NUMERICAL METHODS IN CHEMISTRY

K. JEFFREY JOHNSON

Department of Chemistry
University of Pittsburgh
Pittsburgh, Pennsylvania

MARCEL DEKKER, INC. New York and Basel

Library of Congress Cataloging in Publication Data

Johnson, K. Jeffrey [Date]
Numerical methods in chemistry.

(Undergraduate chemistry ; v. 7)
Includes bibliographical references and index.
1. Chemistry--Data processing. 2. Chemistry--
Computer programs. I. Title.
QD39.3.E46J63 542'.8 79-25444
ISBN 0-8247-6818-3

MARCEL DEKKER, INC.
270 Madison Avenue, New York, New York 10016

Current printing (last digit):
10 9 8 7 6 5 4 3 2

PRINTED IN THE UNITED STATES OF AMERICA

PREFACE

The digital computer has become one of the most important and versatile tools available to chemists. Crystallographers and quantum chemists were among the first to use computers on a large scale. Today chemists are using computers not only for such classical "number-crunching" applications as crystal structure determination and ab initio calculations, but also for routine data reduction, on-line data acquisition and control of experiments, computer-assisted instruction, information retrieval, and even synthesis of organic compounds. An operational knowledge of computers and computer programming is rapidly becoming a requirement for chemists.

This book is intended to serve primarily as a textbook for a numerical methods and computer applications course for chemistry students. The emphasis is on software (computer programming) rather than hardware. Also, the results of numerical analysis and linear algebra are presented and applied to the solution of chemistry problems. The mathematics have not been derived. There are more than 50 computer programs in this book, 43 of which have been fully documented.

The book can be logically divided into three parts. Chapters 1 and 2 provide a review of the Fortran programming language and a collection of programs which have closed-form solutions. Chapters 3 through 8 introduce and apply various numerical methods to the solution of chemistry problems. Chapter 9 contains an overview of some additional applications of computers in chemistry, and includes a relatively extensive bibliography.

The computer programs are all written in Fortran. The particular dialect of Fortran is Fortran-10, provided by the Digital Equipment Corporation. The programs have been implemented on a DECSystem-1099 computer. Slight modifications may be required to implement some of these programs on other computers. The execution times cited do not include compilation time.

This book has evolved with the senior-level elective course, "Numerical Methods in Chemistry," which has been offered several times at the University of Pittsburgh. I am deeply indebted to the students who have participated in the development of this course. I particularly wish to acknowledge my graduate and undergraduate student collaborators who have helped me develop the software documented in these pages. I am happy to acknowledge the Pitt Computer Center staff for their cooperation and assistance, and the University for the computer time. I am grateful to my colleagues for providing me the opportunity to write this book. Mrs. Barbara Hunt deserves special commendation for typing this book and its earlier draft. Finally, I am particularly grateful to my wife, Sharyn, for her patience and encouragement.

K. Jeffrey Johnson

CONTENTS

CHAPTER 1

COMPUTER PROGRAMMING AND FORTRAN

A block diagram of a relatively large, general purpose computer is given in Figure 1.1. The "brain" of the computer is the central processing unit (CPU). The "muscle" is the arithmetic and logical unit. The CPU is the control center for all computer the arithmetic and logical units. The CPU is the control center for all computer operations. These operations are executed using the binary number system (0,1). The rate of execution of these binary operations is on the order of a million to a billion operations per second.

The memory of the computer can be divided into two types, core memory and auxiliary memory. Core memory is a rapid-access device for storage and retrieval of information under control of the CPU. Auxiliary memory is slower (and less expensive) and consists of tapes, disks, drums, and other devices. Core memory consists of an array of computer words, each word consisting, in turn, of a fixed number of bistable memory elements, or bits (binary digits). These unique magnetic or electronic states are correlated with the 0 and 1 of the binary number system. The DECSystem-10 computer contains 36-bit words. The Digital Equipment Corporation sells core memory units for the DECSystem-10 computer in 64k blocks ($k = 2^{10} = 1024$). Each of these 64k words is addressable by the CPU.

