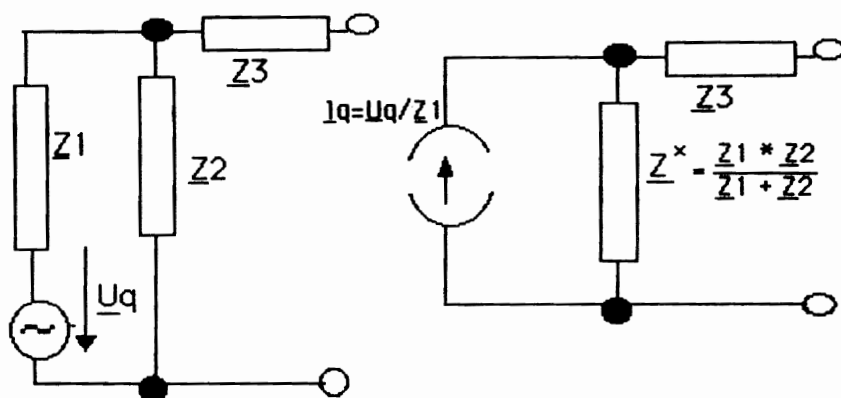
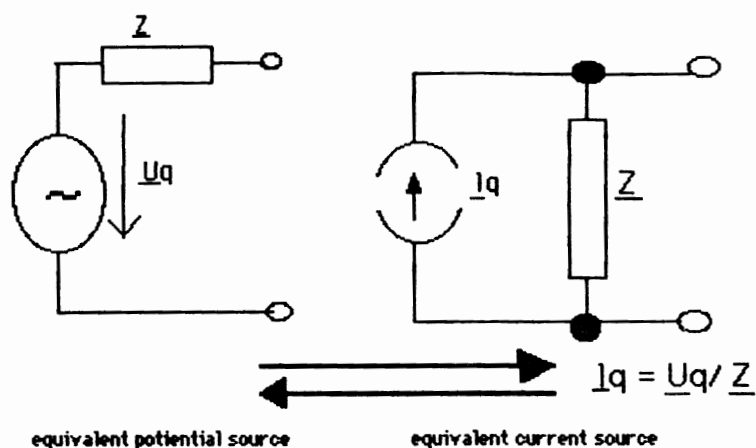
Before we come to the calculation of complex networks, a couple of important matters must first be clarified: The conversion from current sources to potential sources and vice versa. Not exactly earth-shaking, and it isn't worth it to grind out a program for it, but network analysis and node potential analysis cannot be dealt with successfully without these considerations. In addition, this will help the beginner to get an idea as to the tools used and hopefully encourage him to seek more detailed technical literature. It is quite reasonable that misunderstandings may easily occur through the inevitable shortness of these discussions.

The analysis procedures in the following sections both illustrate that the calculations can only be successfully carried out if the network contains only one type of energy sources. Network current analysis--potential sources; node potential analysis--current sources.

We want to take a look at the simplest case here. A current source with parallel resistance can be converted to an ersatz potential source with series resistance. The same is true in reverse--a potential source (resistance in series) can be converted to an ersatz current source (parallel resistance). Circuit diagrams and conversion formulas can be found in figure 10.3.21.

It should be noted that through more skillful and reiterative use of these operations, sources can be shifted and networks reduced to the essentials. This is clarified in figure 10.3.22. The potential source U_q is replaced by an ersatz current source I_q which combines the complex

resistances Z_1 and Z_2 in an ersatz resistance Z . The the current source is transformed back to a potential source U_{qe} and the resistance Z is calculated from the series circuit of the ersatz resistance and R_3 .



relation through
equivalent source
methods

10.3.3 Network current analysis

Theoretical foundations:

Network current analysis is suited to the calculation of branch currents in an arbitrary network which is constructed from only complex resistances and complex potential sources. If complex current sources are also found in the circuit, these are converted to complex ersatz potential sources already mentioned. The general procedure will be explained using a sample network (figure 10.3.31).

1. Determine the number of nodes (N) and branches (B) of the network (figure 10.3.32).

2. Determine a tree with N-1 branches such that all of the nodes are directly or indirectly connected and the tree does not form a closed network. Make sure that currents to be calculated in the independent connection trees do not flow into the tree branches. Possibilities for the tree selection are shown in figure 10.3.33.

3. Now select B-(N-1) networks which each contain a connection tree (otherwise only a tree branch), and choose a network circulation in the direction of the connection branch current.

4. Network equations:

$$\begin{array}{llll}
 (B1 + B4 + B6) \cdot I1 & -B4 & \cdot I2 & -B6 & \cdot I3 = Uq \\
 -B4 & \cdot I1 & + (B2 + B4 + B5) \cdot I2 & -B5 & \cdot I3 = 0 \\
 -B6 & \cdot I1 & -Z5 & \cdot I2 & + (Z3 + Z5 + Z6) \cdot I3 = 0
 \end{array}$$

The main diagonal of the complex resistance matrix consists of the complex sum resistances of the selected network. The other diagonals contain the complex resistances which flow from at least two network currents. If the network currents traverse the branch of the resistance in the right direction, the resistance has a positive sign, otherwise negative. The potential has a negative sign for the proper direction in the network current, otherwise positive.

5. The equation system can be solved using Cramer's rule. You get the values for the network currents.

Application and explanation of the program:

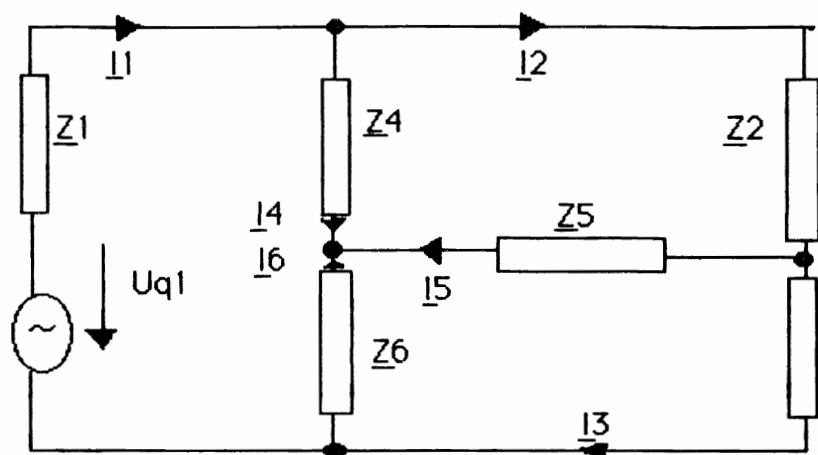
When using this program, go through the first three points above. The program then gives you the necessary number of networks based on the number of nodes and branches.

Enter the values (resistances and potentials) for each network in order. The connection branch will be asked for first and then the individual tree branches.

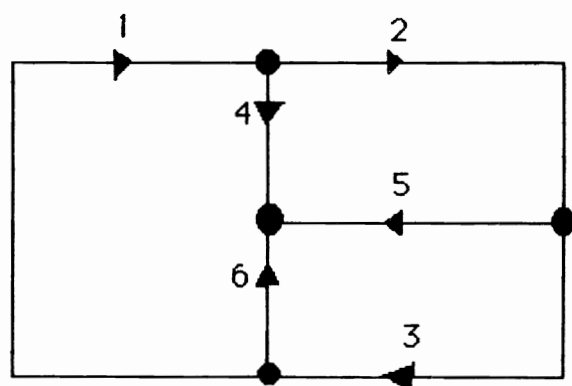
The resistance matrix is set up in the program section "mathematical routines." Next follows the potential vector. The calculation is carried out according to Cramer's rule. First the determinant of the resistance matrix is calculated, then the calculations are continued with the potential vector inserted. The values for the individual currents are produced from the division of the values of the resistance matrix by the inserted potential vector and the complete conductance matrix.

1100 set up the resistance matrix
1400 set up the potential vector
1500 insert the potential vector
1700 calculate the network current
2000 store the matrix values
2200 calculate the determinant
10000 output the network currents
15200 input the network values (nodes, branches)
15300 input the Z and U values
16000 sampel network

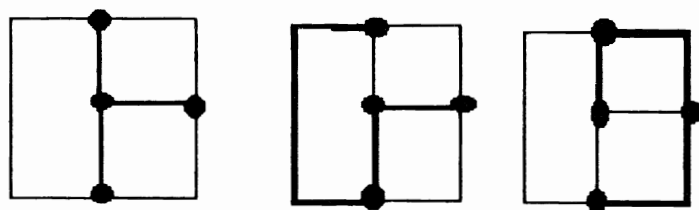
It should also be mentioned that you can interfere in the program section "mathematical routines" to produce interesting values. An example is the complex resistance matrix or its value.



