CERN Program Library

CERNLIB

Short Writeups

Application Software and Databases

Computing and Networks Division

CERN Geneva, Switzerland

```
CERN Program Library Office
CERN-CN Division
CH-1211 Geneva 23
Switzerland
Tel.  +41 22 767 4951
Fax.  +41 22 767 8630
Internet:  cernlib@cern.ch
```


Contact Person: Jamie Shiers /CN (`shiers@cern.ch`)
Technical Realization: Michel Goossens /CN (`goossens@cern.ch`)

Edition – June 1996

Introduction

The CERN Program Library is a large collection of general-purpose programs maintained and offered in both source and object code form on the CERN central computers. Most of these programs were developed at CERN and are therefore oriented towards the needs of a physics research laboratory. Nearly all, however, are of a general mathematical or data-handling nature, applicable to a wide range of problems.

The library is heavily used at CERN and it is distributed in binary or source form to several hundred laboratories and computer centres outside CERN.

Contents and Organization of the Library

The library contains about 2500 subroutines and complete programs which are grouped together by logical affiliation into little over 450 program packages. 80% of the programs are written in Fortran77 and the remainder in C and in assembly code, usually with a FORTRAN version also available.

A unique code is assigned to each package. This code consists of one letter and three or four digits, the letter indicating the category within our classification scheme. A package consists of one or more related subprograms with one package name and one or more user-callable entry names, all described briefly in a "Short write-up", and if necessary, an additional "Long write-up".

A complete list of program packages with titles and entries sorted by class is given at the beginning of this manual. Then follow all the short write-ups, while the Index at the end of the volume shows the page number (as printed near the inner margin) were a package is defined (in **boldface**) or referenced.

Acknowledgements

K.S. Kölbig has done most of the work for having this manual nicely formated, particularly in the area of getting the many mathematical formulae correct.

About the documentation

This document has been produced using LaTeX[1] with the `cernman` class and the `cernlib` package, developed at CERN. A printable version of each of the routines described in this manual can be obtained as a compressed PostScript file from CERN by anonymous ftp. For instance, if you want to transfer the description of routine E112, then you would type the following (commands that you have to type are underlined):[2]

```
ftp asisftp.cern.ch
Trying 128.141.201.136...
Connected to asis01.cern.ch.
220 asis01 FTP server (SunOS 4.1) ready.
Name (asis01:username):  anonymous
Password:  your_mailaddress
ftp> binary
ftp> cd cernlib/doc/ps.dir/shortwrups.dir
ftp> get e112.ps.gz
ftp> quit
```

[1] Leslie Lamport, *LaTeX – A Document Preparation System*, second edition. Addison–Wesley, 1994

[2] You can of course issue multiple `get` commands in one run. If you do not have the `gunzip` utility on your machine, you can get an non-compressed, ready-to-print version by omitting the `.gz` suffix, i.e. in the example above, `get e112.ps`.

Chapter 1: Catalog of Program Packages and Entries

Elementary Functions

B002	PRMFCT	Prime Numbers and Prime Factor Decomposition
B100	RBINOM	Binomial Coefficient
B101	ATG	Arc Tangent Function
B102	ASINH	Hyperbolic Arcsine
B105	RPLNML	Value of a Polynomial
B300	RSRTNT	Integral of type $R(x, \sqrt{a+bx+cx^2})$

Equations and Special Functions

C200	RZEROX	Zero of a Function of One Real Variable
C201	RSNLEQ	Numerical Solution of Systems of Nonlinear Equations
C202	RMULLZ	Zeros of a Real Polynomial
C205	RZERO	Zero of a Function of One Real Variable
C207	RRTEQ3	Roots of a Cubic Equation
C208	RRTEQ4	Roots of a Quartic Equation
C209	CPOLYZ	Zeros of a Complex Polynomial
C210	NZERFZ	Number of Zeros of a Complex Function
C300	ERF	Error Function and Complementary Error Function
C301	FREQ	Normal Frequency Function
C302	GAMMA	Gamma Function for Positive Argument
C303	GAMMF	Gamma Function for Real Argument
C304	ALGAMA	Logarithm of the Gamma Function
C305	CGAMMA	Gamma Function for Complex Argument
C306	CLGAMA	Logarithm of the Gamma Function for Complex Argument
C309	CCLBES	Coulomb Wave, Bessel, and Spherical Bessel Functions for Complex Argument(s) and Order
C312	BESJ0	Bessel Functions J and Y of Orders Zero and One
C313	BESI0	Modified Bessel Functions I and K of Orders Zero and One
C315	RRIZET	Riemann Zeta Function
C316	RPSIPG	Psi (Digamma) and Polygamma Functions
C317	CPSIPG	Psi (Digamma) and Polygamma Functions for Complex Argument
C318	RELFUN	Jacobian Elliptic Functions sn, cn, dn
C320	CELFUN	Jacobian Elliptic Functions sn, cn, dn for Complex Argument
C321	CGPLG	Nielsen's Generalized Polylogarithm
C322	RFRSIN	Fresnel Integrals
C323	RFERDR	Fermi-Dirac Function
C324	RATANI	Arctangent Integral
C326	RCLAUS	Clausen Function

C327	BSIR4	Modified Bessel Functions I and K of Order 1/4, 1/2 and 3/4
C328	CWHITM	Whittaker Function M of Complex Argument and Complex Indices
C330	RASLGF	Legendre and Associated Legendre Functions
C331	RFCONC	Conical Functions of the First Kind
C332	RDILOG	Dilogarithm Function
C334	RGAPNC	Incomplete Gamma Functions
C335	CWERF	Complex Error Function
C336	RSININ	Sine and Cosine Integrals
C337	REXPIN	Exponential Integral
C338	CEXPIN	Complex Exponential Integral
C339	RDAWSN	Dawson's Integral
C340	BSIR3	Modified Bessel Functions I and K of Order 1/3 and 2/3
C341	BSKA	Modified Bessel Functions K of Certain Order
C342	RSTRH0	Struve Functions of Orders Zero and One
C343	BSJA	Bessel Functions J and I with Positive Argument and Non-Integer Order
C344	CBSJA	Bessel Functions J with Complex Argument and Non-Integer Order
C345	RBZEJY	Zeros of Bessel Functions J and Y
C346	RELI1	Elliptic Integrals of First, Second, and Third Kind
C347	RELI1C	Complete Elliptic Integrals of First, Second, and Third Kind
C348	CELINT	Elliptic Integral for Complex Argument
C349	RTHETA	Jacobian Theta Functions

Integration, Minimization, Non-linear Fitting

D101	SIMPS	Integration by Simpson's Rule
D102	RADAPT	Adaptive Gaussian Quadrature
D103	GAUSS	Adaptive Gaussian Quadrature
D104	RCAUCH	Cauchy Principal Value Integration
D105	RTRINT	Integration over a Triangle
D106	RGS56P	Gaussian Quadrature with Five- and Six-Point Rules
D107	RGQUAD	N-Point Gaussian Quadrature
D108	TRAPER	Trapezoidal Rule Integration with an Estimated Error
D110	RGMLT	Gaussian Quadrature for Multiple Integrals
D113	CGAUSS	Adaptive Complex Integration Along a Line Segment
D114	RIWIAD	Adaptive Multidimensional Monte-Carlo Integration **[Obsolete]**
D120	RADMUL	Adaptive Quadrature for Multiple Integrals over N-Dimensional Rectangular Regions
D151	DIVON4	Multidimensional Integration or Random Number Generation **[Obsolete]**
D200	RRKSTP	First-order Differential Equations (Runge-Kutta)
D201	RDEQBS	First-order Differential Equations (Gragg–Bulirsch–Stoer)
D202	RDEQMR	First-order Differential Equations (Runge–Kutta–Merson)
D203	RRKNYS	Second-order Differential Equations (Runge–Kutta–Nyström)

D300	EPDE1	Elliptic Partial Differential Equation
D302	ELPAHY	Fast Partial Differential Equation Solver
D401	RDERIV	Numerical Differentiation
D501	LEAMAX	Constrained Non-Linear Least Squares and Maximum Likelihood Estimation
D503	RMINFC	Minimum of a Function of One Variable
D506	MINUIT	Function Minimization and Error Analysis
D510	FUMILI	Fitting Chisquare and Likelihood Functions **[Obsolete]**
D601	RFRDH1	Solution of a Linear Fredholm Integral Equation of Second Kind
D700	RFT	Real Fast Fourier Transform
D702	CFT	Complex Fast Fourier Transform
D705	RFSTFT	Real Fast Fourier Transform
D706	CFSTFT	Complex Fast Fourier Transform

Interpolation, Approximations, Linear Fitting

E100	POLINT	Polynomial Interpolation
E102	MAXIZE	Maximum and Minimum Elements of Arrays
E103	AMAXMU	Largest Absolute Number in Scattered Vector
E104	FINT	Multidimensional Linear Interpolation
E105	DIVDIF	Function Interpolation
E106	LOCATR	Binary Search for Element in Ordered Array
E201	RLSQPM	Least Squares Polynomial Fit
E208	LSQ	Least Squares Polynomial Fit **[Obsolete]**
E210	NORBAS	Polynomial Splines / Normalized B-Splines
E211	RCSPLN	Cubic Splines and their Integrals
E222	RCHEBN	Solution of Overdetermined Linear System in the Chebychev Norm
E230	TL	Constrained and Unconstrained Linear Least Squares Fitting
E250	LFIT	Least-Squares Fit to Straight Line
E255	PARLSQ	Least-Squares Fit to Parabola **[Obsolete]**
E406	RCHECF	Chebyshev Series Coefficients of a Function
E407	RCHSUM	Summation of Chebyshev Series
E408	RCHPWS	Conversion of Chebyshev to Power and Power to Chebyshev Series
E409	RTRGSM	Summation of Trigonometric Series

Matrices, Vectors and Linear Equations

F001	LAPACK	Linear Algebra Package
F002	RVADD	Elementary Vector Processing
F003	RMADD	Elementary Matrix Processing
F004	RMMLT	Matrix Multiplication
F010	RINV	Linear Equations, Matrix Inversion
F011	RFACT	Repeated Solution of Linear Equations, Matrix Inversion, Determinant

F012	RSINV	Symmetric Positive-Definite Linear Systems
F105	POLROT	Rotate a Three-Dimensional Polar Coordinate System
F110	MXPACK	TC Matrix Manipulation Package **[Obsolete]**
F112	TR	Manipulation of Triangular and Symmetric Matrices
F116	DOTI	Scalar Product of Two Space-Time Vectors
F117	CROSS	Vector Product of Two 3-Vectors
F118	ROT	Rotating a 3-Vector
F121	VECMAN	Vector Algebra
F122	SCATTER	Search Operations on Sparse Vectors
F123	BVSL	Bit Vector Manipulation Package
F150	MXDIPR	Direct or Tensor Matrix Product
F406	RBEQN	Banded Linear Equations
F500	RLHOIN	Linear Homogenous Inequalities

Statistical Analysis and Probability

G100	PROB	Upper Tail Probability of Chi-Squared Distribution
G101	CHISIN	Inverse of Chi-Square Distribution
G102	PROBKL	Kolmogorov Distribution
G103	TKOLMO	Kolmogorov Test
G104	STUDIS	Student's T-Distribution and Its Inverse
G105	GAUSIN	Inverse of Gaussian Distribution
G106	GAMDIS	Gamma Distribution
G110	LANDAU	Landau Distribution
G115	VAVLOV	Approximate Vavilov Distribution and its Inverse
G116	VVILOV	Vavilov Density and Distribution Functions
G900	RANF	Random Number Generator **[Obsolete]**

Operation Research Techniques and Management Science

H101	RSMPLX	Linear Optimization Using the Simplex Algorithm
H301	ASSNDX	Assignment Problem

Input/Output

I101	EPIO	EP Standard Format Input/Output Package
I202	KUIP	KUIP - Kit for a User Interface Package
I302	FFREAD	Format-Free Input Processing **[Obsolete]**

Output and Graphical Data Presentation

J200	VIZPRI	Print Large Characters
J403	XBANNER	Print Banner Text
J530	BINSIZ	Reasonable Intervals for Histogram Binning

Executive Routines

L210	COMIS	COMIS - Compilation and Interpretation System
L400	PATCHY	Source Code Maintenance

Data Handling

M101	SORTZV	Sort One-Dimensional Array
M103	FLPSOR	Sort One-Dimensional Array into Itself
M104	SORCHA	Sort One-Dimensional Character Array into Itself
M107	SORTR	Sort Rows of a Matrix
M109	SORTRQ	Sort Rows of a Matrix
M215	PSCALE	Find Power-of-Ten Scale for Printing
M220	IE3CONV	Conversion To and From IEEE Number Format
M400	CHTOI	Portable Conversion Between Type CHARACTER and Type INTEGER
M409	UBUNCH	Concentrate and Disperse Character Strings **[Partially obsolete]**
M421	BITBYT	Package for Handling Bits and Bytes
M422	PACBYT	Handling Packed Vectors of Bytes
M423	INCBYT	Increment a Byte of a Packed Vector
M426	BLOW	Unpack Full Words into Bytes
M427	PKCHAR	Pack/Unpack Continuous Byte-strings
M428	LOCBYT	Search for Byte-Content
M429	NUMBIT	Number of One-Bits in a Word
M431	IFROMC	Convert Between Character String and Packed ASCII Form
M432	CHPACK	Utility Routines for Character String Parsing and Construction
M433	INDEXX	Utility Package for Character Manipulation
M434	VXINV	Fast VAX Byte Inversion
M436	BUNCH	Pack Bytes into Full Words
M437	GETBIT	Set or Retrieve a Bit in a String
M438	BTMOVE	Move Bit String
M439	GETBYT	Set or Retrieve a Bit String
M441	BITPAK	Handling Bits and Bytes, Bit Zero the Least Significant
M442	NAMEFD	Fortran Emulation of VM/CMS NAMEFIND Command
M501	IUSAME	Locating a String of Same Words
M502	UOPTC	Decoding Options Characters
M503	UBITS	Locate the One-Bits of a Word or an Array
M507	LENOCC	Occupied Length of a Character String
M508	BITPOS	Find One-Bits in a String

Debugging, Error Handlng

N001	KERSET	Error Processing for Sections A-H of KERNLIB **[Partially obsolete]**
N002	MTLSET	Error Processing for MATHLIB
N100	LOCF	Address of a Variable
N103	IUWEED	Detect Indefinite and Infinite in an Array
N105	TRACEQ	Print Trace-Back
N203	TCDUMP	Memory Dump

Service or Housekeeping Programming Aids

Q100	ZEBRA	Dynamic Data Structure and Memory Manager
Q120	HIGZ	High Level Interface to Graphics and Zebra
Q121	PAW	PAW - Physics Analysis Workstation Package
Q122	SIGMA	SIGMA - System for Interactive Graphical Mathematical Applications
Q123	FATMEN	Distributed File and Tape Management System
Q124	CSPACK	Client Server Routines and Utilities
Q180	HEPDB	Distributed Database Management System
Q210	ZBOOK	Dynamic Memory Management **[Obsolete]**
Q901	INDENT	Indent Fortran Source
Q902	FLOP	FLOP - Fortran Language Oriented Parser
Q904	CONVERT	Fortran 77 to Fortran 90 source form conversion tool
Q905	WYLBUR	Wylbur Phoenix - a Line Editor for ASCII Text Files **[Obsolete]**

Magnet and Beam Design, Electronics

T604	POISCR	Solution of Poisson's or Laplace's Equation in Two-Dimensional Regions