All large computer centers have extensive auxiliary storage facilities. These include magnetic tapes, disks, drums, etc. These devices have slower access time than core memory because the information is stored sequentially on the device. For example, magnetic tape must be moved to the appropriate location on the tape to access information. The time required to retrieve information from core memory using the DECSystem-1099 computer is less than 1 μsec. Access times for auxiliary devices range from microseconds to seconds. The auxiliary devices are used for long-term storage of computer programs and data.

The computer outlined in Figure 1.1 is configured for three types of operations: (a) batch processing, (b) remote job entry (RJE), and (c) time sharing. The input and output (I/O) devices for a batch-processing computer usually consist of card readers, magnetic tape drives, and line printers. The jobs (computer programs) are queued and executed sequentially according to some scheduling algorithm (computer-based procedure). Except for large compute-bound (as opposed to I/O-

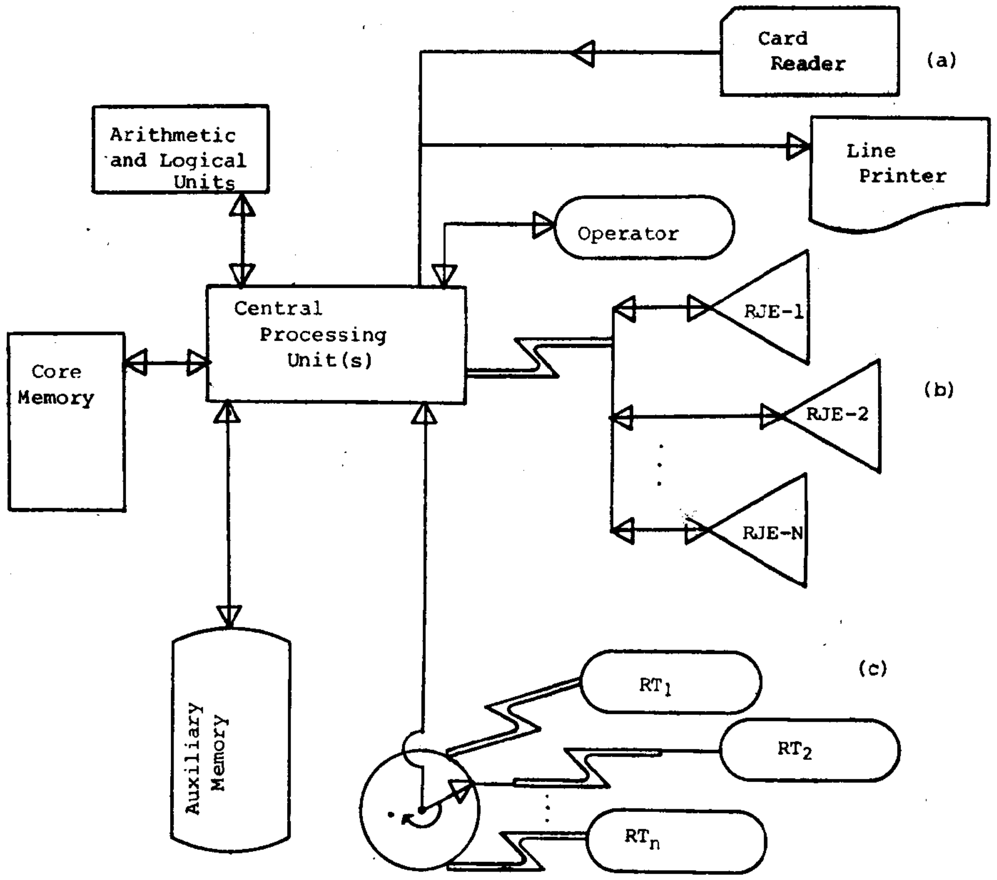

Figure 1.1 Block diagram of a computer showing (a) batch processing, (b) remote job entry, and (c) time sharing.

bound) jobs, the CPU is idle much of the time in batch processing because the CPU is several orders of magnitude faster than the mechanical I/O devices.