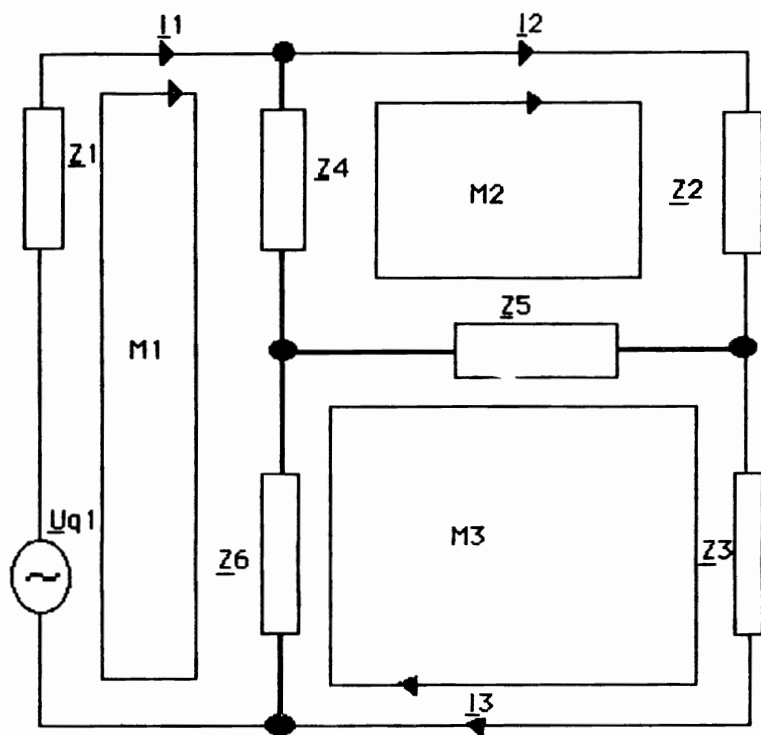
simple network



graph of circuit



three possible circuit trees



Circuit with tree, networks, and network current

```
10 REM *****
20 REM *
30 REM *
40 REM *NETWORK CURRENT ANALYSIS *
50 REM *
60 REM *
70 REM *****
80 REM
90 REM
100 POKE 53280,0:POKE 53281,0:PRINT"(GRY2)"
500 GOSUB 15000
510 GOSUB 10000
520 END
1000 REM
1010 REM *****
1020 REM * MATHEMITACAL ROUTINES *
1030 REM *****
1040 REM
1050 REM
1060 REM *****
1070 REM * CALCULATE THE Z-MATRIX *
1080 REM *****
1090 REM
1100 FOR I=1 TO MA
1110 :FOR J=1 TO MA
1120 ::ZX(J,I)=ZX(I,J)
1130 ::ZY(J,I)=ZY(I,J)
1135 ::V(J,I)=V(I,J)
1140 :NEXT J
1150 NEXT I
1160 FOR I=1 TO MA
1170 :FOR J=1 TO MA
1180 ::IF J=I THEN 1210
1190 ::ZX(I,I)=ZX(I,I)+ZX(I,J)
1200 ::ZY(I,I)=ZY(I,I)+ZY(I,J)
1210 :NEXT J
1220 NEXT I
1230 FOR I=1 TO MA
1240 :FOR J=1 TO MA
1250 ::IF J=I THEN 1280
1260 ::ZX(I,J)=ZX(I,J)*V(I,J)
1270 ::ZY(I,J)=ZY(I,J)*V(I,J)
1280 :NEXT J
1290 NEXT I
1300 GOSUB 2000
```

```
1310 GOSUB 2200
1320 BX=AX:BY=AY
1380 REM
1390 REM *****
1400 REM * CALCULATE THE U-VECTORS *
1410 REM *****
1420 REM
1430 FOR I=1 TO MA
1440 :FOR J=1 TO MA
1450 ::IF J=I THEN 1480
1460 ::UX(I,I)=UX(I,I)+UX(I,J)
1470 ::UY(I,I)=UY(I,I)+UY(I,J)
1480 :NEXT J
1490 NEXT I
1500 REM
1510 REM *****
1520 REM * SUBSTITUTITE THE U-VECTORS *
1530 REM *****
1540 REM
1550 FOR M=1 TO MA
1560 :GOSUB 2000
1570 :FOR I=1 TO MA
1580 ::LX(I,M)=UX(I,I)
1590 ::LY(I,M)=UY(I,I)
1600 :NEXT I
1610 :GOSUB 2200
1620 :AX(M)=AX:AY(M)=AY
1630 NEXT M
1680 REM
1690 REM *****
1700 REM * CALCULATE NETWORK CURRENT*
1710 REM *****
1720 REM
1730 FOR M=1 TO MA
1740 :NE=BX*BX+BY*BY
1745 :IF NE=0 THEN 1765
1750 :IX(M)=(AX(M)*BX+AY(M)*BY)/NE
1760 :IY(M)=(AY(M)*BX-AX(M)*BY)/NE:GOTO 1770
1765 :IX(M)=0:IY(M)=0
1770 NEXT M
1780 RETURN
2000 REM
2010 REM *****
2020 REM * STORE THE VALUES *
2030 REM *****
2040 REM
2050 FOR I=1 TO MA
2060 :FOR J=1 TO MA
```

```

2070 ::LX(I,J)=ZX(I,J)
2080 ::LY(I,J)=ZY(I,J)
2090 :NEXT J
2100 NEXT I
2110 RETURN
2200 REM
2210 REM *****
2220 REM * CALCULATE DETERMINATES *
2230 REM *****
2240 REM
2250 FOR J=1 TO MA-1
2260 :FOR I=J+1 TO MA
2270 ::NR=LX(J,J)*LX(J,J)+LY(J,J)*LY(J,J)
2275 ::IF NR=0 THEN 2295
2280 ::MX=(LX(I,J)*LX(J,J)+LY(I,J)*LY(J,J))/NR
2290 ::MY=(LX(J,J)*LY(I,J)-LX(I,J)*LY(J,J))/NR:GOTO 2300
2295 ::MX=0:MY=0
2300 ::FOR K=J+1 TO MA
2310 ::LX(I,K)=LX(I,K)-(MX*LX(J,K)-MY*LY(J,K))
2320 ::LY(I,K)=LY(I,K)-(MX*LY(J,K)+MY*LX(J,K))
2330 ::NEXT K
2340 :NEXT I
2350 NEXT J
2360 AX=LX(1,1):AY=LY(1,1)
2370 FOR I=2 TO MA
2380 :CX=AX*LX(I,1)-AY*LY(I,1)
2390 :CY=AX*LY(I,1)+AY*LX(I,1)
2400 :AX=CX:AY=CY
2410 NEXT I
2420 RETURN
10000 REM
10010 REM *****
10020 REM * OUTPUT ROUTINE *
10030 REM *****
10040 REM
10050 FOR M=1 TO MA
10060 :PRINT"{CLR}":S=4:Z=4:GOSUB 17000
10070 :PRINT"NETWORK CURRENT"M"IN AMPERES:":Z=8:GOSUB 17000
10080 :PRINT"IX= "IX(M):Z=10:GOSUB 17000
10090 :PRINT" IY= "IY(M):Z=23:GOSUB 17000:GOSUB 17100
10100 NEXT M
14000 PRINT"{CLR}":S=4:Z=4:GOSUB 17000
14010 PRINT"MORE CALCULATIONS?":Z=8:GOSUB 17000
14020 PRINT"Y/N";:PRINT"{GRY2}"
14030 GET T$:IF T$=""THEN 14030
14040 IF T$="N"THEN RETURN
14050 IF T$="J"THEN 10

```

```

15000 REM
15010 REM *****
15020 REM *      INPUT      ROUTINE      *
15030 REM *****
15040 REM
15050 PRINT "{CLR}":S=6:Z=8:GOSUB 17000
15060 PRINT "*****":Z=9:GOSUB 17000
15070 PRINT "*" *":Z=10:GOSUB 17000
15080 PRINT "* NETWORK CURRENT ANALYSIS*":Z=11:GOSUB 17000
15090 PRINT "*" *":Z=12:GOSUB 17000
15100 PRINT "*****"
15110 FOR T=1 TO 5000:NEXT:PRINT "{CLR}"
15120 S=4:Z=4:GOSUB 17000
15130 PRINT "WOULD YOU LIKE AN EXAMPLE NETWORK?":Z=8:GOSUB
17000
15140 PRINT "{WHT}Y/N":PRINT "{GRY2}"
15150 GET T$:IF T$=""THEN 15150
15160 IF T$="N"THEN 15180
15170 IF T$="Y"THEN GOSUB 16000
15180 REM
15190 REM *****
15200 REM *      NETWORK      -      VALUES      *
15210 REM *****
15220 REM
15230 PRINT "{CLR}":S=4:Z=4:GOSUB 17000
15240 PRINT "NETWORK - VALUES!":Z=8:GOSUB 17000
15250 INPUT "{WHT}NUMBER OF NODES :":KN:Z=10:GOSUB 17000
15260 INPUT "NUMBER OF BRANCHES":ZW:PRINT "{GRY2}":S=3:Z=14:
GOSUB 17000
15270 MA=ZW-(KN-1)
15280 PRINT MA;" NETWORKS WILL BE REQUIRED":S=4:Z=23:GOSUB
17000:GOSUB 17100
15290 REM
15300 REM *****
15310 REM *      INPUTS      Z,U - VALUES*
15320 REM *****
15330 REM
15350 FOR I=1 TO MA
15360 :PRINT "{CLR}":S=3:Z=4:GOSUB 17000
15370 :PRINT "INPUT VALUES FOR THE NETWORK:":I:Z=8:GOSUB 17
000
15380 :PRINT "JUNCTION BRANCH RESISTANCE IN OHMS":Z=10:GOSU
B 17000
15390 :PRINT "FLOWING ONLY FROM NETWORK CURRENT {WHT}I":I:Z
=12:GOSUB 17000
15400 :INPUT "{WHT}ZX= ":ZX(I,I):Z=14:GOSUB 17000
15410 :INPUT "ZY= ":ZY(I,I):Z=18:GOSUB 17000
15420 :PRINT "{GRY2}JUNCTION BRANCH POTENTIAL IN VOLTS":Z=