Quantum Mechanics, Particle Physics

U101	LOREN4	Lorentz Transformation
U102	LORENF	Lorentz Transformations
U111	RWIG3J	Wigner 3-j, 6-j, 9-j Symbols; Clebsch-Gordan, Racah W-, Jahn U-Coefficients
U112	RTCLGN	Clebsch-Gordan Coefficients in Rational Form
U501	RDJMNB	Beta-Term in Wigner's D-Function

Random Numbers and General Purpose Utilities

V104	RNDM	Uniform Random Numbers **[Obsolete]**
V105	NRAN	Arrays of Uniform Random Numbers **[Obsolete]**
V113	RANMAR	Uniform Random Number Generator
V114	RANECU	Uniform Random Number Generator
V115	RANLUX	Uniform Random Numbers of Guaranteed Quality
V116	RM48	Double Precision Uniform Random Numbers
V120	RNORML	Gaussian-distributed Random Numbers

V122 CORSET Correlated Gaussian-distributed Random Numbers

V130 RAN3D Random Three-Dimensional Vectors **[Obsolete]**

V131 RN3DIM Random Three-Dimensional Vectors

V135 RNGAMA Gamma or Chi-Square Random Numbers

V136 RNPSSN Poisson Random Numbers

V137 RNBNML Binomial Random Numbers

V138 RNMNML Multinomial Random Numbers

V149 RNHRAN Random Numbers According to Any Histogram

V150 HISRAN Random Numbers According to Any Histogram **[Obsolete]**

V151 FUNRAN Random Numbers According to Any Function **[Obsolete]**

V152 FUNLUX Random Numbers According to Any Function

V202 PERMU Permutations and Combinations

V300 UZERO Preset Parts of an Array

V301 UCOPY Copy an Array

V302 UCOCOP Copy a Scattered Vector

V304 IUCOMP Search a Vector for a Given Element

V306 PROXIM Adjusting an Angle to Another Angle

V401 GRAPH Find Compatible Node-Nets in an Incompatibility Graph

V700 RVNSPC Volume of Intersection of a Circular Cylinder with a Sphere

High Energy Physics Simulation, Kinematics, Phase Space

W150 TRSPRT Transport, Second-Order Beam Optics

W151 TURTLE Beam Transport Simulation, Including Decay

W505 FOWL General Monte-Carlo Phase-Space

W515 GENBOD N-Body Monte-Carlo Event Generator

Statistical Data Analysis and Presentation

Y201 IUCHAN Find Histogram-Channel

Y250 HBOOK Statistical Analysis and Histogramming

Y251 HPLOT HPLOT : HBOOK Graphics Interface for Histogram Plotting

Miscellaneous System-Dependent Facilities

Z001 KERNGT Print KERNLIB Version Numbers

Z007 DATIME Job Time and Date

Z009 CALDAT Calendar Date Conversion

Z020 UMON Usage Monitor for VAX/VMS

Z035 ABEND Abnormal Termination of Fortran Programs

Z036 ABUSER Intercept a Fortran Abend on IBM

Z037 VAXAST Routines to Handle Control-C Interrupts on Vax

Z041 QNEXTE Restart of Next Event

Z042	JUMPXN	Calling a Subroutine by its Address **[Obsolete]**
Z044	INTRAC	Identify Job as Interactive
Z045	IFBATCH	Identify Job as Running in Batch Mode
Z203	XINOUT	Short List Reading and Writing
Z264	IARGC	Returns Command Line Arguments
Z265	CINTF	Immediate Interface Routines to the C Library
Z266	WHOAMI	Get the Name of the Executing Module
Z267	FTOVAX	Convert File-name to and from UNIX Syntax
Z301	VAXTIO	VAX Fortran Interface for Reading and Writing 'Foreign' Tapes
Z303	KAPACK	Random Access I/O Using Keywords **[Obsolete]**
Z310	CFIO	Handle Fixed-length Records on Unix Streams
Z311	CIO	Handle Unix Disk Files
Z313	TMREAD	Terminal Dialog Routines

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.11.1995
Language : Fortran **Revised:**

Prime Numbers and Prime Factor Decomposition

Subroutine subprogram PRMFCT

- sets the first $n \leq 1229$ prime numbers $p_1 = 2,\ p_2 = 3,\ p_5 = 5,\ \ldots\ ,\ p_{1229} = 9973$ into an array;
- performs the decomposition of a positive number $N < 10007$ into its prime factors:

$$N \ = \ 2^{\alpha_1} \cdot 3^{\alpha_2} \cdot 5^{\alpha_3} \ \cdots \ 9973^{\alpha_{1229}};$$

- performs the decomposition of the factorial $N!$ of a positive number $N < 10007$ into its prime factors:

$$N! \ = \ 2^{\alpha_1} \cdot 3^{\alpha_2} \cdot 5^{\alpha_3} \ \cdots \ 9973^{\alpha_{1229}}.$$

Note that this allows in particular to handle quotients of factorials of rather large numbers in an exact way.

Structure:

SUBROUTINE subprogram
User Entry Names: PRMFCT
Files Referenced: Unit 6

Usage:

```
    CALL PRMFCT(MODE,N,NPRIME,NPOWER,M)
```

MODE $=$ 0 : Sets the first n prime numbers into an array.

N (INTEGER) The number n of prime numbers requested.

NPRIME (INTEGER) One-dimensional array of length $\geq$ N. On exit, NPRIME(j), (j $= 1, 2, \ldots,$ N) contains the j-th prime numbers p_j, where $p_1 = 2,\ p_2 = 3,\ p_3 = 5,\ \ldots$

NPOWER (INTEGER) One-dimensional array of length $\geq$ N. On exit, NPOWER(j), (j $= 1, 2, \ldots,$ N) contains the value 1.

M (INTEGER) Contains, on exit, the number n.

MODE $=$ 1, 2 : Performs the decomposition of N (MODE $=$ 1) or $N!$ (MODE $=$ 2) into its prime factors.

N (INTEGER) The number N itself (MODE $=$ 1) or its factorial (MODE $=$ 2) to be decomposed into prime factors.

NPRIME (INTEGER) One-dimensional array of length $\geq$ N. On exit, NPRIME(j), (j $= 1, 2, \ldots,$ M) contains the j-th prime numbers p_j, where $p_1 = 2,\ p_2 = 3,\ p_3 = 5,\ \ldots.$

NPOWER (INTEGER) One-dimensional array of length $\geq$ N. On exit, NPOWER(j), (j $= 1, 2, \ldots,$ M) contains the power α_j corresponding to the prime number p_j.

M (INTEGER) Contains, on exit, the index $M \leq N$ defined by $\alpha_M > 0$ and $\alpha_j = 0$ for $j > M$.

Restrictions:

`MODE` $= 0 : 1 \leq$ `N` ≤ 1229.
`MODE` $= 1$ or `MODE` $= 2 : 2 \leq$ `N` ≤ 10007.

Error handling:

Error `B002.1:` `MODE` $\neq 0$ and `MODE` $\neq 1$ and `MODE` $\neq 2$.
Error `B002.2:` `N` out of range.
In both cases, `NPRIME(j)` and `NPOWER(j)`, ($j = 1, 2, \ldots,$ `N`) are set to zero and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.
•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran
Library: MATHLIB
Submitted: 15.02.1989
Revised: 15.11.1995

Binomial Coefficient

Function subprograms `RBINOM` and `DBINOM` calculate the binomial coefficient

$$\binom{x}{k} = \begin{cases} x(x-1)\dots(x-k+1)/k! & (k>0) \\ 1 & (k=0) \\ 0 & (k<0) \end{cases}$$

for real x and integer k. Function subprogram `KBINOM` calculates the binomial coefficient only for integer $x = n$.

On CDC and Cray computers, the double-precision version `DBINOM` is not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RBINOM`, `DBINOM`, `KBINOM`
Obsolete User Entry Names: `BINOM` $\equiv$ `RBINOM`
Files Referenced: `Unit 6`

Usage:

In any arithmetic expression,

`RBINOM(X,K)`, `DBINOM(X,K)` or `KBINOM(N,K)`

has the value of the binomial coefficient. `RBINOM` is of type `REAL`, `DBINOM` is of type `DOUBLE PRECISION` and `X` has the same type as the function name. `KBINOM, N` and `K` are of type `INTEGER`.

Restrictions:

Function subprogram `KBINOM` can compute only binomial coefficients which lie in the integer range of the machine.

Accuracy:

Full machine accuracy.

Error handling:

If the result of `KBINOM` would lie outside the integer range of the machine, `KBINOM` is set equal to zero and an error message is printed.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 15.02.1989
Revised: 15.03.1993

Arc Tangent Function

Function subprogram ATG calculates, for real arguments x_1 and x_2, $(x_1, x_2) \neq (0., 0.)$, an angle α such that

$$\alpha = \arctan(x_1/x_2) \quad \text{and} \quad 0 \leq \alpha < 2\pi.$$

Note that using the Fortran intrinsic function ATAN2 instead of ATG would result in $-\pi < \alpha \leq \pi$.

Structure:

FUNCTION subprogram
User Entry Names: ATG

Usage:

In any arithmetic expression,

```
ATG(X1,X2)
```

has the value of α (in radians). ATG, X1 and X2 are of type REAL.

Notes:

This function subprogram is equivalent to the statement function

```
ATG(X1,X2)=ATAN2(X1,X2)+(PI-SIGN(PI,X1))
```

where PI $= \pi$.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 07.06.1992
Revised: 15.03.1993

Hyperbolic Arcsine

Function subprograms ASINH and DASINH calculate the hyperbolic arcsine

$$\operatorname{arcsinh}(x) = \ln(x + \sqrt{x^2 + 1})$$

for real argument x.
On CDC and Cray computers, the double precision version DASINH is not available

Structure:

FUNCTION subprograms
User Entry Names: ASINH, DASINH

Usage:

In any arithmetic expression,

ASINH(X) or DASINH(X) has the value $\operatorname{arcsinh}(\mathtt{X})$,

where ASINH is of type REAL, DASINH is of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Approximation by truncated Chebyshev series and functional relations.

Accuracy:

ASINH (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, DASINH (and ASINH on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press New York, 1975) 66.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.12.1994
Revised:

Value of a Polynomial

Function subprograms RPLNML, DPLNML calculate the value of the polynomial

$$p_n(x) \;=\; a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

or

$$q_n(x) \;=\; a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_n$$

for real values x, whereas function subprograms CPLNML, WPLNML calculate the value of the polynomial

$$r_n(z) \;=\; c_0 + c_1z + c_2z^2 + \cdots + c_nz^n$$

or

$$s_n(x) \;=\; c_0z^n + c_1z^{n-1} + c_2z^{n-2} + \cdots + c_n$$

for complex values z.
On CDC and Cray computers, the double-precision versions DPLNML and WPLNML are not available.

Structure:

FUNCTION subprograms
User Entry Names: RPLNML, DPLNML, CPLNML, WPLNML

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
tPLNML(X,N,A,MODE)
```

has, in any arithmetic expression, the value $p_n(x)$ or $q_n(x)$;
for t = C (type COMPLEX), t = W (type COMPLEX*16),

```
tPLNML(Z,N,C,MODE)
```

has, in any arithmetic expression, the value $r_n(z)$ or $s_n(z)$.

X,Z (type according to t) Arguments x or z, respectively.

N (INTEGER) Degree n of $p_n(x)$, $q_n(x)$ or $r_n(z)$, $s_n(z)$.

A,C (type according to t) One-dimensional arrays of dimension (0:d) where d $\geq$ N, containing the coefficients a_k or c_k $(k = 0, \ldots, n)$ in A(k) or C(k), respectively.

MODE (INTEGER) Either +1 for $p_n(x)$, $r_n(z)$ or -1 for $q_n(x)$, $s_n(z)$.

Method:

The Horner scheme is used.

Notes:

A reference with N < 0 or MODE different from +1 or -1 returns the value zero.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.03.1993
Language : Fortran **Revised:**

An integral of type $\mathrm{R}(\mathrm{x}, \sqrt{\mathrm{a}+\mathrm{bx}+\mathrm{cx}^2})$

Subroutine subprograms RSRTNT and DSRTNT calculate, based on indefinite integration, the definite integral

$$I(k,n;a,b,c;u,v) \quad = \quad \int_u^v \frac{x^k\,dx}{(\sqrt{a+bx+cx^2})^n},$$

for $k=-3,-2,-1,0,1,2,3$ and $n=1,3$, provided that $a+bx+cx^2>0$ for $u<x<v$ and the limits u,v are such that the integral converges. In particular, the Cauchy principal value is taken if $k=-1$ and $uv<0$.
On CDC and Cray computers, the double-precision version DSRTNT is not provided.

Structure:

SUBROUTINE subprograms
User Entry Names : RSRTNT, DSRTNT
Files Referenced : Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tSRTNT(K,N,A,B,C,U,V,RES,LRL)
```

K	(INTEGER) Power k of x.
N	(INTEGER) Power n of $\sqrt{a+bx+cx^2}$.
A,B,C	(type according to t) Coefficients a, b, c.
U,V	(type according to t) Limits of integration u, v.
RES	(type according to t) Contains, on exit, the value I provided LRL = .TRUE., the value zero otherwise.
LRL	(LOGICAL) Contains, on exit, the value .TRUE. if the integral exists in the sense described above, the value .FALSE. otherwise.

Restrictions:

1. $|\mathtt{A}|+|\mathtt{B}|+|\mathtt{C}|\neq 0$.   2. $|\mathtt{K}|\leq 3$; N = 1 or N = 3.

Error handling:

Error B300.1: Restriction 1 is not satisfied. Error B300.2: Restriction 2 is not satisfied.
In both cases, RES is set equal to zero and LRL is set equal to .FALSE., and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. I.S. Gradshteyn and I.M. Ryzhik, Table of integrals, series, and products, (Academic Press, New York 1980) Sect. 2.26

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.05.1990
Revised: 01.12.1994

Zero of a Function of One Real Variable

Function subprograms `RZEROX` and `DZEROX` compute, to an attempted specified accuracy, a zero x_0 of a real-valued function $f(x)$ lying in a given interval $[a,b]$, where $f(a)*f(b) \leq 0$.
On computers other than CDC or Cray, only the double precision version `DZEROX` is available. On CDC and Cray computers, only the single-precision version `RZEROX` is available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RZEROX`, `DZEROX`
Obsolete User Entry Names: `ZEROX` $\equiv$ `RZEROX`
Files Referenced: `Unit 6`
External References: User-supplied `FUNCTION` subprogram