RJE is an extension of batch processing designed to increase the utilization of the CPU and to facilitate access to the system by the users. Each RJE station contains I/O devices. In an RJE configuration the CPU has a series of I/O queues to process. The jobs are read, executed sequentially according to the priority assigned by the scheduling algorithm, and the results directed to the appropriate RJE station for printing. However, a powerful CPU will still be idle for a significant fraction of the time, and therefore can support a number of remote terminals, providing a time-sharing environment. The arrow in Figure 1.1 revolves around the ring making contact with each remote user for a small fraction of a second.

During that time information can be read from the terminal, processing of information can be completed, or results printed at the terminal. If the average response time for the time-sharing system is less than 1 sec, it appears to the user at the terminal that he is the sole user of the system. He may in reality be sharing the computer with 50 other time-sharing users, 10 RJE stations, and a background batch queue.

1.1 PROGRAMMING LANGUAGE

Users communicate with computers using one of four classes of computer languages. The most primitive of these is machine language, in which the instructions (store, add, branch, etc.) are coded in binary or some other numeric code. The second class of computer languages is assembly language, where the instructions are coded using mnemonics to represent the numeric codes, for example, STR for store. Each computer has its own set of assembly instructions. Programs written in the assembly language are compiled, that is, translated into machine language by a program called the assembler. The resulting machine language program is then executed. The third class is the compiler languages, for example, Fortran, COBAL, and PL/1. These higher level languages contain key words, for example, READ, GO TO, and DO. Programs written in Fortran (source programs) are translated by a Fortran compiler into machine code (the object program) which is then executed. The fourth class is the set of interpretive languages, for example, BASIC and APL. These languages operate in time-sharing mode, are relatively easy to learn and use, and allow the user a maximum of control over the writing, debugging, and execution of the program.

Fortran is the programming language most widely used by scientists and engineers in the United States. The word Fortran derives from formula translation, and indeed Fortran statements closely resemble algebraic formulas. For example, the formulas $P = nRT/V$ and $A = A_0e^{-kt}$ are coded in Fortran as follows:

```
P=N*R*T/V     and     A=AØ*EXP(-K*T)
```

Here "*" is the multiplication symbol, "Ø" is used to differentiate the number zero and the letter "O," and "EXP" refers to the exponential function. Each computer has its own dialect of Fortran. The programs in this book have been compiled using the Fortran-10 compiler on a DECSystem-1099 computer. Some modifications may be required for successful compilation on other computers.

1.2 ELEMENTS OF THE FORTRAN LANGUAGE

The Fortran compiler reads Fortran statements and translates them into machine-readable code. The set of Fortran statements will be called the source deck and the output of the Fortran compiler will be called the object deck. Source decks are frequently punched on 80-column cards. The following column convention is used:

Column 1	A "C" in column 1 indicates a comment card; comment cards are ignored by the compiler.
Columns 1-5	Statement number; statement numbers may range from 1 to 99999.
Column 6	Continuation; any character (except a blank) in column 6 identifies the card as a continuation of the preceding card.
Columns 7-12	The Fortran source statement.
Columns 73-80	Available for identification, sequence numbers, remarks, etc.; these eight columns are ignored by the Fortran compiler.

Consider the following Fortran statements:

```
C
C     CALCULATE THE DETERMINANT OF A 3X3 MATRIX
C
C        10        20        30        40        50        60
C23456789012345678901234567890123456789012345678901234567890
C
  100 DETOFA=A(1,1)*A(2,2)*A(3,3) + A(1,2)*A(2,3)*A(3,1)
     1 + A(1,3)*A(2,1)*A(3,2) - A(1,2)*A(2,1)*A(3,3)
     2 - A(1,1)*A(2,3)*A(3,2) - A(1,3)*A(2,2)*A(3,1)
```

The first six cards are comment cards. The first character of the Fortran statement starts in column 7. The "1" in column six of the second statement indicates that it is a continuation of the preceding statement. The Fortran-10 compiler will accept up to 19 consecutive continuation cards. The statement "DETOFA=..." has been arbitrarily assigned the statement number 100.