```

```

20:GOSUB 17000
15430 : INPUT "{WHT}UX= "; UX(I,I):Z=22:GOSUB 17000
15440 : INPUT "UY= "; UY(I,I):PRINT "{GRY2}"
15450 : FOR J=I+1 TO MA:IF J>MA THEN 15550
15460 ::PRINT "{CLR}":S=3:Z=4:GOSUB 17000
15470 ::PRINT "INPUT VALUES FOR THE NETWORK:";I:Z=8:GOSUB 1
7000
15480 ::PRINT "TRUE BRANCH RESISTANCE IN OHMS":Z=10:GOSUB 1
7000
15490 ::PRINT "FLOWING THRU THE NETWORK CURRENT {WHT}I";J;:
Z=12:GOSUB 17000
15500 :: INPUT "{WHT}ZX= "; ZX(I,J):Z=14:GOSUB 17000
15510 :: INPUT "ZY= "; ZY(I,J):S=26:Z=12:GOSUB 17000
15511 : PRINT "{GRY2}DIRECTION I";J:Z=13:GOSUB 17000
15512 : PRINT "WITH I";I;" (+)":Z=14:GOSUB 17000
15513 : PRINT "AGAINST";I;" (-)":S=35:Z=15:GOSUB 17000
15514 : INPUT "{WHT}";V$:PRINT "{GRY2}"
15516 : IF V$=CHR$(43) THEN V(I,J)=1
15517 : IF V$=CHR$(45) THEN V(I,J)=-1
15518 : S=3:Z=18:GOSUB 17000
15520 ::PRINT "{GRY2}TRUE BRANCH POTENTIAL IN VOLTS":Z=20:G
OSUB 17000
15530 :: INPUT "{WHT}UX= "; UX(I,J):Z=22:GOSUB 17000
15540 :: INPUT "UY= "; UY(I,J):PRINT "{GRY2}"
15550 UX(J,I)=-UX(I,J)
15560 UY(J,I)=-UY(I,J)
15570 :NEXT J
15580 NEXT I
15590 GOSUB 1000
15600 RETURN
16000 REM
16010 REM *****
16020 REM * SAMPLE NETWORK *
16030 REM *****
16040 REM
16050 PRINT "{CLR}"
16060 PRINT "
|
| "
16070 PRINT " | {RED} | {GRY2}
| "
16080 PRINT " | r>— — r {RED} | {GRY2} r>— — —
— r | "
16090 PRINT " | | | | | "
16100 PRINT " |Z| | | |Z| | M2 | |Z| "
16110 PRINT " | | | | | "
16120 PRINT " | | | {RED} | {GRY2} | — — — —
—<— | "
16130 PRINT " | {RED} | {GRY2}

```



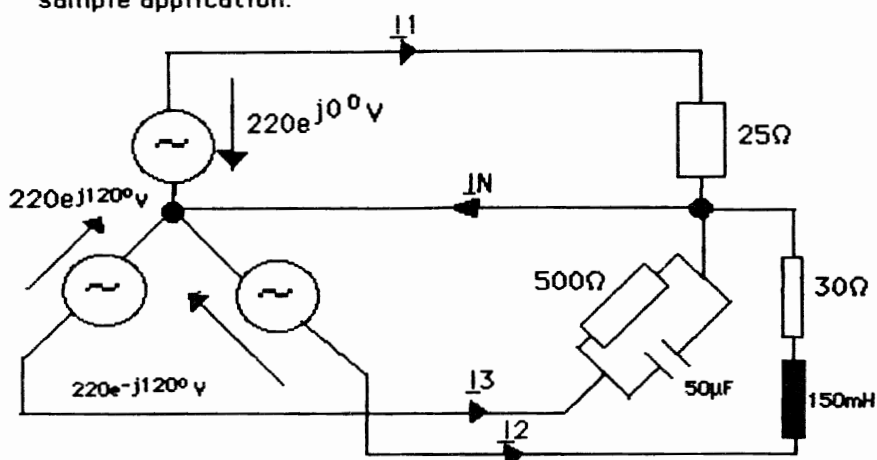
```

      | "
16140 PRINT"      | | M1 | {RED} |-----{GRY2}| Z |{R
ED}-----{GRY2}| "
16150 PRINT"      | {RED} | {GRY2}
      | "
16160 PRINT"      | | {RED} | {GRY2} |>-----
      | "
16170 PRINT"      [ ] "
16180 PRINT"      [U] | | [Z] | M3 | [Z] "
16190 PRINT"      [ ] "
16200 PRINT"      | L ---<| {RED} | {GRY2} L ---
<| | "
16210 PRINT"      | {RED} | {GRY2}
      | "
16220 PRINT"      L-----
      | ":PRINT
16230 PRINT"      4 NODES (N) : 6 BRANCHES(B)" REM *****
*****
16240 PRINT"      B-(N-1)= M - NETWORK"16050 PRINT"{CLR}"
16250 PRINT"      CIRCUIT TREE (RED)":S=5:Z=23:GOS
UB 17000:GOSUB 17100
16260 RETURN
17000 REM
17010 REM *****
17020 REM * PROGRAM CONTROL ROUTINE *
17030 REM *****
17040 REM
17050 POKE 211,S:POKE 214,Z:SYS 58640:RETURN
17100 PRINT"(WHT)PRESS RETURN{GRY2}";
17110 GET T$:IF T$="" THEN 17110
17120 IF T$=CHR$(13) THEN RETURN

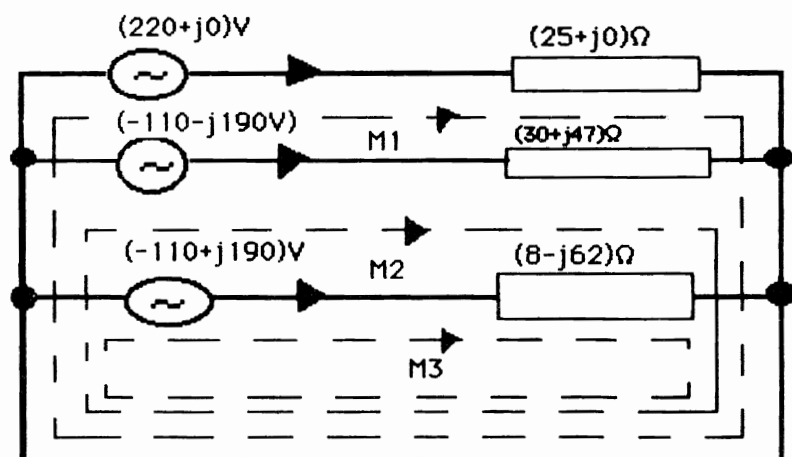
```

READY.

sample application:



Three phase network, Stern - Stern



$$I_1 = (8,80 - j0) \text{ A}$$

$$I_2 = (-3,93 - j0,17) \text{ A}$$

$$I_3 = (-3,24 - j1,36) \text{ A}$$

10.3.4 Node potential analysis

Theoretical foundations:

In contrast to the previously presented network current analysis, here we are dealing with a method of analysis which can be used in particular with networks which contain current sources. This procedure is best suited for analysis of transistor circuits if the transistors are replaced with a substitute circuit diagram which contains their relationships in the form of controlled current sources. If the network should contain potential sources, they can be converted with the help of the substitute current sources as explained earlier.

As far as the actual procedure, it should be said that knowledge of node potentials suffices for complete description of the network because all other potentials and currents can be represented through these potentials. An example should clarify the principle operations (see figure 10.3.41).

1. Select a node of the network as the reference node. A node should be chosen which will prove to be the most efficient for subsequent calculations of the necessary potentials.
2. All nodes are numbered (1,2,3,...). It is important for the program that the reference node have the highest number.
3. Record a potential arrow from each node to the reference node. These are the potentials to be calculated.

4. Set up the network equations and express all potentials which drop over a resistance not connected to the reference node using node potentials (see figure 10.3.42).

You get equations of the following type:

$$\begin{aligned}(U14 - U34)*Y4 + (U14 - U24)*Y3 + (U14 - U44)*Y1 &= Iq1 \\(U24 - U34)*Y5 + (U24 - U14)*Y3 + (U24 - U44)*Y2 &= Iq2 \\(U34 - U14)*Y4 + (U34 - U24)*Y5 + (U34 - U44)*Y6 &= 0\end{aligned}$$

5. Transform the equations so that you receive the following system: (remember that the potential $U44 = 0$)

$$\begin{aligned}(Y1 + Y3 + Y4)*U14 - Y3 & \quad *U24 - Y4 & \quad *U34 &= Iq1 \\-Y3 & \quad *U14 + (Y2 + Y3 + Y5)*U24 - Y5 & \quad *U34 &= Iq2 \\-Y4 & \quad *U14 - Y5 & \quad *U24 + (Y4 + Y5 + Y6)*U34 &= 0\end{aligned}$$

6. This system of equations can be solved according to Cramer's rule in which the solution vector (I-vector) is substituted for the sought-after potential in the conductance matrix, the determinant of this is taken and divided by the determinant of the original of the original matrix.