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
tZEROX(A,B,EPS,MAXF,F,MODE)
```

has, in any arithmetic expression, the value x_0.

`A,B` (type according to `t`) On entry, `A` and `B` must specify the end points of the search interval. Unchanged on exit.

`EPS` (type according to `t`) On entry, `EPS` must be equal to the accuracy parameter (see **Accuracy**). Unchanged on exit.

`MAXF` (`INTEGER`) On entry, `MAXF` must be equal to the maximum permitted number of references to the function `F` within the iteration loop. Unchanged on exit.

`F` (type according to `t`) Name of a user-supplied `FUNCTION` subprogram, declared `EXTERNAL` in the calling program. This subprogram must set `F(X)` = f(`X`).

`MODE` (`INTEGER`) On entry, `MODE` = 1 or `MODE` = 2 defines the algorithm for finding x_0 (see **Method** and **Notes**).

Method:

Two algorithims are incorporated in this subprogram. These are described in Ref. 1 as "Algorithm M" (`MODE` = 1) and "Algorithm R" (`MODE` = 2). Both are mixtures of linear interpolation, rational interpolation and bisection.

Accuracy:

These subprograms try to compute two numbers x_0 and x_1 lying in the interval $[a,b]$ such that

1. $f(x_0)f(x_1) \leq 0$
2. $|f(x_0)| \leq |f(x_1)|$
3. $|x_0 - x_1| \leq 2 * \mathtt{EPS} * (1 + |x_0|)$

If successful, the value of x_0 is assigned to the function name.

Notes:

1. MODE $= 1$ should be used for fairly simple functions whose evaluation is cheap in comparison with the calculations performed in one iteration step of RZEROX or DZEROX.
2. MODE $= 2$ should be used for more expensive functions. Convergence should be faster than for MODE $= 1$, but the evaluation steps are more expensive.
3. For functions which have a pole near the exact zero, MODE $= 1$ is recommended because of the special character of the interpolation formula which is used.

Error handling:

1. $\mathtt{F(A)} * \mathtt{F(B)} > 0$. The function value is set equal to zero.
2. MODE has a value other than 1 or 2. The function value is set equal to zero.
3. The number of references to F exceeds MAXF. The function value is set equal to the last computed value of x_0 (see **Accuracy**)

For each error a message is printed.

Source:

The subprogram is based on Algol programs described in Ref. 1.

References:

1. J.C.P. Bus and T.J. Dekker, Two efficient algorithms with garanteed convergence for finding a zero of a function, ACM Trans. Math. Software **1** (1975) 330–345.

•

Author(s) : J.J. Moré, M.Y. Cosnard

Submitter : K.S. Kölbig

Language : Fortran

Library: MATHLIB

Submitted: 01.06.1989

Revised: 01.12.1994

Numerical Solution of Systems of Nonlinear Equations

Subroutine subprograms RSNLEQ and DSNLEQ compute a vector x_i, $(i = 1, 2, \ldots, n)$, which approximates an exact solution x_i^* of the system of n nonlinear equations with n unknowns

$$F_i(x_1, \ldots, x_n) = 0, \quad (i = 1, 2, \ldots, n).$$

These subroutines incorporate two convergence test, making use of arguments FTOL and XTOL respectively. If x_i, $(i = 1, 2, \ldots, n)$, denotes the result of the current iteration, and x_i' the result of the previous iteration, the calculation is terminated if either of the following tests is successful:

$$\begin{array}{ll} \text{Test 1 :} & \max |F_i(x_1, \ldots, x_n)| \le \mathtt{FTOL}, \\ \text{Test 2 :} & \max |x_i - x_i'| \le \mathtt{XTOL} * \max |x_i|, \end{array}$$

where the maxima are taken over $1 \le i \le n$.

On computers other than CDC and Cray, only the double-precision version DSNLEQ is available. On CDC and Cray computers, only the single-precision version RSNLEQ is available.

Structure:

SUBROUTINE subprograms
User Entry Names : RSNLEQ, DSNLEQ
Obsolete User Entry Names : SNLEQ ≡ RSNLEQ
Files Referenced : Unit 6
External References: User-supplied SUBROUTINE subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tSNLEQ(N,X,F,FTOL,XTOL,MAXF,IPRT,INFO,SUB,W)
```

N (INTEGER) Number n of equations and variables.

X (type according to t) One-dimensional array of length ≥ N. On entry, X(i), (i = 1, ..., N), must contain an estimate to a solution x_i^* of the system of equations. On exit, X(i) contains the final estimate to x_i^*.

F (type according to t) One-dimensional array of length ≥ N. On exit, F(i), (i = 1, ..., N), contains the final value of the residual F_i(X(1), ..., X(N)).

FTOL (type according to t) Accuracy parameter for Test 1.

XTOL (type according to t) Accuracy parameter for Test 2.

MAXF (INTEGER) Maximum permitted number of iterations, where each iteration involves N calls to the user-supplied subroutine SUB. The recommended value for MAXF is 50*(N+3).

IPRT (INTEGER) If IPRT = 0 no intermediate results are printed.
If IPRT = 1 the values of i and X(i), (i = 1, 2, ..., n), are printed after each iteration.

INFO (INTEGER) On exit, the value of INFO shows the reason why execution was terminated as follows:

0 Unacceptable input arguments (N < 1 or FTOL ≤ 0 or XTOL ≤ 0).

1	Test 1 is successful.
2	Test 2 is successful.
3	Test 1 and Test 2 are both successful.
4	Number of iterations is $\geq$ MAXF.
5	Approximate (finite difference) Jacobian matrix is singular
6	Iterations are not making good progress.
7	Iterations are diverging.
8	Iterations are converging, but either (i) XTOL is too small, or (ii) convergence is very slow because the Jacobian is nearly singular near x_i^* or because the variables x_i are badly scaled.

SUB Name of a user-supplied SUBROUTINE subprogram, declared EXTERNAL in the calling program.

W (type according to t) Array containing at least N*(N+3) elements required as working-space.

The user-supplied SUBROUTINE subprogram SUB should be of the form

```
SUBROUTINE SUB(N,X,F,K)
DIMENSION X(*),F(*)
...
```

Statements which set F(K) equal to the value of $F_{\mathtt{K}}(\mathtt{X(1)}, ..., \mathtt{X(N)})$ without changing any other element of array F.

```
...
RETURN
END
```

where X and F are of type t.
Subroutine SUB should not change the value of the argument K unless the user wants to terminate the execution of tSNLEQ, in which case K should be set equal to a negative integer, whose value will be copied into argument INFO of tSNLEQ before exit.

Method:

A modification of Brent's method as described in Ref. 1.

Error handling:

See description of argument INFO.

Notes:

1. Whenever possible the equations $F_i = 0$ should be numbered in order of increasing nonlinearity.
2. These subroutines do not use any techniques which attempt to obtain global convergence. Convergence may therefore fail to occur if the initial estimate is too far from an exact solution.

Source:

This subroutine has been adapted from the Fortran program published in Ref. 2.

References:

1. J.J. Moré and M.Y. Cosnard, Numerical solution of nonlinear equations, ACM Trans. Math. Software **5** (1979) 64–85.
2. J.J. Moré and M.Y. Cosnard, Algorithm 554 BRENTM, A FORTRAN subroutine for the numerical solution of systems of nonlinear equations, Collected Algorithms from CACM (1980).

•

Author(s) : H.-H. Umstätter **Library:** MATHLIB
Submitter : K.S. Kölbig **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Zeros of a Real Polynomial

Subroutine subprogram `RMULLZ` and `DMULLZ` compute the zeros of the polynomial

$$P(z) \;=\; a_0 z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n$$

of degree n with real coefficients a_k and $a_0 \neq 0$.
On computers other than CDC or Cray, only the double-precision version `DMULLZ` is available. On CDC and Cray computers, only the single-precision version `RMULLZ` is available.

Structure:

SUBROUTINE subprograms
User Entry Names : `RMULLZ`, `DMULLZ`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
CALL tMULLZ(A,N,MAXIT,Z)
```

`A` (type according to `t`) One-dimensional array of dimension `(0:d)`, where `d` $\geq$ `N`, containing the coefficients a_k, $(k = 0, 1, \ldots, n)$.

`N` (`INTEGER`) The degree n.

`MAXIT` (`INTEGER`) The maximum number of iterations permitted.

`Z` (`COMPLEX` for `t` = `R`, `COMPLEX*16` for `t` = `D`) One-dimensional array of length $\geq$ `N`. On exit, `Z(i)` contains an approximation to the zero z_i, listed in roughly decreasing order of $|z_i|$.

Method:

The method of Muller (see Ref. 1) is used. This is based on iterated inverse quadratic interpolation followed by deflation to remove each zero as found.

Accuracy:

For well-conditioned polynomials (i.e. polynomials whose zeros are not unduly sensitive to small errors in the coefficients), the relative error of a computed zero of multiplicity m is of order $10^{-d/m}$ where d is the machine precision expressed in decimal digits. For $m > 1$, the m approximations to the single multiple zero are uniformly distributed on a small circle of radius of order $10^{-d/m}$ around the exact zero. Therefore, if the polynomial is well-conditioned, the true value of the multiple zero will be close to the centre $(z_{k+1} + \ldots + z_{k+m})/m$ of this circle.

Error handling:

Error `C202.1`: $a_0 = 0$.
Error `C202.2`: The number of iterations exceeds `MAXIT`.
In both cases, a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called. If the number of iterations exceeds `MAXIT`, those zeros which have not been found are set to 10^{20}.

Notes:

For difficult cases which lead to too many iterations the following transformations may be applied, singly or together, to obtain a better-conditioned polynomial:

1. Reverse the order of the coefficients to obtain a polynomial whose zeros are z_i^{-1}.
2. If the zeros z_i are clustered, or are too unsymmetrically positioned with respect to the origin, compute by synthetic division (see Ref. 3) the coefficients of the polynomial whose argument is $w = z - \widehat{z}$, where $\widehat{z} = -a_1/(na_0)$ is the arithmetic mean of the zeros. The mean of the zeros of this new polynomial is situated at the origin, which is where the subprogram starts searching. Then, provided $|w_i| < |\widehat{z}|$ for most i, $z_i = w_i + \widehat{z}$ will be more accurate zeros.

References:

1. D.E. Muller, A method for solving algebraic equations using an automatic computer, MTAC (later renamed Math. Comp.) **10** (1956) 208–215.
2. J.W. Daniel, Correcting approximations to multiple roots of polynomials, Numer. Math. **9** (1966) 99–102.
3. F.B. Hildebrand, Introduction to numerical analysis, McGraw-Hill, New York (1956), Section 10.9.

•

Author(s) : T. Pomentale
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 20.04.1970
Revised: 15.03.1993

Zero of a Function of One Real Variable

Subroutine subprograms RZERO and DZERO compute, to an attempted specified accuracy, a zero of a real-valued function $f(x)$ lying in a given interval $[a, b]$, where $f(a) * f(b) \leq 0$.

On CDC and Cray computers, the double-precision version DZERO is not available.

Structure:

SUBROUTINE subprograms
User Entry Names: RZERO, DZERO
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035), user-supplied FUNCTION subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tZERO(A,B,X0,R,EPS,MAXF,F)
```

A,B (type according to t) On entry, A and B must specify the end-points of the search interval. Unchanged on exit.

X0 (type according to t) On exit, X0 is the computed approximation to a zero x_0 of the function $f(x)$.

R (type according to t) On exit, the value of R is such that X0 $-x_0 <$ R, unless an error condition is detected (see **Error Handling**).

EPS (type according to t) On entry, EPS must be equal to the accuracy parameter (see **Accuracy**). Unchanged on exit.

MAXF (INTEGER) On entry, MAXF must be equal to the maximum permitted number of references to the function F within the iteration loop. Unchanged on exit.

F (type according to t) Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program.

The user-supplied function subprogram F must be of the form FUNCTION F(X,I) and must set F(X) $= f($X$)$. The INTEGER argument I is set by RZERO before each reference to F as follows:

I = 1 First reference.

I = 2 Subsequent references.

I = 3 Final reference, before a normal (R > 0) exit.

Method:

A mixed strategy is used, based on the Muller method of parabolic interpolation supplemented by bisection.

Accuracy:

The routine tries to compute a value `X0` such that

$$|\mathtt{X0} - x_0| \leq (1 + \mathtt{X0}) * \mathtt{EPS}.$$

If this accuracy is obtained with fewer than `MAXF` references to the function `F` within the iteration loop, the subroutine exits with `R` positive.

Error handling:

Error `C205.1`: $\mathtt{F}(\mathtt{A}, 1) * \mathtt{F}(\mathtt{B}, 1) > 0$. `X0` is set equal to zero and `R` is set equal to $-2|\mathtt{B} - \mathtt{A}|$.
Error `C205.2`: The number of calls to `F` exceeds `MAXF`. `X0` is set equal to zero and `R` is set to $-|\mathtt{B} - \mathtt{A}|/2$.
A message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.
•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 15.01.1988
Language : Fortran | **Revised:** 01.12.1994

Roots of a Cubic Equation

Subroutine subprograms `RRTEQ3` and `DRTEQ3` compute the three roots of

$$x^3 + rx^2 + sx + t = 0 \qquad (*)$$

for real coefficients r, s, t.

On computers other than CDC or Cray, only the double-precision version `DRTEQ3` is available. On CDC and Cray computers, only the single-precision version `RRTEQ3` is available.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RRTEQ3`, `DRTEQ3`
Obsolete User Entry Names: `RTEQ3` $\equiv$ `RRTEQ3`

Usage:

For `t` $=$ `R` (type `REAL`), `t` $=$ `D` (type `DOUBLE PRECISION`),

```
CALL tRTEQ3(R,S,T,X,D)
```

`R,S,T` (type according to `t`) Coefficients r, s, t in $(*)$.

`X` (type according to `t`) One-dimensional array of length ≥ 3. On exit, `X` is set as described below.

`D` (type according to `t`) On exit, `D` is set to the value of the discriminant of $(*)$:
$>$ `0` : One real root `X(1)` and two complex conjugate roots `X(2)` $+ i$`X(3)`, `X(2)` $- i$`X(3)`;
$=$ `0` : Three real roots `X(1)`, `X(2)`, `X(3)`, where at least `X(2)` $=$ `X(3)`;
$<$ `0` : Three distinct real roots `X(1)`, `X(2)`, `X(3)`.

Method:

The classical method of Tartaglia-Vieta is used. In certain cases, the solutions are improved by Newton iteration.

Accuracy:

Depends on the coefficients r, s, t. The values of `X(1)`, `X(2)`, `X(3)` and of `D` may be inaccurate if `|D|` is very small, even in the case of “exact” coefficients.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.01.1988
Language : Fortran **Revised:** 01.12.1994

Roots of a Quartic Equation

Subroutine subprograms RRTEQ4 and DRTEQ4 compute the four roots of

$$x^4 + ax^3 + bx^2 + cx + d = 0 \qquad (*)$$

for real coefficients a, b, c, d.

On computers other than CDC or Cray, only the double-precision version DRTEQ4 is available. On CDC and Cray computers, only the single-precision version RRTEQ4 is available.