1.2.1 Constants

Six types of constants are considered here: integer, real (single precision), double precision, complex, logical, and literal. Integer constants are signed numbers without a decimal point. The DECSystem-10 computer has a 36-bit word (36 binary digits per word). The sign of the integer is stored in the first bit (0 for positive, 1 for negative). The binary equivalent of the integer value is stored in the remaining 35 bits. The available range is

$-2^{35} \leq$ integer value $\leq 2^{35}-1$

or

$-34{,}359{,}738 \leq$ integer value $\leq 34{,}359{,}738{,}367$

The following are examples of valid and invalid integer constants,

Valid	Invalid	Reason
101	135.	Decimal point
-67	-6,830	Comma
34359738367	34359738368	Too large

Real constants are signed numbers written either with a decimal point or in exponential notation. Exponential notation is illustrated by the following examples:

```
0.157 = 1.57E-1      9756 = 9.756E3
      = 15.8E-2           = 97.56E2
      = 157.E-3           = 975.6E1
```

The 36-bit word is divided into two parts to represent real constants. The first 9 bits contain the binary equivalent of the signed exponential part, and the remaining 27 digits contain the binary equivalent of the signed fractional part of the constant. The precision of 27 binary bits is approximately 8 decimal digits. The range (in absolute value) of real constants is

$$1.4 \times 10^{-39} < \text{real constant} < 1.7 \times 10^{38}$$

The following are examples of valid and invalid real constants.

Valid	Invalid	Reason
1.7E-5	-3.4E41	Exponent too large
1.76.937	96,457.02	Comma
1.2345678	107	No decimal point

Double-precision constants use two 36-bit words. Only 9 bits are assigned to the exponential part, so the range of magnitude of double- and single-precision values is the same. However, the prcision is increased to approximately 16 digits. Double-precision constants contain the letter "D" rather than "E." Examples of valid and invalid double-precision constants follow.

Valid	Invalid	Reason
1.00D-3	6.74	D missing
0.0D0	0.0E0	D missing
6.63D-27	0.64D74	Exponent too large

Complex constants are represented by a pair of integer or real constants separated by a comma and enclosed in parentheses. The first number is the real part, and the second number is the imaginary part of the complex constant. For example,

Valid	Invalid	Reason
(0.0,0.0)	6.3,2.4	Parentheses missing
(-5.9E-4,6.3E-3)	(6.9E-4)	Imaginary part missing
(2,-5)	(8.8E72,9.6E-67)	Exponents too large in magnitude

The two logical constants are .TRUE. and .FALSE. The logical constants must be delimited by periods.

Literal constants are denoted either by a pair of apostrophes or with the specification "nH," where n denotes the number of characters in the literal constant and H stands for "Hollerith." Literal constants can contain one or more of the 26 alphabetic letters, the nine numerals, and/or the set of special characters (",#,$,%,&,etc.). For example,

Valid	Invalid	Reason
'DATA'	'Y(OBS)	Closing apostrophe missing
10H 62 PART A	6HY(CALC)	Should be 7H...

1.2.2 Variables and Specification Statements

Variable names in Fortran consist of between one and six alphameric (alphabetic and numeric) characters, the first of which must be alphabetic. Five types of variables will be considered here: integer, real, double-precision, complex and logical. By convention, variable names which begin with one of the letters I, J, K, L, M, or N are considered integer variables. Variable names which begin with any other letter in the alphabet are real (single-precision) variable names. Valid integer variable names include KJJ, MASS, IGO, J100, and LETITB. Valid real variable names include X, DATA, ENTROP, Q74, and Z18XY.

Other types of variables must be explicitly declared using a specification statement. The general form is

type $v_1, v_2, v_3, \ldots$

where TYPE is INTEGER, REAL, DOUBLE PRECISION, COMPLEX, or LOGICAL, and v_1, v_2, etc., are variable names. For example,

```
INTEGER SUM, Z10, R20
REAL J,K10,IJUMP
DOUBLE PRECISION A,B,C
COMPLEX C1,C2,C3,Z
LOGICAL L, LOG, FLAG
```

The IMPLICIT specification statement can be used to override the standard Fortran convention. This statement allows the user to declare all variable names starting with a given letter to be of an arbitrary type. For example,

```
IMPLICIT INTEGER (A-Z)
```

The effect of this statement is to declare all variable names in the program to be of integer type. All variables that are single precision (real) by default can be declared double precision by the statement

```
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
```

Variables may be subscripted by the DIMENSION statement

```
DIMENSION INDEX(25),Y(100),A(10,10)
```

The effect of this statement is to allocate 25 integer values to the subscripted name INDEX, 100 real values to be stored in the vector Y, and 100 real values to be stored (columnwise) in the array A.