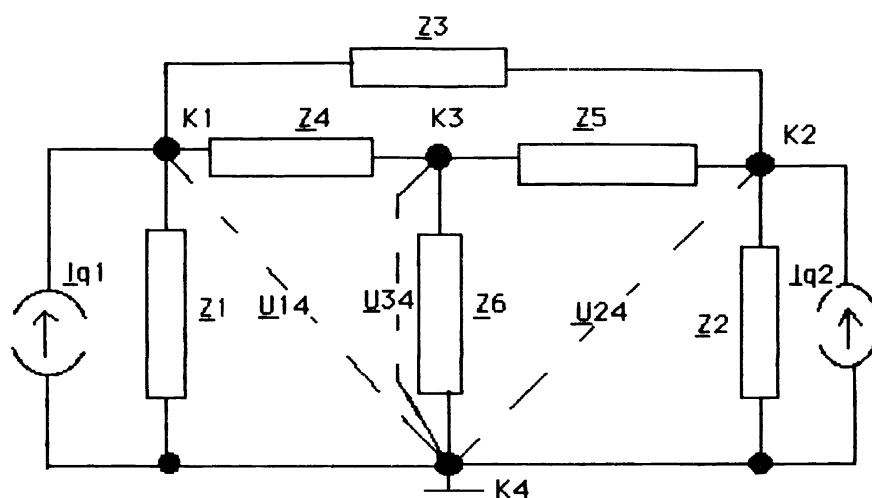
Applications and explanation of the program

As in calculation by hand, you must perform the first three steps. It is important that the reference node have the highest number. The program first asks for the number of nodes and then the reference node. This allows formation of the complex resistance matrix and the complex current vector (number of elements and input).

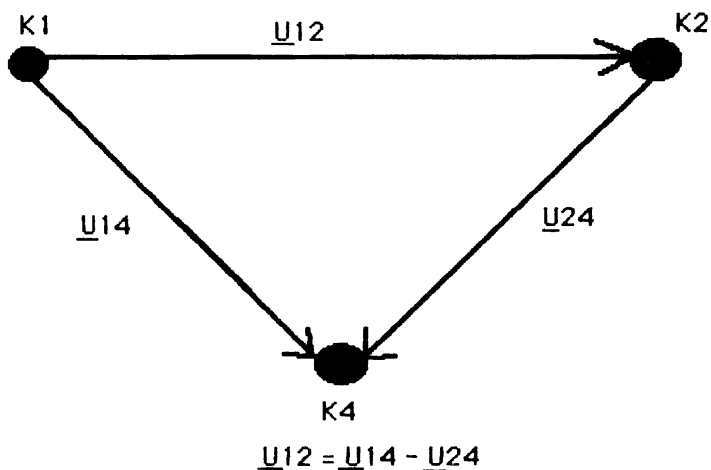
The values for the complex resistance and the complex current source of each branch are requested independent of the type or numbering given in the circuit diagram. The numbers of the nodes between which a branch is found are given in the first line. Z_x or I_x represents the real portion, Z_y or I_y the imaginary portion. If an element is not present, 0;0 must be entered here.

The resistance matrix is inverted in the program section "mathematical routines," creating the conductance matrix (Y). After this the current vector is inserted into the matrix for each complex potential to be calculated. It is necessary to first store the conductance matrix for this purpose since the calculation operations must be repeated according to the number of the potentials to be calculated (nodes-1) and the current vector is always shifted into the next column. The old conductance values must be retained.

```
1100 calculate of the Y matrix from the Z values
1400 calculate of the current vector
1500 insert the current vector
1700 calculate the node potentials
2000 store the conductance matrix temporarily
2200 calculate the determinants
10000 output the node potential
15200 input number of nodes
15300 input Z and I values
16000 sample network
```



Network with node potentials



Replacing a non-node potential with node potential

```
10 REM *****
20 REM *
30 REM *
40 REM *   NODE POTENTIAL ANALYSIS   *
50 REM *
60 REM *
70 REM *****
80 REM
90 REM
100 POKE 53280,0:POKE 53281,0:PRINT "{GRY2}"
500 GOSUB 15000
510 GOSUB 10000
520 END
1000 REM
1010 REM *****
1020 REM * MATHEMATICS      ROUTINES *
1030 REM *****
1040 REM
1050 REM
1060 REM *****
1070 REM * CALCULATE THE   Y-MATRIX *
1080 REM *****
1090 REM
1130 FOR I=1 TO N
1140 :FOR J=1 TO N
1150 ::IF J=I THEN 1190
1160 ::NR=ZX(I,J)*ZX(I,J)+ZY(I,J)*ZY(I,J)
1165 IF NR=0 THEN 1185
1170 ::YX(I,J)=ZX(I,J)/NR
1180 ::YY(I,J)=-ZY(I,J)/NR:GOTO 1190
1185 YX(I,J)=0:YY(I,J)=0
1190 :NEXT J
1200 NEXT I
1210 FOR I=1 TO N
1220 :FOR J=1 TO N
1230 ::IF J=I THEN 1260
1240 ::YX(I,I)=YX(I,I)+YX(I,J)
1250 ::YY(I,I)=YY(I,I)+YY(I,J)
1260 :NEXT J
1270 NEXT I
1280 FOR I=1 TO N
1290 :FOR J=1 TO N
1300 ::IF J=I THEN 1330
1310 ::YX(I,J)=-YX(I,J)
1320 ::YY(I,J)=-YY(I,J)
```

```
1330 :NEXT J
1340 NEXT I
1350 GOSUB 2000
1360 GOSUB 2200
1370 BX=AX:BY=AY
1380 REM
1390 REM *****
1400 REM * CALCULATE THE I-VECTORS *
1410 REM *****
1420 REM
1430 FOR I=1 TO N
1440 :FOR J=1 TO N
1450 ::IF J=I THEN 1480
1460 ::IX(I,I)=IX(I,I)+IX(I,J)
1470 ::IY(I,I)=IY(I,I)+IY(I,J)
1480 :NEXT J
1490 NEXT I
1500 REM
1510 REM *****
1520 REM * SUBSTITUTE FOR I-VECTORS *
1530 REM *****
1540 REM
1550 FOR M=1 TO N-1
1560 :GOSUB 2000
1570 :FOR I=1 TO N-1
1580 ::LX(I,M)=IX(I,I)
1590 ::LY(I,M)=IY(I,I)
1600 :NEXT I
1610 :GOSUB 2200
1620 :AX(M)=AX:AY(M)=AY
1630 NEXT M
1680 REM
1690 REM *****
1700 REM * CALC. NODE POTENTIALS *
1710 REM *****
1720 REM
1730 FOR M=1 TO N-1
1740 :NE=BX*BX+BY*BY
1745 IF NE=0 THEN 1765
1750 :UX(M)=(AX(M)*BX+AY(M)*BY)/NE
1760 :UY(M)=(AY(M)*BX-AX(M)*BY)/NE:GOTO 1770
1765 UX(M)=0:UY(M)=0
1770 NEXT M
1780 RETURN
2000 REM
2010 REM *****
2020 REM * STORE THE Y VALUES *
2030 REM *****
```



```

2040 REM
2050 FOR I=1 TO N-1
2060 :FOR J=1 TO N-1
2070 ::LX(I,J)=YX(I,J)
2080 ::LY(I,J)=YY(I,J)
2090 :NEXT J
2100 NEXT I
2110 RETURN
2200 REM
2210 REM *****
2220 REM *CALCULATE DETERMINATES *
2230 REM *****
2240 REM
2250 FOR J=1 TO N-2
2260 :FOR I=J+1 TO N-1
2270 ::NR=LX(J,J)*LX(J,J)+LY(J,J)*LY(J,J)
2275 ::IF NR=0 THEN 2295
2280 ::MX=(LX(I,J)*LX(J,J)+LY(I,J)*LY(J,J))/NR
2290 ::MY=(LX(J,J)*LY(I,J)-LX(I,J)*LY(J,J))/NR:GOTO 2300
2295 ::MX=0:MY=0
2300 ::FOR K=J+1 TO N-1
2310 ::LX(I,K)=LX(I,K)-(MX*LX(J,K)-MY*LY(J,K))
2320 ::LY(I,K)=LY(I,K)-(MX*LY(J,K)+MY*LX(J,K))
2330 ::NEXT K
2340 :NEXT I
2350 NEXT J
2360 AX=LX(1,1):AY=LY(1,1)
2370 FOR I=2 TO N-1
2380 :CX=AX*LX(I,1)-AY*LY(I,1)
2390 :CY=AX*LY(I,1)+AY*LX(I,1)
2400 :AX=CX:AY=CY
2410 NEXT I
2420 RETURN
10000 REM
10010 REM *****
10020 REM * OUTPUT ROUTINE *
10030 REM *****
10040 REM
10050 FOR M=1 TO N-1
10060 :PRINT "{CLR}":S=4:Z=4:GOSUB 17000
10070 :PRINT"NODE POTENTIAL"M-"N"IN VOLTS:"Z=8:GOSUB 17000
10080 :PRINT"UX= "UX(M):Z=10:GOSUB 17000
10090 :PRINT"UY= "UY(M):Z=23:GOSUB 17000:GOSUB 17100
10100 NEXT M
14000 PRINT"{CLR}":S=4:Z=4:GOSUB 17000
14010 PRINT"MORE CALCULATIONS?":Z=8:GOSUB 17000
14020 PRINT"{WHT}Y/N";:PRINT"{GRY2}"