Structure:

SUBROUTINE subprograms
User Entry Names: RRTEQ4, DRTEQ4
Obsolete User Entry Names: RTEQ4 $\equiv$ RRTEQ4
External References: RRTEQ3 (C207), DRTEQ3 (C207)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tRTEQ4(A,B,C,D,Z,DC,MT)
```

A,B,C,D (type according to t) Coefficients a, b, c, d in $(*)$.

Z (COMPLEX for t = R, COMPLEX*16 for t = D) One-dimensional array of length ≥ 4. On exit, Z contains the roots of $(*)$.

DC (type according to t) On exit, DC is set to the value of the discriminant of the cubic resolvent of $(*)$.

MT (INTEGER) On exit, MT specifies the type of the roots:
= 1 : Four real roots in Z(1), . . . , Z(4);
= 2 : Two pairs of complex conjugate roots, one pair in Z(1),Z(2), the other in Z(3),Z(4);
= 3 : Two real roots in Z(1),Z(2), and one pair of complex conjugate roots in Z(3),Z(4).

Method:

The equation is solved by the classical procedure, i.e., by solving its cubic resolvent and by combining the square roots of these solutions appropriately.

Accuracy:

Depends on the coefficients a, b, c, d. The values of Z(1), . . . , Z(4) and of DC may be inaccurate if |DC| is very small. MT may be uncertain in such cases.

•

Author(s) : T. Pomentale
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 07.06.1992
Revised:

Zeros of a Complex Polynomial

Subroutine subprograms CPOLYZ and WPOLYZ compute the zeros of the polynominal

$$P(z) = c_0z^n + c_1z^{n-1} + \cdots + c_{n-1}z + c_n$$

of degree n with complex coefficients c_k and $c_0 \neq 0$.
On computers other than CDC or Cray, only the double-precision version WPOLYZ is available. On CDC and Cray computers, only the single-precision version CPOLYZ is available.

Structure:

SUBROUTINE subprograms
User Entry Names: CPOLYZ, WPOLYZ
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

For t = C (type COMPLEX), t = W (type COMPLEX*16),

```
CALL tPOLYZ(C,N,MAXIT,Z,R)
```

C (type according to t) One-dimensional array of dimension (0:d), where d $\geq$ N, containing the coefficients c_k, $(k = 0, 1, \ldots, n)$.

N (INTEGER) The degree n.

MAXIT (INTEGER) The maximum number of iterations permitted.

Z (type according to t) One-dimensional array of length $\geq$ N. On entry, Z(1), ..., Z(N) must contain starting approximations for the zeros z_i. If no starting approximations are available, the Z(i) should be set to zero. On exit, Z(i) contains an approximation to the zero z_i.

R (REAL for t = C, DOUBLE PRECISION for t = W) One-dimensional array of dimension $\geq$ N. On exit, R(1), ..., R(N) contain an estimated radius r_i of a circle centered at Z(i) within which the true zero z_i is expected to lie.

Notes:

Note that, because of accumulation of rounding errors, unreliable results can be obtained for large n even for well-conditioned polynomials.

Error handling:

Error C209.1: $c_0 = 0$.
Error C209.2: The number of iterations exceeds MAXIT.
Error C209.3: An estimated radius r_i cannot be computed for a certain value of i.
In all cases, a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. T. Pomentale, Homotopy iterative methods for polynomial equations, J. Inst. Maths. Applics. **13** (1974) 201–213.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 07.06.1992
Revised:

Number of Zeros of a Complex Function

Function subprogram `NZERFZ` calculates the number of zeros of a complex function $f(z)$ inside a closed polygon in the complex z-plane. $f(z)$ must be analytic inside this polygon.

Structure:

`FUNCTION` subprogram
User Entry Names: `NZERFZ`
Files Referenced : `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035), User-supplied `FUNCTION` subprogram

Usage:

In any arithmetic expression,

```
NZERFZ(F,ZP,N)
```

has a value equal to the number of zeros inside the defined polygon.

`F` Name of a user-supplied `FUNCTION` subprogram, declared `EXTERNAL` in the calling program. This subprogram must set `F(Z)` $= f($`Z`$)$.

`ZP` One-dimensional array of length $\geq$ `N` containing the vertices of the polygon in the z-plane.

`N` Number of vertices.

`F`, `ZP` and `Z` (in `F`) are of type `COMPLEX*16` on computers other than CDC or Cray, and of type `COMPLEX` on CDC and Cray computers.

Method:

The logarithmic residual (winding number) of $f(z)$ is found by integrating $f'(z)/f(z)$ numerically along the edges of the polygon.

Notes:

No zero or singularity of $f(z)$ should lie on or too near the polygon. The edges of the polygon should not cross each other. Numerically unstable functions (e.g. polynomials of high degree) can result in unreliable values or in timing problems.

Error handling:

Error `C210.1`: The integration is not successful. This often indicates that the polygon passes through or too near to a zero or singularity. The function value is set to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

•

Author(s) : G.A. Erskine
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB or Fortran Compiler Library
Submitted: 20.04.1970
Revised: 07.06.1992

Error Function and Complementary Error Function

Function subprograms ERF, ERFC and DERF, DERFC compute the error and complementary error functions

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt, \qquad \mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\, dt,$$

defined for all values of the real argument x.
On CDC and Cray computers, the double-precision versions DERF and DERFC are not available.

Structure:

FUNCTION subprograms
User Entry Names: ERF, ERFC, DERF, DERFC

Usage:

In any arithmetic expression,

ERF(X)	or	DERF(X)	has the value	erf(X),
ERFC(X)	or	DERFC(X)	has the value	erfc(X),

where ERF, ERFC, are of type REAL, DERF, DERFC, are of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Computation by rational Chebyshev approximation.

Accuracy:

The system-supplied versions (see **Notes**) have full machine accuracy. The CERN-supplied versions of ERF and ERFC have full single-precision accuracy (slightly less on CDC and Cray computers). The CERN-supplied versions of DERF and DERFC have an accuracy of 15 significant digits.

Notes:

On some computers, one or both of these functions is available in the system-supplied Fortran mathematical library. In this case the system-supplied version will be loaded instead of the CERN version.

References:

1. W.J. Cody, Rational Chebyshev approximations for the error function, Math. Comp. **22** (1969) 631–637.

•

Author(s) : G.A. Erskine
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 07.06.1992
Revised:

Normal Frequency Function

Function subprograms FREQ and DFREQ compute the normal frequency function

$$\mathrm{freq}(x) \quad = \quad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2}\,dt,$$

defined for all values of the real argument x.
On CDC and Cray computers, the double-precision version DFREQ is not available.

Structure:

FUNCTION subprograms
User Entry Names: FREQ, DFREQ

Usage:

In any arithmetic expression,

FREQ(X) or DFREQ(X) has the value freq(X),

where FREQ is of type REAL, DFREQ is of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Computation by rational Chebyshev approximation for the error function, using the formula

$$\mathrm{freq}(x) \;=\; \begin{cases} \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\,(x/\sqrt{2}) & (x \geq 0), \\ \frac{1}{2}\,\mathrm{erfc}\,(|x|/\sqrt{2}) & (x < 0). \end{cases}$$

Accuracy:

FREQ has full single-precision accuracy (slightly less on CDC and Cray computers). DFREQ has an accuracy of 15 significant digits.

References:

1. W.J. Cody, Rational Chebyshev approximations for the error function, Math. Comp. **22** (1969) 631–637.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB or Fortran Computer Library
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:** 15.03.1993

Gamma Function for Positive Argument

Function subprograms GAMMA, DGAMMA and QGAMMA calculate the gamma function

$$\Gamma(x) \;=\; \int_0^\infty e^{-t} t^{x-1} dt \qquad (x > 0)$$

for real argument $x > 0$.
The quadruple-precision version QGAMMA is available only on computers which support a REAL*16 Fortran data type.

Structure:

FUNCTION subprograms
User Entry Names: GAMMA, DGAMMA, QGAMMA
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

GAMMA(X), DGAMMA(X) or QGAMMA(X) has the value Γ(X),

where GAMMA is of type REAL, DGAMMA is of type DOUBLE PRECISION, QGAMMA is of type REAL*16, and X has the same type as the function name.

Method:

Approximation by truncated Chebyshev series and functional relations.

Accuracy:

The system-supplied version (see **Notes**) has full machine accuracy. The CERN version of GAMMA (except on CDC and Cray computers) has full single-precision accuracy. The CERN version of DGAMMA, QGAMMA (and of GAMMA, DGAMMA on CDC and Cray computers) have an accuracy which is approximately one digit less than machine precision.

Error handling:

Error C302.1: X $\leq$ 0. The function value is set equal to zero, and a message is written on Unit 6 unless subroutine MTLSET (N002) has been called.

Notes:

If the function GAMMA or DGAMMA is available in the system-supplied Fortran mathematical library, the system-supplied function will be loaded instead of the CERN version.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press, New York 1975) 4.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 07.06 1992
Revised:

Gamma Function for Real Argument

Function subprograms GAMMF and DGAMMF calculate the gamma function

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt \qquad (x > 0), \qquad \Gamma(x) = \frac{\pi}{\Gamma(1-x)\sin \pi x} \qquad (x < 0)$$

for real argument $x \neq -n$, $(n = 0, 1, 2, \cdots)$.
On CDC and Cray computers, the double-precision version DGAMMF is not available.

Structure:

FUNCTION subprograms
User Entry Names: GAMMF, DGAMMF
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

GAMMF(X) or DGAMMF(X) has the value $\Gamma($X$)$,

where GAMMF is of type REAL, DGAMMF is of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Approximation by truncated Chebyshev series and functional relations.

Accuracy:

GAMMF (except on CDC and Cray computers) has full single-precision accuracy. DGAMMF (and of GAMMF on CDC and Cray computers) has an accuracy which is approximately one digit less than machine precision.

Error handling:

Error C303.1: X $= -n$, $(n = 0, 1, 2, \cdots)$. The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press, New York 1975) 4.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB or Fortran Compiler Library
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:** 15.03.1993

Logarithm of the Gamma Function

Function subprograms ALGAMA, DLGAMA and QLGAMA compute the logarithm of the gamma function

$$\ln \Gamma(x) = \ln \int_0^\infty e^{-t} t^{x-1}\, dt \qquad (x > 0)$$

for real argument $x > 0$.
The quadruple-precision version QLGAMA is available only on computers which support a REAL*16 Fortran data type.

Structure:

FUNCTION subprograms
User Entry Names: ALGAMA, DLGAMA, QLGAMA
Obsolete User Entry Names: ALOGAM $\equiv$ ALGAMA, DLOGAM $\equiv$ DLGAMA
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

ALGAMA(X), DLGAMA or QLGAMA(X) has the value $\ln \Gamma(\mathtt{X})$,

where ALGAMA is of type REAL, DLGAMA is of type DOUBLE PRECISION, QLGAMA is of type REAL*16, and X has the same type as the function name.

Method:

Rational approximations.

Accuracy:

The system-supplied version (see **Notes**) has full machine accuracy. The CERN-supplied version of ALGAMA (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, the CERN-supplied versions of DLGAMA, QLGAMA (and of ALGAMA, DLGAMA on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error C304.1: X $\leq$ 0. The function value is set equal to zero, and a message is written on on Unit 6, unless subroutine MTLSET (N002) has been called.

Notes:

If the function ALGAMA or DLGAMA is available in the system-supplied Fortran mathematical library, the system-supplied function will be loaded instead of the CERN version.

References:

1. W.J. Cody and K.E. Hillstrom, Chebyshev approximations for the natural logarithm of the gamma function, Math. Comp. **21** (1967) 198–203.
2. J.F. Hart et al., Computer approximations (John Wiley Sons, New York 1968) 287.

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 02.05.1966
Language : Fortran | **Revised:** 15.03.1993

Gamma Function for Complex Argument

Function subprograms CGAMMA and WGAMMA calculate the gamma function

$$\Gamma(z) \;=\; \int_0^\infty e^{-t} t^{z-1} dt \qquad (\text{Re } z > 0)$$

for complex arguments $z \neq -n$, $(n = 0, 1, 2, \cdots)$.
The double-precision version WGAMMA is available only on computers which support a COMPLEX*16 Fortran data type.

Structure:

FUNCTION subprograms
User Entry Names: CGAMMA, WGAMMA
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

CGAMMA(Z) or WGAMMA(Z) has the value Γ(Z),

where CGAMMA is of type COMPLEX, WGAMMA is of type COMPLEX*16, and Z has the same type as the function name.

Method:

The method is described in Ref. 1.

Accuracy:

CGAMMA (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument Z, WGAMMA (and CGAMMA on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error C305.1: Z $= -n$, $(n = 0, 1, 2, \cdots)$. The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, The special functions and their approximations, v.II, (Academic Press, New York 1969) 304–305

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.03.1994
Language : Fortran **Revised:**

Logarithm of the Gamma Function for Complex Argument

Function subprograms `CLGAMA` and `WLGAMA` calculate the logarithm of the gamma function

$$\ln\Gamma(z) \;=\; \ln\int_0^\infty e^{-t}t^{z-1}\,dt \qquad (\mathrm{Re}\; z > 0)$$

for complex $z \neq -n$, $(n = 0, 1, 2, \ldots)$. The imaginary part Im $\ln\Gamma(z)$ is calculated in such a way that it is continuous for $|\arg z| < \pi$, i.e. it is not taken mod 2π.

The double-precision version `WLGAMA` is available only on computers which support a `COMPLEX*16` Fortran data type.

Structure:

`FUNCTION` subprograms
User Entry Names: `CLGAMA`, `WLGAMA`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`CLGAMA(Z)` or `WLGAMA(Z)` has the value $\ln\Gamma(\mathtt{Z})$,

where `CLGAMA` is of type `COMPLEX`, `WLGAMA` is of type `COMPLEX*16`, and `Z` has the same type as the function name.

Method:

The method is described in Ref. 1.

Accuracy:

`CLGAMA` (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument `X`, `WLGAMA` (and `CLGAMA` on CDC and Cray computers) has an accuracy of approximately two significant digits less than the machine precision.

Error handling:

Error `C306.1`: $\mathtt{Z} = -n$, $(n = 0, 1, 2, \cdots)$. The function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

References:

1. K.S. Kölbig, Programs for computing the logarithm of the gamma function, and the digamma function, for complex argument, Computer Phys. Comm. **4** (1972) 221–226.

•

Author(s) : I.J. Thompson, A.R. Barnett
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 15.01.1988
Revised: 15.11.1995

Coulomb Wave, Bessel, and Spherical Bessel Functions for Complex Argument(s) and Order

Subroutine subprograms CCLBES and WCLBES calculate any one of the following sequences of functions:

1. Regular and irregular Coulomb wave functions $F_{\lambda+n}(\eta,z),\ G_{\lambda+n}(\eta,z)$
   and their first derivatives with respect to z, $F'_{\lambda+n}(\eta,z),\ G'_{\lambda+n}(\eta,z)$,
   or simple combination of these;
2. Spherical Bessel functions $j_{\lambda+n}(z),\ y_{\lambda+n}(z)$
   and their first derivatives with respect to z, $j'_{\lambda+n}(z),\ y'_{\lambda+n}(z)$,
   or simple combination of these (spherical Hankel functions);
3. Bessel functions $J_{\lambda+n}(z),\ Y_{\lambda+n}(z)$
   and their first derivatives with respect to z, $J'_{\lambda+n}(z),\ Y'_{\lambda+n}(z)$,
   or simple combination of these (Hankel functions);
4. Modified Bessel functions $I_{\lambda+n}(z),\ K_{\lambda+n}(z)$
   and their first derivatives with respect to z, $I'_{\lambda+n}(z),\ K'_{\lambda+n}(z)$;

for complex arguments η, z, complex order λ, and $n = 0, 1, \dots, N$.

The double-precision version WCLBES is available only on computers which support a COMPLEX*16 Fortran data type.

Structure:

SUBROUTINE subprograms
User Entry Names: CCLBES, WCLBES
Internal Entry Names: C309R1, C309R2, C309R3, C309R4, C309R5, C309R6, C309R7, C309R8
Files Referenced: Unit 6
External References: CLGAMA (C306), WLGAMA (C306), CPSIPG (C317), WPSIPG (C317)

Usage:

For t = C (type COMPLEX), t = W (type COMPLEX*16),