1.2.3 Operators and Expressions

The five arithmetic operators and a subset of the functions available in Fortran are given in Table 1.1.

Arithmetic expressions consist of arithmetic operators, constants, variables, and function references. Some examples of assignment statements containing expressions are

```
1. DISCR=B*B-4.*A*C
2. ROOT1=(-B + SQRT(DISCR))/(2*A)
3. J=J+1
4. P=(A*B)/(C*D)
5. A=AO*EXP(-CONST*T)
6. Z(I)=(X(I)-Y(I))*ABS(SIN(J*PI*Z))
```

Statement 1 contains only real variable names and a real constant (4.). Statement 2 is a mixed-mode expression because it contains both real variable names and the integer constant 2. When this statement is compiled, Fortran-10 converts the integer constant to a real constant. Statement 3 is a mathematical absurdity, but in Fortran it simply means the replacement of the value of J by the value of J + 1. Statement 4 contains parentheses to avoid the following ambiguity:

Fortran expression	Equivalent algebraic expression
A*B/C*D	ABD/C
A*B/C/D	AB/CD
(A*B)/(C*D)	AB/CD

Table 1.1 Arithmetic Operators and Some Functions

Operator or function	Definition	Example	Comment
+	Addition	Y=A+B	
-	Subtraction	Y=A-B	
*	Multiplication	Y=A*B	
/	Division	Y=A/B	B not 0
**	Exponentiation	Y=A**B	
ABS(arg)	Absolute value	Y=ABS(A)	
SQRT(arg)	Square root	Y=SQRT(A)	arg > 0
EXP(arg)	Exponential, e	Y=EXP(A)	
SIN(arg)	Sine	Y=SIN(A)	arg in radians
ALOG(arg)	ln	Y=ALOG(A)	arg > 0
ALOG10(arg)	log	Y=ALOG10(A)	arg > 0

Statements 5 and 6 show the two versions of the "-" operator. In statement 5, the "-" indicates the unary operation of negation, and statement 6 contains the binary operation of subtraction.

Care should be taken when mixing modes with the exponentiation operator. If the exponent is real, the base must be positive to avoid imaginary numbers. For example, -4**0.5 is 2i, where i is the square root of -1. Also, A**B and A**I are evaluated differently. The former uses the identity,

$$A^B = \exp(B \ln A)$$

and the latter will use repetitive multiplication of I < 10.

Logical expressions consist of relational and logical operators, constants, variables and arithmetic expressions. Six relational operators and five logical operators are given in Table 1.2. The truth table for the five logical operators is given in Table 1.3. Examples of logical expressions assuming I and J are integers, X and Y are single precision, and P and Q are logical variables are

```
I.LT.J
(X.LT.Y) .OR. (X.EQ.0.)
P .AND. (I.EQ.J)
```

If a Fortran statement contains several operators, the execution is determined by the following hierarchy:

functions > ** > (*,/) > (+,-) > (<,≤,>,≥,=,≠)
  > .NOT. > .AND. > .OR. > (.EQV.,.XOR.)