```

```

14030 GET T$:IF T$=""THEN 14030
14040 IF T$="N"THEN RETURN
14050 IF T$="Y"THEN 10
15000 REM
15010 REM *****
15020 REM *      INPUT      ROUTINE      *
15030 REM *****
15040 REM
15050 PRINT"{CLR}":S=6:Z=8:GOSUB 17000
15060 PRINT"*****":Z= 9:GOSUB 17000
15070 PRINT"*" *":Z=10:GOSUB 17000
15080 PRINT"* NODE POTENTIAL ANALYSIS *":Z=11:GOSUB 17000
15090 PRINT"*" *":Z=12:GOSUB 17000
15100 PRINT"*****"
15110 FOR T=1 TO 5000:NEXT:PRINT"{CLR}"
15120 S=4:Z=4:GOSUB 17000
15130 PRINT"WOULD YOU LIKE A SAMPLE-NETWORK?":Z=8:GOSUB 17
000
15140 PRINT"{WHT}Y/N";:PRINT"{GRY2}"
15150 GET T$:IF T$=""THEN 15150
15160 IF T$="N"THEN 15180
15170 IF T$="Y"THEN GOSUB 16000
15180 REM
15190 REM *****
15200 REM *      NNETWORK - VALUES      *
15210 REM *****
15220 REM
15230 PRINT"{CLR}":S=4:Z=4:GOSUB 17000
15240 PRINT"NETWORK - VALUES!":Z=8:GOSUB 17000
15250 INPUT"{WHT}NUMBER OF NODES :";N:PRINT"{GRY2}"
15260 REM
15270 REM *****
15280 REM * INPUT THE      Z,I-VALUES*
15290 REM *****
15300 REM
15310 FOR I=1 TO (N-1)
15320 :FOR J=(I+1) TO N
15330 ::PRINT"{CLR}":S=4:Z=4:GOSUB 17000
15340 ::IF J=I THEN 15460
15350 ::PRINT"INUT VALUES BETWEEN NODES: ";I-";J:Z=8:GOSUB
17000
15360 ::PRINT"COMPLES RESISTANCE IN OHMS:":Z=10:GOSUB 17
000
15370 ::INPUT"{WHT}ZX= ";ZX(I,J):Z=12:GOSUB 17000
15380 ::INPUT"ZY= ";ZY(I,J):PRINT"{GRY2}":Z=16:GOSUB 17000
15390 ::ZX(J,I)=ZX(I,J)
15400 ::ZY(J,I)=ZY(I,J)
15410 ::PRINT"SOURCE CURRENT IN AMPERES:":Z=18:GOSUB 17

```

000

15420 :: INPUT "WHT" IX= " ; IX(I,J):Z=20:GOSUB 17000

15430 :: INPUT "IY= " ; IY(I,J):PRINT "GRY2"

15440 :: IX(J,I)=-IX(I,J)

15450 :: IY(J,I)=-IY(I,J)

15460 :NEXT J

15470 NEXT I

15480 GOSUB 1000

15490 RETURN

16000 REM

16010 REM *****

16020 REM * SAMPLE L - NETWORK *

16030 REM *****

16040 REM

16050 PRINT "CLR"

16060 PRINT "

16070 PRINT "

16080 PRINT "

16090 PRINT "

16100 PRINT "

16110 PRINT "

"

16120 PRINT " | | {BLUE} \ {GRY2} ----- {BLUE} / {GRY2

} | ----- {BLUE} / {GRY2} | | "

16130 PRINT " | | {BLUE} \ / {GRY2} | {BLUE}

/ {GRY2} | | "

16140 PRINT " | | {BLUE} | | {GRY2} | {BLUE}

| {GRY2} | | "

16150 PRINT " | | {BLUE} U14 | {GRY2} | {BLUE}

U24 {GRY2} | | "

16160 PRINT " | | {BLUE} | | {GRY2} | | {BLUE}

| {GRY2} | | "

16170 PRINT " I |Z| {BLUE} | | {GRY2} |Z| {BLUE}

| {GRY2} |Z| I "

16180 PRINT " | | {BLUE} | | {GRY2} | | {BLUE}

| {GRY2} | | "

16190 PRINT " | | {BLUE} | U34 {GRY2} | {BLUE}

| {GRY2} | | "

16200 PRINT " | | {BLUE} | | {GRY2} | {BLUE}

| {GRY2} | | "

16210 PRINT " | | {BLUE} | | {GRY2} | {BLUE}

| {GRY2} | | "

16220 PRINT " | | {BLUE} \ \ {GRY2} | 4 {BLUE}

\ {GRY2} | | "

16230 PRINT "

"

16240 PRINT " ----- ":PRINT

16260 PRINT " 4 NODE NETWORK "

```
16270 PRINT"      POTENTIALS TO BE CALCULATED (BLUE)"
16280 S=4:Z=23:GOSUB 17000:GOSUB 17100
17000 REM
17010 REM *****
17020 REM * PROGRAM CONTROL ROUTINES*
17030 REM *****
17040 REM
17050 POKE 211,S:POKE 214,Z:SYS 58640:RETURN
17100 PRINT"{WHT}WPRESS RETURN{GRY2}";
17110 GET T$:IF T$=""THEN 17110
17120 IF T$=CHR$(13)THEN RETURN
```

READY.

10.4 Measurement and control

The Commodore 64 is equipped with two analog to digital converters. These are normally used for games to convert analog signals from the paddles to digital signals in the range from 0 to 255.

Naturally there are also more significant uses of these interfaces. For example, one could connect a light sensitive resistance, a photo resistor to the paddle input and use it to measure light intensity, or a temperature-depended resistance, a thermistor, and use the computer as temperature measuring device.

The resolution of the measured values is limited to 256 values, which cannot be changed because of the word size of only 8 bits. Better measuring devices can be found that connect to the IEEE bus, such as those made by Hewlett Packard. These are not within the range of ordinary hobbyists.

Two pairs of paddles, or in our case four measuring devices, can be connected to the computer. The resistances are connected across pins 7 (+5V) and 5 (POT AY) or 9 (POT AX). We have taken a program from the Abacus book The Anatomy of the Commodore 64 which reads the values of the paddles. Unfortunately these values are not accessible from BASIC so we must use a small machine language program.

Lines 10 through 50 contain the necessary bytes which will be POKEd into the the appropriate memory with the help of line 70. The machine language program is called with SYS

53182. The values are made available in memory locations 830,831 (port 1) and 828,829 (port 2). The machine language program must be executed before each measurement. Once the program has been read into memory, it is not affected by NEW, which means that once the program up to line 70 has been executed, it can be deleted and a program to process the paddle values can be loaded.

The values for the fire buttons on the paddles are also read in, but they are not of interest to us here.

You can now create a table in which to place the bytes representing the light intensity values. Try to approximate the table values with a function and then program this so that you can see the light converted directly on the screen. One use of this set-up might be in the darkroom where you could measure the light reflected from the surfaces where the paper later goes, convert the value to the requisite exposure time and control the exposure time directly with the help of a relay connected to the computer. You can use the datasette motor control output (pin C3) as the control output for the relay. You will have to manage a fair amount of data in order to get the desired exposure time. Note also the time which is required to actually shut off the relay after the CLOSE is executed. If you cannot manage with this and know more about electronics, you should use the user port. Note the warning below!

Since the values for all of the enlargers and paper/developer combinations are different, you must calibrate the equipment yourself.

It is possible to measure the pH value optically. If you use a red light-emitting diode, for instance, and monitor a test tube with litmus indicator between this and the photo resistor in an exactly defined distance, the light intensity depends directly on the pH value of the solution. The litmus indicator is violet for a basic solution, that is, it lets little red light through, and red for an acidic solution, meaning that it lets more red light through. It should be said that we are dealing here with more of an estimate than a measurement.

This gives a relationship between the pH value and the resistance of the photo resistor. You must again make quantifications yourself.

In addition to these possibilities, you can measure any quantity which can be relayed by means of a potentiometer. With the help of a potentiometer-coupled barometer, a thermocouple, and a potentiometer-coupled hygrometer, it is possible to construct a computer controlled weather station which displays the weather conditions at regular intervals as determined by the TI\$ clock or the real-time clock in the CIA. See the section on time measurement for more on the real-time clock. You could also display the values graphically on the screen or printer. The only limit is imagination.

WARNING -- please be careful when working with the computer interfaces. A momentary voltage on the wrong pin could cause a good deal of damage. It is best to limit your interfacing to the joystick ports and the cassette interface. These are not as sensitive as the user port and the expansion port.

If you build your own measuring devices, note that the value range of the resistance that will be varied must be in the range 0 to 500K ohm in order to make use of the full resolution of 256 values.

Using the user port for input and output

The user port, whose pin layout is given in the user's guide on page 143, is a universal I/O interface with a total of eight data lines. The individual lines may be selected as input or output, that is, if you need four bits of input and four of output, you can divide these lines accordingly.

POKE 56579,15

switches four bits to input and 4 bits to output.

Approximately 10 mA of current can be drawn from each pin on the user port when used as output. This means that the load on the line must have a resistance of at least 500 ohms. This follows from Ohm's law concerning the relationship between voltage, current, and resistance.