```
CALL tCLBES(Z,ETA,ZLMIN,NL,F,G,FP,GP,SIG,KFN,MODE,JFAIL,JPR)
```

Z — (type according to t) Argument $z \neq 0$.

ETA — (type according to t) Argument η (ignored if KFN > 0).

ZLMIN — (type according to t) Order λ_{min} of the first function in the computed sequence.

NL — (INTEGER) Specifies the order λ_{min} + NL of the last function in the computed sequence. (NL $\geq$ 0).

F,G,FP,GP — (type according to t) One-dimensional arrays with dimension (0:d) where d is in each case $\geq$ NL + 1. On exit, each of F(n),G(n),FP(n),GP(n) may contain the value of a function of order $\lambda_{min} + n$, or its first order derivative, ($n = 0, 1, \dots,$ NL), as specified jointly by KFN and |MODE|.

SIG (type according to t) One-dimensional array with dimension (0:d), where d $\geq$ NL + 1. On exit, provided KFN $=$ 0, SIG(n) contains the Coulomb phase shift $\sigma(\eta)$ for $\lambda = \lambda_{min} + n$, $(n = 0, 1, \ldots, \text{NL})$.

KFN (INTEGER) Specifies, in conjunction with the absolute value of MODE, the type of functions which are stored.

MODE (INTEGER) The absolute value of MODE specifies, in conjunction with KFN, the type of function which are stored, and also specifies which of the arrays F,G,FP,GP are in fact set to meaningful values. The sign of MODE specifies whether or not the functions are multiplied by a scaling factor.

JFAIL (INTEGER) On exit, JFAIL is set to zero if no error condition is detected. Otherwise JFAIL is set as described under **Error handling**.

JPR (INTEGER)

$=$ 0 : Suppress printing of error messages.

$=$ 1 : Print error messages.

The type of function which is stored in array F depends only on KFN, while the type of function which is stored in array G depends both on KFN and on |MODE|. Arrays FP and GP (if set) contain the first order derivatives with respect to z of the functions in F and G, respectively. Using the abbreviations ($i = \sqrt{-1}$)

$$\begin{array}{llll} F_\lambda \equiv F_\lambda(\eta, z), & G_\lambda \equiv G_\lambda(\eta, z), & H_\lambda^\pm & \equiv G_\lambda \pm iF_\lambda, \\ j_\lambda \equiv j_\lambda(z), & y_\lambda \equiv y_\lambda(z), & h_\lambda^{(1,2)} & \equiv j_\lambda \pm iy_\lambda, \\ J_\lambda \equiv J_\lambda(z), & Y_\lambda \equiv Y_\lambda(z), & H_\lambda^{(1,2)} & \equiv J_\lambda \pm iY_\lambda, \\ I_\lambda \equiv I_\lambda(z), & K_\lambda \equiv K_\lambda(z), & & \end{array}$$

the choice of function is given by the following table:

Array	|MODE|	KFN			
		-1 or 0	1	2	3
F	all values	F_λ	j_λ	J_λ	I_λ
G	$1, 2, 3, 4$	G_λ	y_λ	Y_λ	K_λ
	$11, 12$	H_λ^+	$h_\lambda^{(1)}$	$H_\lambda^{(1)}$	$-$
	$21, 22$	H_λ^-	$h_\lambda^{(2)}$	$H_\lambda^{(2)}$	$-$

If KFN=0 the phase shifts $\sigma(\eta)$ are stored in array SIG. Otherwise SIG is not set.

Which of arrays F,G,FP,GP are in fact set is determined by |MODE| according to the following table:

|MODE|	F	G	FP	GP
1, 11, 21	set	set	set	set
2, 12, 22	set	set	-	-
3	set	-	set	-
4	set	-	-	-

In both the tables above, a dash indicates that the corresponding array does not contain meaningful values on exit. These arrays are, however, used internally as working space, and must therefore be dimensioned correctly. The sign of MODE specifies whether or not the functions are to be multiplied by a scaling factor, depending only on z, which will bring their values closer to unity when $|z|$ is large, or η is small and $|\lambda| < |z|$. The same scaling factor is applied to the first order derivatives in FP or GP as is applied to the functions in F or G, respectively.

`MODE` > 0 : No scaling factor.

`MODE` < 0 : Let $S = \mathrm{Im}(z)$ if `KFN` < 3, $S = \mathrm{Re}(z)$ if `KFN` $= 3$; then the scaling factors for `F` and `G` are

$$
\begin{array}{lll}
\texttt{F}: & \exp(-|S|) & \{F, j, J, I\} \\
\texttt{G}: & \exp(-|S|) & \{G, y, Y\} \\
 & \exp(S) & \{H^+, h^{(1)}, H^{(1)}, K\} \\
 & \exp(-S) & \{H^-, h^{(2)}, H^{(2)}\}.
\end{array}
$$

Method:

The method is described in the References.

Restrictions:

See Ref. 1, in particular Sect. 4.

Accuracy:

The absolute values of the results are usually accurate to within two or three decimal digits of the machine precision. For details of exceptions see Ref. 1, Sect. 4.

Error handling:

If an error condition is detected, `JFAIL` is set to one of the following values and a message is printed if `JPR` $= 1$.

- > 0 An arithmetic error occurred during the final recursion. Correct results are available up to and including subscript value `NL-JFAIL-1`.
- -1 One of the continued fraction calculations failed or there was an arithmetic error before any results could be calculated.
- -2 Argument out of range.
- -3 One or more functions corresponding to λ_{min} could not be calculated. Some values corresponding to $\lambda > \lambda_{min}$ may be correct.
- -4 Excessive internal cancellation probably renders the result meaningless.

Source:

This program package is a modified version of the CPC Program Library package `COULCC` (see Ref. 1). The changes are formal, not computational.

References:

1. I.J. Thompson and A.R. Barnett, COULCC: A continued-fraction algorithm for Coulomb functions of complex order with complex arguments, Comput. Phys. Comm. **36** (1985) 363–372.
2. I.J. Thompson and A.R. Barnett, Coulomb and Bessel functions of complex arguments and order, J. Comput. Phys. **64** (1986) 490–509.

Long Write-up:

A copy of Ref. 1 is available in the Program Library Office.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 18.10.1967
Revised: 15.03.1993

Bessel Functions J and Y of Orders Zero and One

Function subprograms BESJ0, BESJ1, BESY0, BESY1 and DBESJ0, DBESJ1, DBESY0, DBESY1 calculate the Bessel functions

$$J_0(x),\ J_1(x),\ Y_0(x),\ Y_1(x)$$

for real arguments x, where $x > 0$ for $Y_0(x)$ and $Y_1(x)$.

On CDC and Cray computers, the double-precision versions DBESJ0 etc. are not available.

Structure:

FUNCTION subprograms
User Entry Names: BESJ0, BESJ1, BESY0, BESY1, DBESJ0, DBESJ1, DBESY0, DBESY1
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

BESJ0(X)	or	DBESJ0(X)	has the value	J_0(X),
BESJ1(X)	or	DBESJ1(X)	has the value	J_1(X),
BESY0(X)	or	DBESY0(X)	has the value	Y_0(X),
BESY1(X)	or	DBESY1(X)	has the value	Y_1(X),

where BESJ0 etc. are of type REAL, DBESJ0 etc. are of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Approximation by truncated Chebyshev series.

Accuracy:

BESJ0 etc. (except on CDC and Cray computers) have full single-precision accuracy. For most values of the argument X, DBESJ0 etc. (and BESJ0 etc. on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error C312.1: X $\leq$ 0 for $Y_0(x)$ or $Y_1(x)$. The function value is set equal to zero, and a message is written on Unit 6 unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, Mathematical functions and their approximations (Academic Press, New York 1975) 322–324.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran
Library: MATHLIB
Submitted: 07.12.1970
Revised: 15.03.1993

Modified Bessel Functions I and K of Orders Zero and One

Function subprograms BESI0, BESI1, BESK0, BESK1 and DBESI0, DBESI1, DBESK0, DBESK1 calculate the modified Bessel functions

$$I_0(x),\ I_1(x),\ K_0(x),\ K_1(x)$$

for real arguments x, where $x > 0$ for $K_0(x)$ and $K_1(x)$.

On CDC and Cray computers, the double-precision versions DBESI0 etc. are not available.

Structure:

FUNCTION subprograms
User Entry Names: BESI0, BESI1, BESK0, BESK1, EBESI0, EBESI1, EBESK0, EBESK1,
DBESI0, DBESI1, DBESK0, DBESK1, DEBSI0, DEBSI1, DEBSK0, DEBSK1
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

BESI0(X)	or	DBESI0(X)	has the value	$I_0(\mathtt{X})$,
BESI1(X)	or	DBESI1(X)	has the value	$I_1(\mathtt{X})$,
BESK0(X)	or	DBESK0(X)	has the value	$K_0(\mathtt{X})$,
BESK1(X)	or	DBESK1(X)	has the value	$K_1(\mathtt{X})$,
EBESI0(X)	or	DEBSI0(X)	has the value	$\exp(-\lvert\mathtt{X}\rvert) * I_0(\mathtt{X})$,
EBESI1(X)	or	DEBSI1(X)	has the value	$\exp(-\lvert\mathtt{X}\rvert) * I_1(\mathtt{X})$,
EBESK0(X)	or	DEBSK0(X)	has the value	$\exp(\lvert\mathtt{X}\rvert) * K_0(\mathtt{X})$,
EBESK1(X)	or	DEBSK1(X)	has the value	$\exp(\lvert\mathtt{X}\rvert) * K_1(\mathtt{X})$,

where BESI0 etc. are of type REAL, DBESI0 etc. are of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Approximation by rational functions (I for $|x| < 8$, K for $1 \le x \le 5$), by an algorithm based on power series (K for $0 < x < 1$), or else by truncated Chebyshev series.

Accuracy:

BESI0 etc. (except on CDC and Cray computers) have full single-precision accuracy. For most values of the argument X, DBESI0 etc. (and BESI0 etc. on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error C313.1: X $\le$ 0 for $K_0(x)$ or $K_1(x)$. The function value is set equal to zero, and a message is written on Unit 6 unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, Mathematical functions and their approximations (Academic Press, New York 1975) 329, 331, 363, 366.

2. N.M. Temme, On the numerical evaluation of the modified Bessel function of the third kind, J. Comp. Phys. **19** (1975) 324–337.

•

RRIZET	CERN Program Library	C315

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 07.06.1992
Revised:

Riemann Zeta Function

Function subprograms `RRIZET` and `DRIZET` calculate the Riemann zeta function

$$\zeta(x) \;=\; \sum_{k=1}^{\infty} k^{-x} \;=\; \frac{1}{\Gamma(x)} \int_0^{\infty} \frac{t^{x-1}}{e^t - 1} dt \qquad (x > 1)$$

for real arguments $x \neq 1$, where $\zeta(x)$ is defined by analytic continuation for $x < 1$. For $x = 1$, $\zeta(x)$ has a pole of order one.

On CDC and Cray computers, the double-precision version `DRIZET` is not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RRIZET`, `DRIZET`
Files Referenced: `Unit 6`
External References: `GAMMA` (C302), `DGAMMA` (C302), `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`RRIZET(X)` or `DRIZET(X)`

has the value $\zeta($`X`$)$ if `X` < 1, and $\zeta($`X`$) - 1$ if `X` > 1, where `RRIZET` is of type `REAL`, `DRIZET` is of type `DOUBLE PRECISION`, and where `X` has the same type as the function name.

Method:

Rational Chebyshev approximation. For $x < \frac{1}{2}$ the reflection formula for $\zeta(x)$ is used.

Accuracy:

`RRIZET` (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument `X`, `DRIZET` (and `RRIZET` on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error `C315.1`: `X` $= 1$. The function value is set to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

References:

1. W.J. Cody, K.E. Hillstrom, and H.C. Thather, Jr., Chebyshev approximations for the Riemann zeta function, Math. Comp. **25** (1971) 537–547.

•

RPSIPG	CERN Program Library	C316

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Psi (Digamma) and Polygamma Functions

Function subprograms RPSIPG and DPSIPG calculate either the logarithmic derivative of the gamma function (the psi, or digamma, function)

$$\psi(x) \equiv \psi^{(0)}(x) = \frac{d \ln \Gamma(x)}{dx}$$

or one of the other polygamma functions

$$\psi^{(k)}(x) = \frac{d^k}{dx^k}\psi(x) = \frac{d^{k+1}}{dx^{k+1}} \ln \Gamma(x)$$

for real arguments $x \neq -n$, $(n = 0, 1, 2, \ldots)$ and $k = 0, 1, 2, \ldots, 6$.
Note that the Euler constant $\mathbf{C} = -\psi(1) = 0.57721\ \ldots$ (also denoted by γ) and the Catalan constant $\mathbf{G} = \frac{1}{8}\big(\psi'(\frac{1}{4}) - \pi^2\big) = 0.91596\ \ldots$ can be calculated by using this subprogram.
On CDC and Cray computers, the double-precision version DPSIPG is not available.

Structure:

FUNCTION subprograms
User Entry Names: RPSIPG, DPSIPG
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

RPSIPG(X,K) or DPSIPG(X,K) has the value $\psi^{(\mathtt{K})}(\mathtt{X})$,

where RPSIPG is of type REAL, DPSIPG is of type DOUBLE PRECISION, and where X has the same type as the function name. K is of type INTEGER.

Method:

Rational Chebyshev approximation ($k = 0$), approximation by truncated Chebyshev series ($k > 0$), and functional relations.

Accuracy:

RPSIPG (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, DPSIPG (and RPSIPG on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error C316.1: $\mathtt{K} < 0$ or $\mathtt{K} > 6$.
Error C316.2: $\mathtt{X} = -n$, $(n = 0, 1, 2, \ldots)$.
In both cases, the function value is set to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. W.J. Cody, A.J. Strecock and H.C. Thather, Jr., Chebyshev approximations for the psi function, Math. Comp. **27** (1973) 123–127.
2. Y.L. Luke, Mathematical functions and their approximations (Academic Press, New York, l975) 5–6.

•

CPSIPG	CERN Program Library	C317

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.11.1995
Language : Fortran **Revised:**

Psi (Digamma) and Polygamma Functions for Complex Argument

Function subprograms CPSIPG and WPSIPG calculate either the logarithmic derivative of the gamma function (the psi, or digamma, function)

$$\psi(z) \equiv \psi^{(0)}(z) = \frac{d\ln\Gamma(z)}{dx}$$

or one of the other polygamma functions

$$\psi^{(k)}(z) = \frac{d^k}{dz^k}\,\psi(z) = \frac{d^{k+1}}{dz^{k+1}}\ln\Gamma(z)$$

for complex arguments $z \neq -n$, $(n = 0, 1, 2, \dots)$ and $k = 0, 1, 2, 3, 4$.
The double-precision version WPSIPG is available only on computers which support a COMPLEX*16 Fortran data type.

Structure:

FUNCTION subprograms
User Entry Names: CPSIPG, WPSIPG
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

CPSIPG(Z,K) or WPSIPG(Z,K) has the value $\psi^{(\mathtt{K})}(\mathtt{Z})$,

where CPSIPG is of type COMPLEX, WPSIPG is of type COMPLEX*16, and where Z has the same type as the function name. K is of type INTEGER.

Method:

The method for $\psi(z)$ described in Ref. 1 is adapted accordingly.

Accuracy:

CPSIPG (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument Z, WPSIPG (and CPSIPG on CDC and Cray computers) has an accuracy of approximately two significant digit less than the machine precision.

Error handling:

Error C317.1: K $<$ 0 or K $>$ 4.
Error C317.2: X $= -n$, $(n = 0, 1, 2, \dots)$.
In both cases, the function value is set to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. K.S. Kölbig, Programs for computing the logarithm of the gamma function, and the digamma function, for complex arguments, Computer Phys. Comm. **4** (1972) 221-226.

•

Author(s) : K.S. Kölbig, H.-H. Umstätter **Library:** MATHLIB
Submitter : **Submitted:** 30.01.1980
Language : Fortran **Revised:** 01.12.1994

Jacobian Elliptic Functions sn, cn, dn

Function subprograms `RELFUN` and `DELFUN` calculate, for real argument x and real modulus k, the Jacobian elliptic functions $\mathrm{sn}(x,k)$, $\mathrm{cn}(x,k)$ and $\mathrm{dn}(x,k)$. The function $\mathrm{sn}(x,k)$ is the inverse of the elliptic integral of the first kind and is defined implicitly by

$$x = \int_0^{\mathrm{sn}(x,k)} \frac{du}{\sqrt{(1-u^2)(1-k^2u^2)}} \qquad (k^2 \le 1).$$

The functions $\mathrm{cn}(x,k)$ and $\mathrm{dn}(x,k)$ are defined by

$$\mathrm{sn}^2(x,k) + \mathrm{cn}^2(x,k) = 1, \quad k^2\mathrm{sn}^2(x,k) + \mathrm{dn}^2(x,k) = 1, \quad \mathrm{cn}(0,k) = \mathrm{dn}(0,k) = 1.$$

This definition can be extended for $k^2 > 1$ (Ref. 2) by means of

$$\mathrm{sn}(x,k) = k_1\mathrm{sn}(kx,k_1), \quad \mathrm{cn}(x,k) = \mathrm{dn}(kx,k_1), \quad \mathrm{dn}(x,k) = \mathrm{cn}(kx,k_1),$$

where $k_1 = 1/k$. For $k = 0$ and $k^2 = 1$ these functions are elementary:

$$\mathrm{sn}(x,0) = \sin x, \quad \mathrm{cn}(x,0) = \cos x, \quad \mathrm{dn}(x,0) = 1,$$

$$\mathrm{sn}(x,\pm 1) = \tanh x, \quad \mathrm{cn}(x,\pm 1) = \mathrm{dn}(x,\pm 1) = \mathrm{sech}\, x.$$

Note that for $k^2 \neq 1$ the Jacobian elliptic functions are periodic (with respect to x) with period $4\mathrm{K}(k)$ if $k^2 < 1$ and $4k_1\mathrm{K}(k_1)$ if $k^2 > 1$, where $\mathrm{K}(k)$ is the complete elliptic integral of the first kind.

On CDC and Cray computers, the double-precision version `DELFUN` is not available.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RELFUN`, `DELFUN`
Obsolete User Entry Names: `ELFUN` $\equiv$ `RELFUN`

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
CALL tELFUN(X,AK2,SN,CN,DN)
```

`X` (type according to `t`) The argument x.