In the case of equal operator precedence, computation proceeds from left to right. For example, the statement

```
Y=A+B/SQRT(C)*D**E+COS(F)
```

Table 1.2 Relational and Logical Operators

Relation or operation	Operator	Example (X and Y are single precision; P and Q are logical)
>	.GT.	X.GT.Y
≥	.GE.	X.GE.Y
<	.LT.	X.LT.Y
≤	.LE.	X.LE.Y
=	.EQ.	X.EQ.Y
≠	.NE.	X.NE.Y
AND	.AND.	P.AND.Q
Inclusive OR	.OR.	P.OR.Q
Exclusive OR	.XOR.	P.XOR.Q
Equivalence	.EQV.	P.EQV.Q
Complementation	.NOT.	P.NOT.Q

T407 B 1

Table 1.3 Truth Table (T = true, F = false)

P	Q	P.AND.Q	P.OR.Q	P.XOR.Q	P.EQV.Q
T	T	T	T	F	T
T	F	F	T	T	F
F	T	F	T	T	F
F	F	F	F	F	T

is executed as follows. Let G=SQRT(C) and H=COS(F),

```
     then           Y=A+B/G*D**E+H
     Now let        P=D**E,
     then           Y=A+B/G*P+H
     Next, let      Q=B/G,
     then           Y=A+P*Q+H
     Finally, let   R=P*Q,
     then           Y=A+R+H
```

1.3 CONTROL STATEMENTS

The following Fortran statements are considered in this section: GO TO (branching), IF (conditional branching), DO (looping), CONTINUE, PAUSE, STOP, and END. The execution of a Fortran program proceeds sequentially from the first statement to the last (END) unless one of these control statements is encountered.

1.3.1 The GO TO Statement

Branching is accomplished in Fortran programs with one of three variants of the GO TO statement:

```
     1. Unconditional    GO TO n1
     2. Computed         GO TO (n1, n2,n3,...,nk),i
     3. Assigned         ASSIGN 25 to i
                         ..............
                         GO TO i
                         or
                         GO TO i, (12,25,24)
```

Here n1, n2, ..., nk are statement numbers and i is an integer variable name. The unconditional GO TO statement causes control to branch to the specified statement number. The computed GO TO transfers control to n1 if i = 1, to n2 if i = 2, etc. If i is not in the range $1 \leq i \leq k$, then the next statement is executed. The assigned GO TO transfers control to the statement number corresponding to the value of i. If i is not assigned one of the values in the list (12,25,24), then the next statement is executed. Some examples of GO statements follow.

```
     GO TO 100                      ASSIGN 85 to JJ
     .........                      ...............
     IGO=4                          GO TO JJ
     .....                          ........
     GO TO (10,20,15,75,88),IGO     GO TO JJ, (15,35,85,75)
```

In these examples control is branched to statement number 75 by the computed GO TO, and to statement number 85 by the assigned GO TO statement.

1.3.2 The IF Statement

There are two IF statements in Fortran for conditional branching, the arithmetic IF and the logical IF. The general form of the arithmetic IF is

```
IF ( arithmetic expression ) n1,n2,n3
```

Control is branched as follows:

If the arithmetic expression is <0, branch to statement n1,
If the arithmetic expression is =0, branch to statement n2,
If the arithmetic expression is >0, branch to statement n3.

Statement numbers n1 and n2, or n2 and n3 may be equal, providing the conditions ≤ and ≥, respectively. For example,

```
IF(FOFX)20,25,25
IF((ABS(X-XOLD)/X)-TOL)95,95,91
```

The general form of the logical IF is

```
IF(logical expression) statement
```

Here if the logical expression is true, then the statement is executed. Otherwise, control passes to the next statement. Consider the following examples:

```
1.  IF(N.LE.0)STOP
2.  IF(IPLT.EQ.1)CALL PLOT(X,Y,N,M)
3.  X=P+Q
    IF(X.GE.Y)X=P-Q
```

In the third example, the value assigned to X is P + Q if X is less than Y, P - Q otherwise.

1.3.3 The DO and CONTINUE Statements

Loops (iterative procedures) are coded in Fortran using the DO statement,

```
DO n  i=j1,j2,j3
```

Here n is the statement number designating the terminal statement of the loop, i is an integer variable name, referred to as the index variable, and j1, j2, and j3 are integer constants or variables denoting, respectively, the initial value, final value, and increment of the index variable. The increment is 1 by default. For example,

```
   SUM=0
   DO 10 I=1,N
   SUM=SUM+X(I)
10 CONTINUE
20 AVG=SUM/N
```