WARNING - shorting a pin to ground can DESTROY certain computer interfaces!!

To connect a relay, for the lamp in an enlarger, for instance, you must place a switching transistor between the interface and the relay. Since a total of eight relays may be connected, an additional source of current is necessary. Don't forget to tie both ground lines together so that the transistor receives a potential against its emitter.

The resistance before the base of the transistor should be a few kilo ohms. Select the largest value which will still allow the interface to turn the transistor on to minimize current draw on the port.

In order to get some practice working with the relationships between the value in memory location 56577 and the potential conditions on the user port, and later for the control of 8 light-emitting diodes, get eight 470 ohm resistors and a socket for the user port. The circuit is explained in figure 10.6 a. It does not matter which ground pin you use.

Now you must enter

POKE 56579,255

in order to switch all of the lines to output. Every value which you now POKE into memory location 56577 will be represented as a binary number on the LEDs. The weak light of the LEDs is caused by the high resistance of 470 ohms. This resistance is, however, necessary to protect the computer interface.

With this knowledge you can now use your C64 as an A/D converter which reads analog values over the paddle port and outputs these as binary values over the user port.

We would like to repeat our warning one more time: If electronics is not your area of specialty, you should consider finding someone who knows more about electronics to help you with the interfaces.

An alternative to the light-emitting diodes is a screen representation of the bits with the following program which runs in connection with the machine language routine for reading the paddle values. If you only need the values from port 1, as is the case here, you can read these values directly from memory locations 54267 and 54298, without the machine language program.

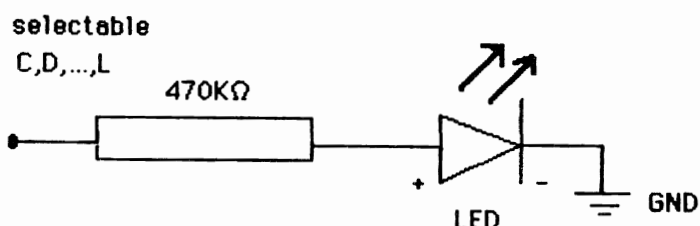


Fig. 10.6a

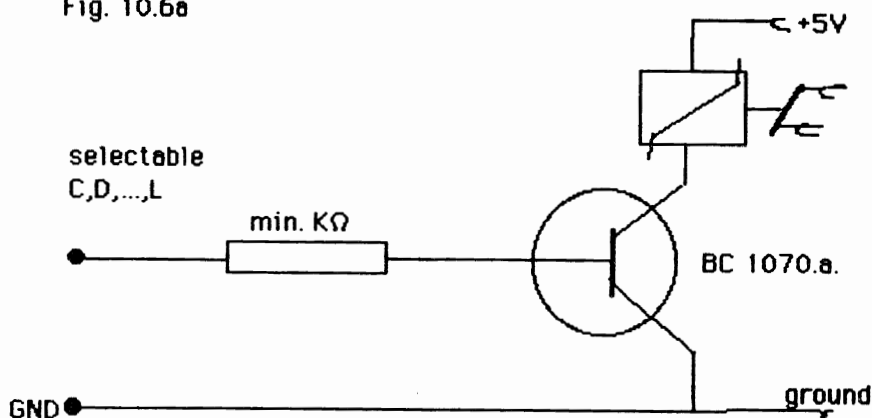


Fig. 10.6b

```
1000 REM BYTE REPRESENTATION ON SCREEN
1010 POKE 56579,255 : REM ALL LINES TO OUTPUT
1020 SYS 53182 : REM MACHINE LANGUAGE PROGRAM
1030 A=PEEK(830): REM READ THE VALUE FROM PADDLE 1 ON PORT 1
1040 POKE 56577,A : REM PLACE THE VALUE ON THE USER PORT
1050 X0=(A AND 1)=1
1060 X1=(A AND 2)=2
1070 X2=(A AND 4)=4
1080 X3=(A AND 8)=8
1090 X4=(A AND 16)=16
1100 X5=(A AND 32)=32
1110 X6=(A AND 64)=64
1120 X7=(A AND 128)=128
1130 PRINT"{home}";-X0;-X1;-X2;-X3;-X4;-X5;-X6;-X7
1140 GOTO 1020
```

If you want to run this program without the machine language program, you must make the following changes: Delete line 1020, replace the (830) in line 1030 with (54297) and change the jump address in line 1140 to 1030.

Naturally, it is unnecessary to POKE this value for the representation on the screen, but this allows devices connected later to react to the input signal as well.

To read a value, a potential of +5V as is found on the port itself must be present on the appropriate pin. This allows you to collect digital values or simply to read a set of external switches.

Don't forget to set the appropriate bits of memory location 56579 to 0 in order to switch the lines to input. The value on the port can be read with PEEK(56577).

```
4 :
5 REM *** READ 4 PADDLE VALUES      ***
6 :
10 DATA 120,169,128,32,236,207,142,60,3,140,61,3,173
20 DATA 0,220,41,12,141,159,2,169,64,32,236,207,142
30 DATA 62,3,140,63,3,173,1,220,41,12,141,160,2,169
40 DATA 255,141,2,220,88,96,141,0,220,9,192,141,2
50 DATA 220,162,0,202,208,253,174,25,212,172,26,212,96
60 FOR M=53182 TO 53247
70 READ A:POKE M,A:NEXT:REM READ MACHINE LANGUAGE PROGRAM
80 AX=830:REM PADDLE 1 IN CONTROLPORT 1
90 AY=831:REM PADDLE 2 IN CONTROLPORT 1
100 BA=832:REM BUTTON FROM PADDLE PORT A
110 BX=828:REM PADDLE 1 IN CONTROLPORT 2
120 BY=829:REM PADDLE 2 IN CONTROLPORT 2
130 BB=830:REM BUTTON FROM PADDLE PORT B
135 REM *** MAIN PROGRAM              ***
136 :
140 SYS 53182:PRINT"{CLR}":REM EXECUTE MACHINE LANGUAGE PR
OGRAM,GET ALL VALUES
150 PRINTPEEK(AX);TAB(6);PEEK(AY);TAB(12);PEEK(BA);TAB(18)
;
160 PRINTPEEK(BX);TAB(24);PEEK(BY);TAB(30);PEEK(BB)
170 GOTO 140:REM ENDLESS LOOP , OUTPUT VALUES CONTINUOUSLY

READY.
```

Chapter 11 : CAD -- Computer-Aided Design

11.1 What exactly is CAD?

You have no doubt seen figures on television or in technical magazines which represented the floor plan of a house, for instance. And it seemed to you that these figures had the look of a computer display. Such pictures are products of CAD systems, Computer-Aided Design.

How does one design with a computer and what sort of aid does it give to the engineer or designer? Perhaps you will object that one can develop a floor plan or printed-circuit board layout quite well on paper, and even the inside (engine) and outside of a car can be designed without a computer. Unfortunately, however, if you make a mistake you often have to start all over again, costing time, materials and money. With a computer system changes can be made at any time, by simply removing an existing line with the wave of a lightpen or in the case of P.C. board layouts, replacing one element with another.

Computer-aided design can be characterized by more than just drawing on a video screen--there are other important possibilities and capabilities. Naturally, a CAD system can save and print out a finished design, which can often be fed directly into the production process. Less conspicuous are the structures already present in the system. Let us take as an example a CAD system on which layouts for printed circuit boards will be developed. Such a system has at its disposal a large library which contains all of the basic elements which are used in such an application. With the press of a

key, the movement of a lightpen, or the touch on a graphics tablet one can call up an element at will and place it at the desired location. Even many standard layouts which will be used multiple times can be called from such a library file and represented on the system.

You may have suspected that the computer is also in the position to inform the designer of possible errors in the layout. Possible improvements can also be "proposed" by the computer, so that even today it is possible to foresee the time when a perfect system of this type could almost completely replace people, as disquieting as this though may be. But don't worry, your Commodore 64 certainly isn't a threat to your job, especially not in the area of CAD. It doesn't have the all the necessary capabilities.

Now we come to the question of what hardware requirements a CAD system needs. A professional system is far out of the reach of many of us, since a system with the capabilities mentioned above requires a enormous amount of memory, special, expensive software, a special color screen with storage capability, and very likely hard disks as a storage medium as well. In addition we have minor items such as graphics tablets (devices something like the Koala Pad for the C64) as well as an appropriate plotter.

You are probably wondering why we have included a chapter on CAD in this book if the Commodore 64 neither comes close to a professional CAD system in price or technology. Even the little Commodore allows limited attempts at design on the screen. Because of the slow speed of graphics routines in BASIC, these parts of CAD-type programs must be written in machine language.

11.2 CAD with the Commodore 64

The first point to consider when thinking about a CAD system is just what it is one wants to design or develop. This question is as important for professionals as it is for us, since the programming effort and memory required depend on the answer. In the next two sections I give suggestions for possible applications which can be realized on the C64. The first example can be programmed completely in BASIC, although at the cost of speed. The Commodore 64 possesses one prerequisite for a CAD program in its capability to display high-resolution graphics, even if this doesn't extend to the possibilities offered by industrial systems. An appropriate color monitor would also be quite expensive, costing about 2500-3500 dollars, enough to break any home computer budget. The management of the high-resolution graphics on the C64 is not one of the easiest things to do, as you may know, whether one is working in BASIC or machine language.