`AK2` (type according to `t`) The value of k^2 (the square of the modulus).

`SN` (type according to `t`) On exit, `SN` $= \mathrm{sn}($`X`$, k)$.

`CN` (type according to `t`) On exit, `CN` $= \mathrm{cn}($`X`$, k)$.

`DN` (type according to `t`) On exit, `DN` $= \mathrm{dn}($`X`$, k)$.

Method:

The sequence of the Gaussian arithmetic-geometric mean is used together with the Gauss transformation and, where appropriate, the Jacobi imaginary transformation. For values $\texttt{AK2} > 1$, the reciprocal modulus transformation is performed. For details see **References**.

Accuracy:

RELFUN (except on CDC and Cray computers) has full single-precision accuracy. For most values of the arguments, DELFUN (and RELFUN on CDC and Cray computers) has an accuracy of approximately two significant digits less than the machine precision.

Restrictions:

$|x| \leq 3\mathrm{K}(k)$ $(0 < k^2 < 1)$, $|x| \leq 3k_1\mathrm{K}(k_1)$ $(k^2 > 1)$, where $\mathrm{K}(x)$ is the complete elliptic integral of the first kind. (See entries RELIKC and DELIKC in RELI1C (C347)).

References:

1. M. Abramowitz and I.A. Stegun, ed., Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, Sections 16.12 and 17.6, 9th printing with corrections, (Dover, New York 1972).
2. H.E. Salzer, Quick calculation of Jacobian elliptic functions, Comm. ACM **5** (1962) 399.
3. L.V. King, On the dirct numerical calculation of elliptic functions and integrals, Cambridge Univ. Press (1924) 32.
4. D.J. Hofsommer and R.P. van de Riet, On the numerical calculation of elliptic integrals of the first and second kind and the elliptic functions of Jacobi, Numer. Math. **5** (1963) 291–302.
5. P.F. Byrd and M.D. Friedman, Handbook of elliptic integrals for engineers and scientists, 2nd ed., Springer-Verlag Berlin (1971).

•

Author(s) : H.-H. Umstätter
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 30.01.1980
Revised: 07.06.1992

Jacobian Elliptic Functions sn, cn, dn for Complex Argument

Function subprograms `CELFUN` and `WELFUN` calculate, for complex argument z and real modulus k, the Jacobian elliptic functions $\mathrm{sn}(z,k)$, $\mathrm{cn}(z,k)$ and $\mathrm{dn}(z,k)$. The function $\mathrm{sn}(z,k)$ is the inverse of the elliptic integral of the first kind and is defined implicitly by

$$z = \int_0^{\mathrm{sn}(z,k)} \frac{dw}{\sqrt{(1-w^2)(1-k^2w^2)}} \qquad (k^2 \le 1).$$

The functions $\mathrm{cn}(z,k)$ and $\mathrm{dn}(z,k)$ are defined by

$$\mathrm{sn}^2(z,k) + \mathrm{cn}^2(z,k) = 1, \quad k^2\mathrm{sn}^2(z,k) + \mathrm{dn}^2(z,k) = 1, \quad \mathrm{cn}(0,k) = \mathrm{dn}(0,k) = 1.$$

For $k = 0$ and $k^2 = 1$ these functions are elementary:

$$\mathrm{sn}(z,0) = \sin z, \quad \mathrm{cn}(z,0) = \cos z, \quad \mathrm{dn}(z,0) = 1,$$

$$\mathrm{sn}(z,\pm 1) = \tanh z, \quad \mathrm{cn}(z,\pm 1) = \mathrm{dn}(z,\pm 1) = \mathrm{sech}\, z.$$

Note that the Jacobian elliptic functions are doubly-periodic in the z-plane. For details see the relevant treatises or handbooks.

The double-precision version `WELFUN` is available only on computers which support a `COMPLEX*16` Fortran data type.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `CELFUN`, `WELFUN`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

For `t` = `C` (type `COMPLEX`), `t` = `W` (type `COMPLEX*16`),

```
CALL tELFUN(Z,AK2,SN,CN,DN)
```

`Z` (type according to `t`) The argument z.

`AK2` (`REAL` for `t` = `C`, `DOUBLE PRECISION` for `t` = `W`) The value of k^2 (the square of the modulus).

`SN` (type according to `t`) On exit, `SN` = $\mathrm{sn}(\mathtt{Z},k)$.

`CN` (type according to `t`) On exit, `CN` = $\mathrm{cn}(\mathtt{Z},k)$.

`DN` (type according to `t`) On exit, `DN` = $\mathrm{dn}(\mathtt{Z},k)$.

Method:

The Jacobian elliptic functions with complex argument $z = x + iy$ are computed from their representations in terms of Jacobian elliptic functions with real arguments x or y (Ref. 1, formula 125.01). See also the Short Write-up for `ELFUN` (C318).

Accuracy:

CELFUN (except on CDC and Cray computers) has full single-precision accuracy. For most values of the arguments, WELFUN (and CELFUN on CDC and Cray computers) has an accuracy of approximately two significant digits less than the machine precision.

Restrictions:

$|\mathrm{Re}\, z| \leq 3\mathrm{K}(k)$, $|\mathrm{Im}\, z| \leq 3\mathrm{K}(k')$ where $k' = \sqrt{1-k^2}$ is the complementary modulus, and $\mathrm{K}(x)$ is the complete elliptic integral of the first kind. (See entries RELIKC and DELIKC in RELI1C (C347)).

Error handling:

Error C320.1: $|$AK2$| > 1$. The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. P.F. Byrd and M.D. Friedman, Handbook of elliptic integrals for engineers and scientists, 2nd ed., Springer-Verlag Berlin (1971).

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 12.09.1985
Language : Fortran **Revised:** 15.03.1993

Nielsen's Generalized Polylogarithm

Function subprograms CGPLG and WGPLG calculate the complex-valued generalized polylogarithm function

$$S_{n,m}(x) \;=\; \frac{(-1)^{n+m-1}}{(n-1)!\,m!}\int_0^1 t^{-1}\ln^{n-1}t\,\ln^m(1-xt)\,dt \qquad (*)$$

for real arguments x and integer n and m satisfying $1 \le n \le 4,\ 1 \le m \le 4,\ n+m \le 5$; i.e., one of the functions $S_{1,1}$, $S_{1,2}$, $S_{2,1}$, $S_{1,3}$, $S_{2,2}$, $S_{3,1}$, $S_{1,4}$, $S_{2,3}$, $S_{3,2}$, $S_{4,1}$. If $x \le 1$, $S_{n,m}(x)$ is real, and the imaginary part is set equal to zero.

The double-precision version WGPLG is available only on computers which support a COMPLEX*16 Fortran data type.

Structure:

FUNCTION subprograms
User Entry Names: CGPLG, WGPLG
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

CGPLG(N,M,X) or WGPLG(N,M,X) has the value $S_{\mathtt{N},\mathtt{M}}(\mathtt{X})$,

where CGPLG is of type COMPLEX, WGPLG is of type COMPLEX*16, X is of type REAL for CGPLG and of type DOUBLE PRECISION for WGPLG, and where N and M are of type INTEGER.

Method:

The method is described in Ref. 1. Note that the imaginary part of the function defined as $S_{n,m}(x)$ in Ref. 1 has the opposite sign to the imaginary part of the function defined by (*). See Ref. 2.

Accuracy:

CGPLG (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, WGPLG (and CGPLG on CDC and Cray computers) has an accuracy of approximately two significant digits less than the machine precision. The loss of accuracy is greater when X is very close to 1.

Error handling:

Error C321.1: N, M < 1 or N, M > 4 or N + M > 5. The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. K.S. Kölbig, J.A. Mignaco and E. Remiddi, On Nielsen's generalized polylogarithms and their numerical calculation, BIT **10** (1970) 38–71.
2. K.S. Kölbig, Nielsen's generalized polylogarithms, SIAM J. Math. Anal. **17** (1986) 1232–1258.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.05.1987
Language : Fortran **Revised:** 01.12.1994

Fresnel Integrals

Function subprograms RFRSIN, RFRCOS and DFRSIN, DFRCOS calculate the Fresnel integrals

$$S(x) = \int_0^x \frac{\sin t}{\sqrt{t}} dt \qquad (x \geq 0), \qquad S(-x) = -S(x),$$
$$C(x) = \int_0^x \frac{\cos t}{\sqrt{t}} dt \qquad (x \geq 0), \qquad C(-x) = -C(x),$$

for real arguments x.

On CDC and Cray computers, the double-precision versions DFRSIN, DFRCOS are not available.

Structure:

FUNCTION subprograms
User Entry Names: RFRSIN, RFRCOS, DFRSIN, DFRCOS
Obsolete User Entry Names: FRSIN $\equiv$ RFRSIN, FRCOS $\equiv$ RFRCOS

Usage:

In any arithmetic expression,

RFRSIN(X)	or	DFRSIN(X)	has the value	S(X),
RFRCOS(X)	or	DFRCOS(X)	has the value	C(X),

where RFRSIN, RFRCOS are of type REAL, DFRSIN, DFRCOS are of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Approximation by truncated Chebyshev series.

Accuracy:

RFRSIN and RFRCOS (except on CDC and Cray computers) have full single-precision accuracy. For most values of the argument X, DFRSIN and DFRCOS (and RFRSIN and RFRCOS on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

References:

1. Y.L. Luke, The special functions and their approximations, v. II, (Academic Press New York, 1969) 328–329.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.05.1987
Language : Fortran **Revised:** 01.12.1994

Fermi-Dirac Function

Function subprograms RFERDR and DFERDR calculate the Fermi-Dirac function

$$F_k(x) = \int_0^\infty \frac{t^{k/2}}{1+e^{t-x}}\,dt$$

for real argument x, and $k = -1, 1, 3$.
On CDC and Cray computers, the double-precision version DFERDR is not available.

Structure:

FUNCTION subprograms
User Entry Names: RFERDR, DFERDR
Obsolete User Entry Names: FERDR $\equiv$ RFERDR
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

RFERDR(X,K) or DFERDR(X,K) has the value $F_{\mathrm{K}}(\mathrm{X})$,

where RFERDR is of type REAL, DFERDR is of type DOUBLE PRECISION, and X has the same type as the function name. K (INTEGER) = -1, or 1 or 3.

Method:

Rational approximation.

Accuracy:

RFERDR (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, DFERDR (and RFERDR on CDC and Cray computers) has, for X $>$ 0, an accuracy of 7-10 digits and for X $<$ 0, an accuracy of 10 to 14 digits.

Error handling:

Error C323.1: K $\neq -1, 1, 3$. The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. W.J. Cody and H.C. Thacher,Jr., Rational approximations for Fermi-Dirac integrals of order $-1/2$, $1/2$ and $3/2$, Math. Comp. **21** (1967) 30–40.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.05.1987
Language : Fortran **Revised:** 01.12.1994

Arctangent integral

Function subprograms `RATANI` and `DATANI` calculate the arctangent integral

$$\mathrm{Ti}_2(x) = \int_0^x \frac{\arctan t}{t}\, dt$$

for real argument x.
On CDC and Cray computers, the double-precision version `DATANI` is not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RATANI`, `DATANI`
Obsolete User Entry Names: `ATANI` $\equiv$ `RATANI`

Usage:

In any arithmetic expression,

`RATANI(X)` or `DATANI(X)` has the value $\mathrm{Ti}_2(\mathtt{X})$,

where `RATANI` is of type `REAL`, `DATANI` is of type `DOUBLE PRECISION`, and `X` has the same type as the function name.

Method:

Approximation by truncated Chebyshev series and functional relations.

Accuracy:

`RATANI` (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument `X`, `DATANI` (and `RATANI` on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press New York, 1975) 67.

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 01.12.1994
Language : Fortran | **Revised:**

Clausen Function

Function subprograms RCLAUS and DCLAUS calculate the Clausen function

$$\mathrm{Cl}_2(x) \;=\; -\int_0^x \ln\left|2\sin\frac{t}{2}\right|\,dt \;=\; \sum_{k=1}^{\infty}\frac{\sin kx}{k^2}$$

for real arguments x.
On CDC and Cray computers, the double-precision version DCLAUS is not available.

Structure:

FUNCTION subprograms
User Entry Names: RCLAUS, DCLAUS

Usage:

In any arithmetic expression,

RCLAUS(X) or DCLAUS(X) has the value $\mathrm{Cl}_2(\mathtt{X})$,

where RCLAUS is of type REAL, DCLAUS is of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

For $0 \le x \le \pi$, the function is approximated by truncated Chebyshev series. For x outside this range, the relations $\mathrm{Cl}_2(\pi + x) = -\mathrm{Cl}_2(\pi - x)$ and $\mathrm{Cl}_2(2n\pi \pm x) = \pm\mathrm{Cl}_2(x)$ are used.

Accuracy:

RCLAUS (except on CDC and Cray computers) has full single-precision accuracy in the interval $0 \le x \le 2\pi$. For most values of the argument $\mathtt{X} \in [0, 2\pi]$, DCLAUS (and RCLAUS on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision. Accuracy is lost near the zero of $\mathrm{Cl}_2(x)$ at $x = \pi$ and for large values of $|x|$.

References:

1. K.S. Kölbig, Chebyshev coefficients for the Clausen function $\mathrm{Cl}_2(x)$, J. Comput. Appl. Math. **64** (1995) 295–297.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.05.1987
Language : Fortran **Revised:** 15.03.1993

Modified Bessel Functions I and K of Order 1/4, 1/2 and 3/4

Function subprograms BSIR4, BSKR4 and DBSIR4, DBSKR4 calculate the modified Bessel functions

$$I_{\nu/4}(x) \quad \text{and} \quad K_{\nu/4}(x)$$

for real arguments $x > 0$ and $\nu = -3, -2, -1, 1, 2, 3$. The value $x = 0$ is permitted for the functions I if $\nu > 0$. Note that the functions K are even with respect to ν.
On CDC and Cray computers, the double-precision versions DBSIR4 etc. are not available.

Structure:

FUNCTION subprograms
User Entry Names: BSIR4, BSKR4,EBSIR4, EBSKR4, DBSIR4, DBSKR4, DEBIR4, DEBKR4
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

BSIR4(X,NU)	or	DBSIR4(X,NU)	has the value	$I_{\mathtt{NU}/4}(\mathtt{X})$,
BSKR4(X,NU)	or	DBSKR4(X,NU)	has the value	$K_{\mathtt{NU}/4}(\mathtt{X})$,
EBSIR4(X,NU)	or	DEBIR4(X,NU)	has the value	$\exp(-\mathtt{X}) * I_{\mathtt{NU}/4}(\mathtt{X})$,
EBSKR4(X,NU)	or	DEBKR4(X,NU)	has the value	$\exp(\mathtt{X}) * K_{\mathtt{NU}/4}(\mathtt{X})$,

where BSIR4 etc. are of the type REAL, DBSIR4 etc. are of the type DOUBLE PRECISION, and X has the same type as the function name. NU is of type INTEGER and must have one of the values -3,-2,-1,1,2,3.

Method:

Approximation by rational functions (I for $|x| < 8$, K for $1 \le x \le 5$), by an algorithm based on power series (K for $0 < x < 1$), or else by truncated Chebyshev series. The cases $|\nu| = 2$ are elementary.

Accuracy:

BSIR4 etc. (except on CDC and Cray computers) have full single-precision accuracy. For most values of the argument X, DBSIR4 etc. (and BSIR4 etc. on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error C327.1: $\mathtt{X} \le 0$, or $\mathtt{X} < 0$, respectively, or $\mathtt{NU} \neq$ -3,-2,-1,1,2,3. The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, Mathematical functions and their approximations (Academic Press, New York 1975) 350, 357, 363, 366.

2. N.M. Temme, On the numerical evaluation of the modified Bessel function of the third kind, J. Comp. Phys. **19** (1975) 324–337.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran
Library: MATHLIB
Submitted: 15.01.1988
Revised: 15.03.1993

Whittaker Function M of Complex Argument and Complex Indices

Function subprograms `CWHITM` and `WWHITM` compute the Whittaker function

$$M_{\kappa,\mu}(z) \;=\; e^{-\frac{1}{2}z} z^{\frac{1}{2}+\mu} M(\tfrac{1}{2}+\mu-\kappa, 1+2\mu, z)$$

for complex arguments z and complex indices κ, μ, where $M(a,b,z)$ is Kummer's function (See Ref. 1). The z-plane is cut along the negative real axis.

The double-precision version `WWHITM` is available only on computers which support a `COMPLEX*16` Fortran data type.

Structure:

`FUNCTION` subprograms
User Entry Names: `CWHITM`, `WWHITM`
Files Referenced: `Unit 6`
External References: `CLGAMA` (C306), `WLGAMA` (C306), `CCLBES` (C309), `WCLBES` (C309), `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`CWHITM(Z,KA,MU)` or `WWHITM(Z,KA,MU)` has the value $M_{\mathtt{KA},\mathtt{MU}}(\mathtt{Z})$,

where `KA` $=\kappa$ and `MU` $=\mu$. `CWHITM` is of type `COMPLEX`, `WWHITM` is of type `COMPLEX*16`, and `Z`, `KA` and `MU` have the same type as the function name.

Method:

For $\mu-\kappa+\frac{1}{2}$ or $\mu+\kappa+\frac{1}{2}$ equal to a negative integer, $M_{\kappa,\mu}(z)$ reduces to a polynomial in z. For other values, a regular Coulomb wave function $F_0(\nu,\rho)$ is computed by using subprogram `CCLBES` (C309) in conjunction with functional relations.

Restrictions:

$\mu \neq -\frac{1}{2}, -\frac{3}{2}, \ldots$; Re $z \geq 0$ if Im $z = 0$.

Accuracy:

`CWHITM` (except on CDC and Cray computers) has full single-precision accuracy. For most values of the arguments, `WWHITM` (and `CWHITM` on CDC and Cray computers) has an accuracy of approximately two to three decimal digits less than the machine precision.

Error handling:

Error `C328.1`: `Z` $=$ `X` $+$ `iY` with `X` $<$ `0` and `Y` $=$ `0`.
Error `C328.2`: $2 *$ `MU` $= -\,n,\ (n = 1, 2, \ldots)$.
In both cases, the function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called. An error message is also written on `Unit 6` if the internal call to `CCLBES` or `WCLBES` returns `JFAIL` $\neq 0$ (see Short write-up for `CCLBES` (C309)).

References:

1. M. Abramowitz and I.A. Stegun (Eds.), Handbook of Mathematical Functions, Chapter 13, Confluent Hypergeometric Functions, 9th printing with corrections, (Dover, New York 1972).

2. L.J. Slater, Confluent hypergeometric functions, (University Press, Cambridge 1960)

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 15.05.1987
Language : Fortran | **Revised:** 01.12.1994

Legendre and Associated Legendre Functions

Subroutine subprograms RASLGF and DASLGF calculate, for a given real argument x, $(-1 \le x \le 1)$, and a given integer value of the order m, a sequence of either unnormalized or normalized Legendre $(m = 0)$ or Associated Legendre $(m \ne 0)$ functions of degree $n = 0, 1, 2, \ldots, N$, defined by

$$P_n^m(x) = (1-x^2)^{m/2} \frac{d^m}{dx^m} P_n(x) \qquad (m \ge 0) \qquad (1a)$$

$$P_n^m(x) = \frac{(n+m)!}{(n-m)!} P_n^{-m}(x) \qquad (m < 0) \qquad (1b)$$

$$\overline{P_n^m}(x) = \sqrt{\frac{2n+1}{2} \frac{(n-m)!}{(n+m)!}}\, P_n^m(x), \qquad (2)$$

respectively, where

$$P_n(x) \equiv P_n^0(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2-1)^n$$

is the Legendre polynominal of degree n. Note that some authors use an additional factor $(-1)^m$ in the definition (1).

On CDC and Cray computers, the double-precision version DASLGF is not available.

Structure:

SUBROUTINE subprograms
User Entry Names: RASLGF, DASLGF
Obsolete User Entry Names: ASLGF $\equiv$ RASLGF
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tASLGF(MODE,X,M,NL,P)
```

MODE (INTEGER) = 1 : Unnormalized functions (1),
= 2 : Normalized functions (2).

X (type according to t) The argument x.

M (INTEGER) The order m (upper index) of all functions in the computed sequence. It is permissible for M to be negative.

NL (INTEGER) Specifies the degree N of the last function in the computed sequences.

P (type according to t) One-dimensional array of dimension (0:d) where d $\ge$ NL.
On exit, P(n), (n = 0, 1, ..., NL), contains P_n^m(X) or $\overline{P_n^m}$(X) as specified by MODE. (See **Notes**).

Method:

The functions $P_n^m(x)$ are for $m > 0$ calculated by means of the standard recurrence relation.

Restrictions:

1. $-1 \leq X \leq 1$.
2. $MODE = 1$ or 2.
3. If $M = 0 : 0 \leq NL \leq 100$:
   if $M \neq 0 : |M| \leq 27$ and $0 \leq NL \leq 55 - |M|$; ($0 \leq NL \leq 33 - |M|$ on VAX/VMS).

Accuracy:

RASLGF (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, DASLGF (and RASLGF on CDC and Cray computers) has an accuracy of approximately two significant digits less than the machine precision.

Notes:

In accordance with the definitions, $P(n) = 0$ for $n = 0, 1, \cdots, |M| - 1$.

Error handling:

Error C330.1: $|X| > 1$.
Error C330.2: $MODE \neq 1$ and $MODE \neq 2$.
Error C330.3: M and NL incompatible.
In all cases, a message is written on Unit 6, unless subroutine MTLSET (N002) has been called. The initial contents of array P(n) is left unchanged.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.02.1989
Language : Fortran **Revised:** 01.12.1994

Conical Functions of the First Kind

Function subprograms RFCONC and DFCONC calculate the (real valued) conical function of the first kind

$$P^{m}_{-\frac{1}{2}+i\tau}(x)$$

for real $x > -1$, $\tau \geq 0$, and $m = 0, 1$, where $P^m_\nu(x)$ is the Legendre (or spherical) function of the first kind and $i = \sqrt{-1}$.
On CDC and Cray computers, the double-precision version DFCONC is not available.

Structure:

FUNCTION subprograms
User Entry Names: RFCONC, DFCONC
Obsolete User Entry Names: FCONC $\equiv$ RFCONC
Files Referenced: Unit 6
External References: CGAMMA (C305), WGAMMA (C305), CLGAMA (C306), WLGAMA (C306), BESJ0 (C312), DBESJ0 (C312), BESJ1 (C312), DBESJ1 (C312), BESI0 (C313), DBESI0 (C313), BESI1 (C313), DBESI1 (C313), RELIKC (C347), DELIKC (C347), RELIEC (C347), DELIEC (C347), MTLMTR (N002), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
tFCONC(X,TAU,M)}
```

has, in any arithmetic expression, the value $P^{\mathtt{M}}_{\frac{1}{2}+i*\mathtt{TAU}}(\mathtt{X})$.

X (type according to t) Variable x.

TAU (type according to t) The imaginary part of the index, τ.

M (INTEGER) Order m. (M = 0 or M = 1).

Method:

Either (i) series expansions based on the Gaussian hypergeometric function and evaluated by direct summation or from rational approximations, or (ii) asymptotic expressions in terms of Bessel functions. For $\tau = 0$ the conical functions (for $m = 0, 1$) can be expressed in terms of complete elliptic integrals. For details see Ref. 1.

Restrictions:

X ≥ -1, TAU ≥ 0, M = 0 or 1.

Accuracy:

RFCONC (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, DFCONC (and RFCONC on CDC and Cray computers), an accuracy of not less than 10 significant digits is usually obtained. If x and τ are not too large the accuracy increases to about 12-13 significant digits.

Error handling:

Error C331.1: X < -1 or TAU < 0 or M $\neq 0$ and M $\neq 1$.
Error C331.2: Problems of convergence for a hypergeometric function.
In both cases, the function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

Notes:

This program is an (only formally) modified version of the CPC Program Library Package FCONIC (see Ref. 1).

References:

1. K.S. Kölbig, A program for computing the conical functions of the first kind $P^m_{1/2+i\tau}(x)$ for $m = 0$ and $m = 1$, Computer Phys. Comm. **23** (1981) 51–61.
2. M.I. Zhurina and L.N. Karmazina, Tables and formulae for the spherical functions $P^m_{1/2+i\tau}(z)$, (Pergamon Press, Oxford 1966).

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 19.10.1966
Revised: 01.12.1994

Dilogarithm Function

Function subprograms `RDILOG` and `DDILOG` calculate the dilogarithm function

$$\mathrm{Li}_2(x) \;=\; -\int_0^x \frac{\ln|1-t|}{t} dt$$

for real arguments x.

On CDC and Cray computers, the double-precision version `DDILOG` is not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RDILOG`, `DDILOG`
Obsolete User Entry Names: `DILOG` $\equiv$ `RDILOG`

Usage:

In any arithmetic expression,

`RDILOG(X)` or `DDILOG(X)` has the value $\mathrm{Li}_2(\mathtt{X})$,

where `RDILOG` is of type `REAL`, `DDILOG` is of type `DOUBLE PRECISION`, and `X` has the same type as the function name.

Method:

Approximation by truncated Chebyshev series and functional relations.

Accuracy:

`RDILOG` (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument `X`, `DDILOG` (and `RDILOG` on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press, New York 1975) 67.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.05.1990
Revised: 01.12.1994

Incomplete Gamma Functions

Function subprograms RGAPNC, DGAPNC and RGAGNC, DGAGNC calculate the incomplete gamma function

$$P(a,x) = \begin{cases} \dfrac{1}{\Gamma(a)} \displaystyle\int_0^x e^{-t}\, t^{a-1}\, dt & (a > 0) \\ e^{-x}\, x^a \, \dfrac{M(1, a+1, x)}{\Gamma(a+1)} & (a \le 0), \end{cases}$$

and the complementary incomplete gamma function

$$G(a,x) = \begin{cases} \dfrac{1}{\Gamma(a)} \displaystyle\int_x^\infty e^{-t}\, t^{a-1}\, dt & (a > 0) \\ e^{x}\, x^{-a} \displaystyle\int_x^\infty e^{-t}\, t^{a-1}\, dt & (a \le 0), \end{cases}$$

respectively, for real arguments $x \ge 0$ and a. $M(a,b,x)$ is Kummer's function (see Ref. 3).
On CDC and Cray computers, the double-precision versions DGAPNC and DGAGNC are not available.

Structure:

FUNCTION subprograms
Uses Entry Names: RGAPNC, RGAGNC, DGAPNC, DGAGNC
Obsolete User Entry Names: GAPNC $\equiv$ RGAPNC, GAGNC $\equiv$ RGAGNC
Files Referenced: Unit 6
External References: ALGAMA (C304), DLGAMA (C304), MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

RGAPNC(A,X)	or	DGAPNC(A,X)	has the value	P(A,X),
RGAGNC(A,X)	or	DGAGNC(A,X)	has the value	G(A,X),

where RGAPNC and RGAGNC are of type REAL, DGAPNC and DGAGNC are of type DOUBLE PRECISION, A and X have the same type as the function name.

Method:

The method is described in Ref. 1.

Accuracy:

RGAPNC and RGAGNC (except on CDC and Cray computers) have full single-precision accuracy. For most values of the arguments, DGAPNC, DGAGNC (and RGAPNC, RGAGNC on CDC and Cray computers) have an accuracy of approximately two significant digits less than the machine precision.

Restrictions:

For $P(a,x)$: Either (i) X $>$ 0, or (ii) X $=$ 0 and A $\ge$ 0.
For $G(a,x)$: Either (i) X $>$ 0, or (ii) X $=$ 0 and A $\ne$ 0.

Error handling:

Error C334.1: $X < 0$.
Error C334.2: For RGAPNC and DGAPNC: $A < 0$ and $X = 0$; for RGAGNC and DGAGNC: $A = X = 0$.
Error C334.3: Problems with convergence (unlikely).
In all cases, the function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

Notes:

When speed is more important than accuracy, e.g. for applications in statistics, use GAMDIS (G106) for computing $P(a, x)$. Note, however, that in this case the arguments A and X must be interchanged.

Source:

The subprograms are based on a Fortran program for the incomplete gamma functions published in Ref. 2.

References:

1. W. Gautschi, A computational procedure for incomplete gamma functions, ACM Trans. Math. Software **5** (1979) 466–481.
2. W. Gautschi, Algorithm 542, Incomplete gamma functions, Collected Algorithms from CACM (1979).
3. M. Abramowitz and I.A. Stegun (Eds.), Handbook of Mathematical Functions, Chapter 13, Confluent Hypergeometric Functions, 9th printing with corrections, (Dover, New York 1972).

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.12.1970
Language : Fortran **Revised:** 15.03.1993

Complex Error Function

Function subprograms `CWERF` and `WWERF` calculate the complex error function

$$w(z) \;=\; e^{-z^2}\left[1+\frac{2i}{\sqrt{\pi}}\int_0^z e^{t^2}\,dt\right] \;=\; e^{-z^2}\left[1-\mathrm{erf}\,(-iz)\right] \;=\; e^{-z^2}\,\mathrm{erfc}\,(-iz)$$

for complex arguments z, where $i=\sqrt{-1}$.
The double-precision version `WWERF` is available only on computers which support a `COMPLEX*16` Fortran data type.

Structure:

`FUNCTION` subprograms
User Entry Names: `CWERF`, `WWERF`

Usage:

In any arithmetic expression,

`CWERF(Z)` or `WWERF(Z)` has the value $w(\mathtt{Z})$,

where `CWERF` is of type `COMPLEX`, `WWERF` is of type `COMPLEX*16`, and `Z` has the same type as the function name.

Method:

The method is described in Ref. 2.

Accuracy:

`CWERF` (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument `Z`, `WWERF` (and `CWERF` on CDC and Cray computers) has an accuracy of approximately two significant digits less than the machine precision.

Notes:

This subprogram is a modified version of the algorithm presented in Ref. 1.