Those interested in writing CAD programs, as we said before, will not be able to avoid writing in machine language. Therefore in this book we will deal only with the possibilities of the C64 in general, since it is written for BASIC programmers, who may not feel at home with assembly language programming. Those who do know assembly language should at least take the suggestions given as food for thought.

11.2.1 Three-dimensional representations

In professional CAD, one distinguishes between three basic types of representations, explained in the list below. These divisions also apply roughly for microcomputers such as the C64.

Representation of graphics in CAD

1. The 2-D representation:

This includes all two-dimensional shapes such as lines, circles, arcs, curves, polygons, and single points. One can rotate, scale, invert, reverse, reflect, and superimpose the images. The objects can be filled with patterns and labeled as well.

2. The 2 1/2-D representation:

These figures are not yet "true" three-dimensional representations, but through rotating and moving the object, one can get quite acceptable results.

3. The 3-D representation:

Here one has the possibility to represent an object on the screen from all angles and views. In contrast to the 2-D model, the construction elements can be freely positioned in space. Naturally, the effort on the part of the programmer is relatively great, but still executable on small computers.

With all three methods one can still decide whether only the edges of the objects will be displayed, or if they will be filled. Outline drawings can be likened to wire models (see figure 1).

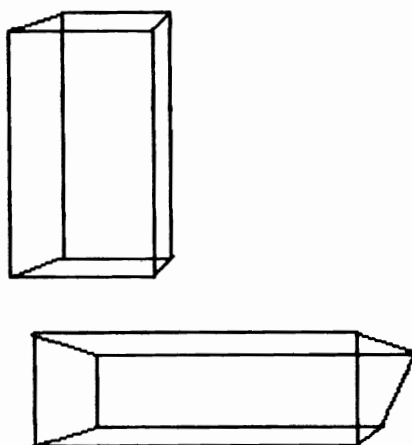


fig. 1

In order to achieve a spatial representation on the screen, which is two-dimensional, we must enter into spatial geometry. Several perspective possibilities work well for use on computers. One speaks of either isometric, dimetric, or central projections. The computation involved is required to calculate the screen positions of points for the desired representation with the help of projection equations and transformation matrices. We will note the most important equations, along with a short description of each perspective.

1. Cavalier perspective

This is probably the simplest procedure for three-dimensional representation. The X,Y coordinates are placed parallel to the screen plane. This avoids having to compress the X and Y coordinates. The projection of the z-axis is rotated 45 degrees from the horizontal and is thereby cut in half. Other angles are also possible. A computer program calculates the new points with the following pair of equations:

$$x_1 = x - z * v * \cos \text{Phi (angle)}$$

$$y_1 = y - z * v * \sin \text{Phi (angle)}$$

where v is the scaling factor

In the simplest cases the required points and values can be stored in DATA statements, but that would contradict the dynamic structure of a CAD system. A program must be created which allows a new point to be entered and displays it using the new positions obtained from the formulas above.

2. Isometric projection

Here the relationship of the three axes to each other is 1:1:1, yielding an angle of 120 degrees between the axes. The equations are required for this projection are:

$$x_1 = x * \cos (30) - z * \cos (30)$$

$$y_1 = y - x * \cos (60) - z * \cos (60)$$

BASIC extension packages such as ULTRABASIC-64 or Simons' BASIC where the new coordinates calculated in the subroutine can be easily processed with appropriate commands make programming projection programs easier. Speed which at least comes closer to the speed of machine language can be achieved by compiling the BASIC program. This is for BASIC programmers who have no desire to learn assembly language.

3. Central projection

The representation which is most "true to nature" is the central projection. The principle of the central projection is shown in figure 2. The following equations yield the values of x and y:

$$y_1 = (y - h) / (h - z) * f$$

$$x_1 = (x - b) / (d - z) * f$$

d : viewer distance (lateral)

h : viewer eye height

b : viewer distance (frontal)

f : dimension factor ("focal length")

The dimetric will not be described in greater detail here since it is a variation of the cavalier perspective in which the angle between the x and z axes is greater than 90 degrees.

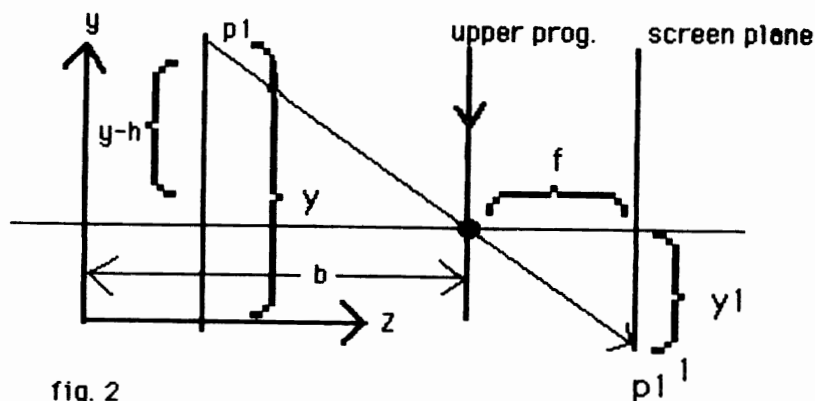


fig. 2

The desired type of projection can be achieved through the use of the appropriate set of formulas. If the graphic is to be made to move, the time expenditure becomes very large and machine language must be used.

11.3 Creating a printed-circuit board layout

In contrast to the applications mentioned up to this point, this project is concerned with representations in two-dimensional space. The so-called Lee algorithm is used for creating a PC layout on a computer. This is probably the oldest and best-known algorithm for untangling conductor paths. This procedure is denoted as routing. One must consider the way the board is represented in the computer. The Lee algorithm works very well with the help of a matrix. There is also the possibility of working with vectors, but this increases the computation time considerably. Even if we use the matrix, the amount of available memory is limited since we require room for the operating system. The entire 64K of the '64 is not available to us.

The individual matrix positions (solder holes) must first be converted to corresponding memory locations so that they can be addressed. Here is a short description of the Lee algorithm:

If we imagine the PC board as a 128x128 matrix, then we get elements with an edge length of 1/20 of an inch. This represent the solder points. The router takes the start and end points of a connection and begins with the neighbor fields of the start point. The unused fields are marked (encoded) and the connection made. Both are saved. The router finds its way back to the start by way of the coding.

The flow of such a program should have the following construction:

Create: Component list List of the connections

Placing the components:

Place and element with the keyboard, etc.

 rotation, shifting

Create connections

Combine component list and position

Improve connection list (optimizing)

Untangling conductor paths:

Divide by hand

Move the paths

Automate disentanglement

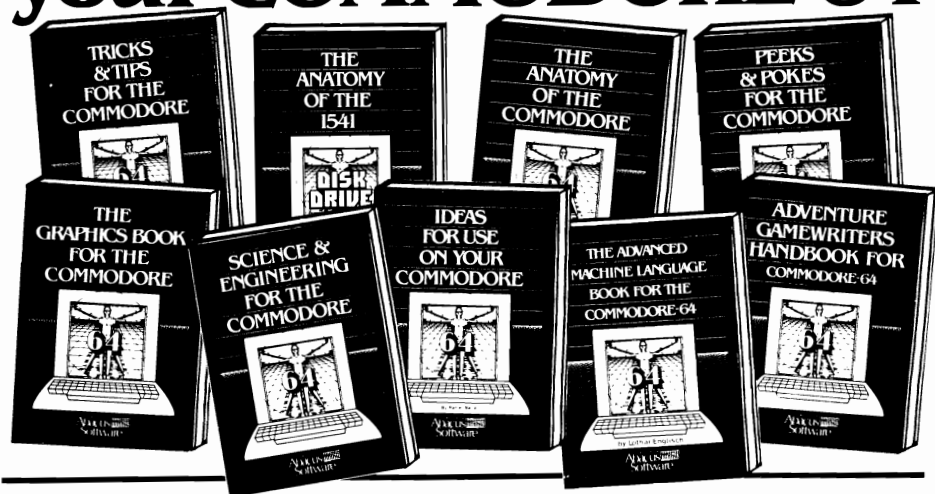
Is data correct? Then output

Appendix

The program listings in this book were produced on an Epson printer with a Card? printer interface. The interface was set in the SPECIAL LISTING MODE to clarify the program listings. The chart below identifies the special characters and their Commodore values.

DEFINITION	LISTING SYMBOL
"CURSOR DOWN	{C/DN}"
"CURSOR UP	{C/UP}"
"CURSOR LEFT	{C/LF}"
"CURSOR RIGHT	{C/RT}"
"HOME CURSOR	{HOME}"
"CLEAR SCREEN	{CLR}"
"REVERSE ON	{RVON}"
"REVERSE OFF	{RVOF}"
"RED	{RED}"
"WHITE	{WHT}"
"GREEN	{GRN}"
"BLUE	{BLUE}"
"ORANGE	{ORNG}"
"BLACK	{BLK}"
"BROWN	{BRN}"
"LIGHT RED	{LRED}"
"GREY 1	{GRY3}"
"GREY 2	{GRY2}"
"GREY 3	{GRY1}"
"LIGHT GREEN	{LGRN}"
"LIGHT BLUE	{LBLU}"
"PURPLE	{PURP}"
"YELLOW	{YELO}"
"CYAN	{CYAN}"
"SHIFTED SPACE	[SS]
"INSERT	{INS}"
"DELETE	[DL]
"F1 FUNCTION KEY	{F1}"
"F2 FUNCTION KEY	{F2}"
"F3 FUNCTION KEY	{F3}"
"F4 FUNCTION KEY	{F4}"
"F5 FUNCTION KEY	{F5}"
"F6 FUNCTION KEY	{F6}"
"F7 FUNCTION KEY	{F7}"
"F8 FUNCTION KEY	{F8}"

Required Reading for your COMMODORE 64



TRICKS & TIPS FOR YOUR C-64 - treasure chest of easy-to-use programming techniques. Advanced graphics, easy data input, enhanced BASIC, CP/M, character sets, transferring data between computers, more.
ISBN# 0-916439-03-8 275 pages \$19.95

GRAPHICS BOOK FOR C-64 - from fundamentals to advanced topics this is most complete reference available. Sprite animation, Hires, Multicolor, lightpen, IRQ, 3D graphics, projections. Dozens of samples.
ISBN# 0-916439-05-4 350 pages \$19.95

SCIENCE & ENGINEERING ON THE C-64 - starts by discussing variable types, computational accuracy, sort algorithms, more. Topics from chemistry, physics, biology, astronomy, electronics. Many programs.
ISBN# 0-916439-09-7 250 pages \$19.95

ANATOMY OF 1541 DISK DRIVE - bestselling handbook available on using the floppy disk. Clearly explains disk files with many examples and utilities. Includes complete commented 1541 ROM listings.
ISBN# 0-916439-01-1 320 pages \$19.95

ANATOMY OF COMMODORE 64 - insider's guide to the '64' internals. Describes graphics, sound synthesis, I/O, kernel routines, more. Includes complete commented ROM listings. Fourth printing.
ISBN# 0-916439-00-3 300 pages \$19.95

IDEAS FOR USE ON YOUR C-64 - Wonder what to do with your '64? Dozens of useful ideas including complete listings for auto expenses, electronic calculator, store window advertising, recipe file, more.
ISBN# 0-916439-07-0 200 pages \$12.95

PEEKS & POKES FOR THE C-64 - programming quickies that will simply amaze you. This guide is packed full of techniques for the BASIC programmer.
ISBN# 0-916439-13-5 180 pages \$14.95

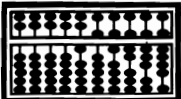
ADVANCED MACHINE LANGUAGE FOR C-64 - covers topics such as video controller, timer and real time clock, serial and parallel I/O, extending BASIC commands, interrupts. Dozens of sample listings.
ISBN# 0-916439-06-2 210 pages \$14.95

ADVENTURE GAMEWRITER'S HANDBOOK - is a step-by-step guide to designing and writing your own adventure games. Includes listing for an automated adventure game generator.
ISBN# 0-916439-14-3 200 pages \$14.95

Call today for the name of your nearest local dealer Phone: (616) 241-5510

Other titles are available, call or write for a complete free catalog.

For postage and handling include \$4.00 (\$6.00 foreign) per order. Money order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted. Michigan residents include 4% sales tax. CANADA: Book Center, Montreal Phone: (514) 332-4154

You Can Count On  **Abacus Software**

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510

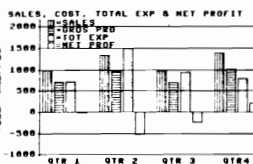
Make your '64 work fulltime

MAKE YOUR OWN CHARTS...

CHARTPAK-64

produces professional quality charts and graphs instantly from your data. 8 chart formats. Hardcopy in two sizes to popular dot matrix printers. \$39.95

ISBN# 0-916439-19-4

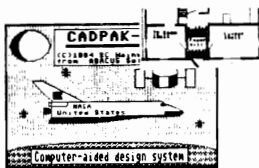


Also Available CHARTPLOT-64 for unsurpassed quality charts on plotters. ISBN# 0-916439-20-6 \$84.95

DETAIL YOUR DESIGNS...

CADPAK-64

superb lightpen design tool; exact placement of object using our Accu-Point positioning. Has two complete screens. Draw LINES, BOXES, CIRCLES, ELLIPSES; pattern FILLING; freehand DRAW; COPY sections of screen; ZOOM in and do detail work. Hard copy in two sizes to popular dot matrix printers. ISBN# 0-916439-18-6 \$49.95



CREATE SPREADSHEETS & GRAPHS...

POWER PLAN-64

not only a powerful spreadsheet packages available, but with built in graphics too. The 275 page manual has tutorial section and HELP screens are always available. Features field protection; text formatting; windowing; row and column copy, sort; duplicate and delete. ISBN# 0-916439-22-4 \$49.95

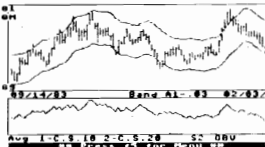
Coordinates: C10		POWER PLAN-64	
	A	B	C
1	Sales	Jan	Feb
2	Distributors	47.2	54.2
3	Retailers	27.9	35.4
4	Mail Order	18.5	22.7
5			
6		93.6	113.3
7			
8	Expenses		
9	Materials	8.2	9.2
10	Office	2.0	2.8
11	Shipping	4.4	5.0
12	Advertising	12.9	15.8
13	Payroll	10.5	10.7
14			
15		38.0	41.5
16			
17	Profit	55.6	71.8

FREE PEEKS & POKES POSTER WITH SOFTWARE
For name & address of your nearest dealer call (616) 241-5510

CHART YOUR OWN STOCKS...

TAS-64

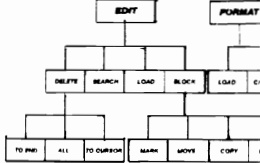
sophisticated technical analysis charting package for the serious stock market investor. Capture data from DJN/RS or Warner services or enter and edit data at keyboard. 7 moving averages, 3 oscillators, trading bands, least squares, 5 volume indicators, relative charts, much more. Hardcopy in two sizes, most printers. ISBN# 0-916439-24-0 \$84.95



DO YOUR OWN WORD PROCESSING

TEXTOMAT-64

flexible wordprocessing package supporting 40 or 80 columns with horizontal scrolling. Commands are clearly displayed on the screen awaiting your choice. Quickly move from editing to formatting to merging to utilities. Will work with virtually any printer.



ISBN# 0-916439-12-7 \$39.95

ORGANIZE YOUR DATA...

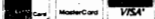
DATAMAT-64

powerful, yet easy-to-use data management package. Free form design of screen using up to 50 fields per record. Maximum of 2000 records per diskette. Complete and flexible reporting. Sorting on multiple fields in any combination. Select records for printing in desired format. ISBN# 0-916439-16-X \$39.95

INVENTORY FILE	
Item Number	Description
Onhand	Price
Location	
Reord. Pt.	Reord. Qty
Cost	

Other titles available. For FREE CATALOG and name of nearest dealer, write or call (616) 241-5510. For postage and handling, include \$4.00 (\$6.00 foreign) per order. Money Order and checks in U.S. dollars only. Mastercard, VISA and American Express accepted. Michigan residents include 4% sales tax.

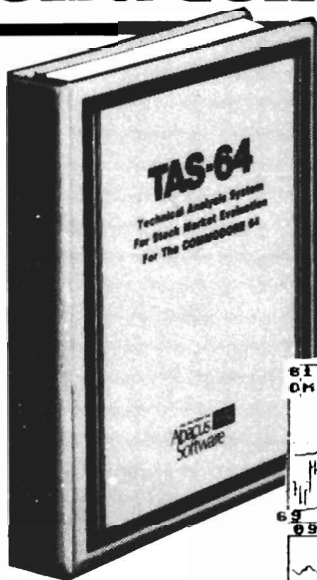
CANADA: Book Center, Montreal (514) 332-4154



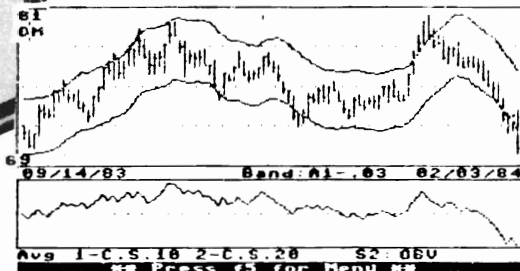
You Can Count On Abacus Software

P.O. Box 7211 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510

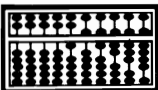
Technical Analysis on a Commodore 64



- Online Data Collection with DJN/RS or Warner Data Entry and Edit for 300 periods
- 7 Moving Averages
- 5 Volume Indicators
- Least Squares
- Trading Bands
- Comparison and Relative Charts
- Hardcopy to most printers
- 150 Page Manual
- 1541 Drive & Optional Printer
- \$84.95 (+ \$4.00 shipping).



You Can Count On
Abacus



Software

P.O. Box 7211 Grand Rapids, MI 49510 For Quick Service Call 616 241-5510

SCIENCE & ENGINEERING FOR THE COMMODORE

This book is for owners of the Commodore 64 who wish to use their computer for serious science and engineering applications. It contains examples from the fields of astronomy, biology, physics, mathematics and others. Each subject is discussed thoroughly and includes complete program listings.

ISBN 0-916439-09-7

YOU CAN COUNT ON
Abacus
Software



P.O. BOX 7211 GRAND RAPIDS, MICH. 49510 PHONE 616-241-5510