References:

1. W. Gautschi, Algorithm 363, Complex Error Function, Collected Algorithms from CACM (1969).
2. W. Gautschi, Efficient Computation of the Complex Error Function, SIAM J. Numer. Anal. **7** (1970) 187–198.
3. K.S. Kölbig, Certification of Algorithm 363 Complex Error Function, Comm. ACM **15** (1972) 465–466.

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 07.12.1970
Language : Fortran | **Revised:** 01.12.1994

Sine and Cosine Integrals

Function subprograms `RSININ`, `RCOSIN` and `DSININ`, `DCOSIN` calculate the sine and cosine integrals

$$\begin{aligned}\mathrm{Si}(x) &= \int_0^x \frac{\sin t}{t}dt \\ \mathrm{Ci}(x) &= \gamma + \ln|x| + \int_0^x \frac{\cos t - 1}{t}dt \qquad (x \neq 0)\end{aligned}$$

for real arguments x, where $\gamma = 0.57721\ldots$ is Euler's constant.
On CDC and Cray computers, the double-precision versions `DSININ` and `DCOSIN` are not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RSININ`, `RCOSIN`, `DSININ`, `DCOSIN`
Obsolete User Entry Names: `SININT` ≡ `RSININ`, `COSINT` ≡ `RCOSIN`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`RSININ(X)`	or	`DSININ(X)`	has the value	Si(X),
`RCOSIN(X)`	or	`DCOSIN(X)`	has the value	Ci(X),

where `RSININ` and `RCOSIN` are of type `REAL`, `DSININ` and `DCOSIN` are of type `DOUBLE PRECISION`, and `X` has the same type as the function name.

Method:

Approximation by truncated Chebyshev series.

Accuracy:

`RSININ` and `RCOSIN` (except on CDC and Cray computers) have full single-precision accuracy. For most values of the argument `X`, `DSININ`, `DCOSIN` (and `RSININ`, `RCOSIN` on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error `C336.1: X = 0` for `RCOSIN` or `DCOSIN`. The function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

References:

1. Y.L. Luke, The special functions and their approximations, v.II, (Academic Press, New York l969) 325–326

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.12.1970
Language : Fortran **Revised:** 15.03.1993

Exponential Integral

Function subprograms `REXPIN` and `DEXPIN` calculate the exponential integral

$$E_1(x) \;=\; -\mathrm{Ei}(-x) \;=\; \int_x^\infty \frac{e^{-t}}{t} dt$$

for real arguments x. For $x < 0$, the real part of the principal value of the integral is taken.
On CDC and Cray computers, the double-precision versions `DEXPIN` and `DEXPIE` are not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `REXPIN`, `REXPIE`, `DEXPIN`, `DEXPIE`
Obsolete User Entry Names: `EXPINT` $\equiv$ `REXPIN`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`REXPIN(X)`	or	`DEXPIN(X)`	has the value	$E_1(\mathtt{X})$,
`REXPIE(X)`	or	`DEXPIE(X)`	has the value	$e^{\mathtt{X}} E_1(\mathtt{X})$,

where `REXPIN` and `REXPIE` are of type `REAL`, `DEXPIN` and `DEXPIE` are of type `DOUBLE PRECISION`, and `X` has the same type as the function name.

Method:

Polynomial and rational approximations.

Accuracy:

`REXPIN` and `REXPIE` (except on CDC and Cray computers) have full single-precision accuracy. For most values of the argument `X`, `DEXPIN`, `DEXPIE` (and `REXPIN`, `REXPIE` on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error `C337.1`: `X` $=$ `0`. The function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

References:

1. W.J. Cody and H.C. Thatcher,Jr., Rational Chebyshev approximations for the exponential integral $E_1(x)$, Math. Comp. **22** (1968) 641–649.
2. W.J. Cody and H.C. Thatcher,Jr., Chebyshev approximations for the exponential integral Ei(x), Math. Comp. **23** (1969) 289–303.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.05.1990
Revised: 15.03.1993

Exponential Integral for Complex Argument

Function subprograms CEXPIN and WEXPIN calculate the the exponential integral

$$E(z) \;=\; \int_0^z t^{-1}\,(1-e^{-t})\,dt$$

for complex arguments z.
The double-precision version WEXPIN is available only on computers which support a COMPLEX*16 Fortran data type.

Structure:

FUNCTION subprograms
Use Entry Names : CEXPIN, WEXPIN
Files referenced : Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

CEXPIN(Z) or WEXPIN(Z) has the value $E(\mathtt{Z})$,

where CEXPIN is of type COMPLEX, WEXPIN is of type COMPLEX*16, and Z has the same type as the function name.

Method:

Padé approximants are used to compute $E(z)=E(x+iy)$ in the following (partly overlapping) regions of the z-plane:

(i) $(\frac{1}{7}(x-1))^2+(\frac{1}{5}y)^2 \le 1 \quad (x\ge -5),$

(ii) $(\frac{1}{15}(x+12))^2+(\frac{1}{12}y)^2 \ge 1 \quad (x\ge -12),$

(iii) $(\frac{1}{12}y)^2 \ge 1 \quad (x< -12).$

In the remaining region, consisting mainly of a strip along the negative real axis, $E(z)$ is computed by numerical integration (which is very much slower than the evaluation of the Padé approximations).

Accuracy:

CEXPIN (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument Z, WEXPIN (and CEXPIN on CDC and Cray computers) has an accuracy of approximately two significant digits less than the machine precision.

Error handling:

Error C338.1: Numerical integration not successful (unlikely). The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, the special functions and their approximations, v. II, (Academic Press, New York 1969) 198–199, 402–416.

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 07.12.1970
Language : Fortran | **Revised:** 01.12.1994

Dawson's Integral

Function subprograms RDAWSN and DDAWSN calculate the Dawson integral

$$F(x) \ = \ e^{-x^2} \int_0^x e^{t^2} \, dt$$

for real arguments x.
On CDC and Cray computers, the double-precision version DDAWSN is not available.

Structure:

FUNCTION subprograms
User Entry Names: RDAWSN, DDAWSN
Obsolete User Entry Names: DAWSON $\equiv$ RDAWSN

Usage:

In any arithmetic expression,

RDAWSN(X) or DDAWSN(X) has the value $F(\mathrm{X})$,

where RDAWSN is of type REAL, DDAWSN is of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Rational approximation.

Accuracy:

RDAWSN (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, DDAWSN (and RDAWSN on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

References:

1. W.J. Cody, K.A. Paciorek and H.C. Thacher,Jr., Chebyshev approximations for Dawson's integral, Math. Comp. **24** (1970) 171–178.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.12.1970
Language : Fortran **Revised:** 15.03.1993

Modified Bessel Functions I and K of Order 1/3 and 2/3

Function subprograms BSIR3, BSKR3 and DBSIR3, DBSKR3 calculate the modified Bessel functions

$$I_{\nu/3}(x) \quad \text{and} \quad K_{\nu/3}(x)$$

for real arguments $x > 0$ and $\nu = -2, -1, 1, 2$. The value $x = 0$ is permitted for the functions I if $\nu > 0$. Note that the functions K are even with respect to ν.
On CDC and Cray computers, the double-precision versions DBSIR3 etc. are not available.

Structure:

FUNCTION subprograms
User Entry Names: BSIR3, BSKR3, EBSIR3, EBSKR3, DBSIR3, DBSKR3, DEBIR3, DEBKR3
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

BSIR3(X,NU)	or	DBSIR3(X,NU)	has the value	$I_{\mathtt{NU}/3}(\mathtt{X})$,
BSKR3(X,NU)	or	DBSKR3(X,NU)	has the value	$K_{\mathtt{NU}/3}(\mathtt{X})$,
EBSIR3(X,NU)	or	DEBIR3(X,NU)	has the value	$\exp(-\mathtt{X}) * I_{\mathtt{NU}/3}(\mathtt{X})$,
EBSKR3(X,NU)	or	DEBKR3(X,NU)	has the value	$\exp(\mathtt{X}) * K_{\mathtt{NU}/3}(\mathtt{X})$,

where BSIR3 etc. are of the type REAL, DBSIR3 etc. are of the type DOUBLE PRECISION, and X has the same type as the function name. NU (INTEGER) has one of the values -2,-1,1,2.

Method:

Approximation by rational functions (I for $|x| < 8$, K for $1 \le x \le 5$), by an algorithm based on power series (K for $0 < x < 1$), or else by truncated Chebyshev series.

Accuracy:

BSIR3 etc. (except on CDC and Cray computers) has full single-precision accuracy. For most values of the argument X, DBSIR3 etc. (and BSIR3 etc. on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error C340.1: $\mathtt{X} \le 0$ or $\mathtt{X} < 0$, repectively, or $\mathtt{NU} \ne -2, -1, 1, 2$.
The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, Mathematical functions and their approximations (Academic Press, New York 1975) 352, 355, 363, 366.
2. N.M. Temme, On the numerical evaluation of the modified Bessel function of the third kind, J. Comp. Phys. **19** (1975) 324–337.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.10.1994
Language : Fortran **Revised:**

Modified Bessel Functions K of Certain Order

Subroutine subprograms BSKA and DBSKA calculate the sequence of modified Bessel functions

$$K_{a+n}(x)$$

for real argument $x > 0$ and a chosen $a \in \left\{0,\ \frac{1}{2},\ \frac{1}{3},\ \frac{1}{4},\ \frac{2}{3},\ \frac{3}{4}\right\}$.
On CDC and Cray computers, the double-precision versions DBSKA and DEBKA are not available.

Structure:

SUBROUTINE subprograms
User Entry Names: BSKA, EBSKA, DBSKA,DEBKA
Files Referenced: Unit 6
External References: BESK0 (C313), BESK1 (C313), EBESK0 (C313), EBESK1 (C313), DBESK0 (C313), DBESK1 (C313), DEBSK0 (C313), DEBSK1 (C313), BSKR4 (C327), EBSKR4 (C327), DBSKR4 (C327), DEBKR4 (C327), BSKR3 (C340), EBSKR3 (C340), DBSKR3 (C340), DEBKR3 (C340), MTLMTR (N002), ABEND (Z035)

Usage:

Single-precision version:

```
CALL BSKA(X,IA,JA,NL,B)     or     CALL EBSKA(X,IA,JA,NL,B)
```

X (REAL) Argument x.

IA,JA (INTEGER) Numerator i and denominator j of $a = i/j$. Only the pairs

(IA,JA) = (0,1), (1,2), (1,3), (1,4), (2,3), (3,4)

are permitted. For example, IA $= 2$ and JA $= 3$ corresponds to $a = 2/3$.

NL (INTEGER) Specifies the order $a +$ NL of the last Bessel function in the computed sequence.

B (REAL) One-dimensional array with dimension (0:d) where d $\geq$ NL.
On exit, B(n), (n $= 0, 1, 2, \ldots,$ NL), contains $K_{a+\mathtt{n}}(\mathtt{X})$ for BSKA, $\exp(\mathtt{X}) * K_{a+\mathtt{n}}(\mathtt{X})$ for EBSKA, respectively.

Double-precision version:

```
CALL DBSKA(X,IA,JA,NL,B)     or     CALL DEBKA(X,IA,JA,NL,B)
```

where X and B are of type DOUBLE PRECISION.

Method:

The well-known recurrence relation for modified Bessel functions is used.

Restrictions:

X > 0, NL ≤ 100. Only the pairs (IA,JA) given above are permitted.

Error handling:

Error `C341.1`: `X` $\leq$ `0`.
Error `C341.2`: Pair (`IA,JA`) not permitted.
Error `C341.3`: `NL` $>$ `100`.
In all cases, a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called. The initial contents of array `B` is left unchanged.

•

RSTRH0	CERN Program Library	C342

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.11.1971
Language : Fortran **Revised:** 01.12.1994

Struve Functions of Orders Zero and One

Function subprograms RSTRH0, RSTRH1 and DSTRH0, DSTRH1 calculate the Struve functions

$$\mathbf{H}_n(x) \;=\; (\tfrac{1}{2}x)^{n+1}\sum_{k=0}^{\infty}\frac{(-1)^k(\frac{1}{2}x)^{2k}}{\Gamma(k+\frac{3}{2})\Gamma(k+n+\frac{3}{2})}$$

for real arguments x and $n = 0, 1$.
On CDC and Cray computers, the double-precision versions DSTRH0, DSTRH1 are not available.

Structure:

FUNCTION subprograms
User Entry Names: RSTRH0, RSTRH1, DSTRH0, DSTRH1
Obsolete User Entry Names: STRH0 $\equiv$ RSTRH0, STRH1 $\equiv$ RSTRH1
External References: BESJ0 (C312), DBESJ0 (C312), BESY0 (C312), DBESY0 (C312)

Usage:

In any arithmetic expression,

RSTRH0(X)	or	DSTRH0(X)	has the value	$\mathbf{H}_0$(X),
RSTRH1(X)	or	DSTRH1(X)	has the value	$\mathbf{H}_1$(X),

where RSTRH0, RSTRH1 are of type REAL, DSTRH0, DSTRH1 are of type DOUBLE PRECISION, and X has the same type as the function name.

Method:

Approximation by truncated Chebyshev series.

Accuracy:

RSTRH0 and RSTRH1 (except on CDC and Cray computers) have full single-precision accuracy. For most values of the argument X, DSTRH0, DSTRH1 (and RSTRH0, RSTRH1 on CDC and Cray computers) have an accuracy of approximately one significant digit less than the machine precision.

References:

1. Y.L. Luke, The special functions and their approximations, v.II (Academic Press, New York 1969) 370–371.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 24.01.1986
Language : Fortran **Revised:** 15.03.1993

Bessel Functions J and I with Positive Argument and Non-Integer Order

Subroutine subprograms BSJA, BSIA, DBSJA, DBSIA and QBSJA, QBSIA calculate the sequences of Bessel functions

$$J_{a+n}(x),\ J_{a-n}(x),\ I_{a+n}(x) \text{ or } I_{a-n}(x),$$

for real argument $x > 0$, $0 \leq a < 1$, and $n = 0, 1, 2, \ldots, N$.

The quadruple-precision versions QBSJA and QBSIA are available only on computers which support a REAL*16 Fortran data type.

Structure:

SUBROUTINE subprograms
User Entry Names: BSJA, BSIA, DBSJA, DBSIA, QBSJA, QBSIA
Files Referenced: Unit 6
External References: GAMMA (C302), DGAMMA (C302), QGAMMA (C302), MTLMTR (N002), ABEND (Z035)

Usage:

Single-precision version:

```
CALL BSJA(X,A,NL,ND,B)     or     CALL BSIA(X,A,NL,ND,B)
```

X (REAL) Argument x.

A (REAL) Order a of the first Bessel function in the computed sequence.

NL (INTEGER) Specifies the order $a+$ NL of the last Bessel function in the computed sequence. It is permissible for NL to be negative.

ND (INTEGER) Requested number of correct significant decimal digits.

B (REAL) One-dimensional array with dimension (0:d) where d $\geq$ |NL|.
On exit, B(n), (n $= 0, 1, 2, \ldots,$ |NL|), contains $J_{a+\mathrm{n}}(\mathrm{X})$, $J_{a-\mathrm{n}}(\mathrm{X})$, $I_{a+\mathrm{n}}(\mathrm{X})$ or $I_{a-\mathrm{n}}(\mathrm{X})$, the plus sign of the subscript being taken if NL ≥ 0, the minus sign if NL < 0.

Double-precision version:

```
CALL DBSJA(X,A,NL,ND,B)     or     CALL DBSIA(X,A,NL,ND,B)
```

where X, A and B are of type DOUBLE PRECISION.

Quadruple-precision version:

```
CALL QBSJA(X,A,NL,ND,B)     or     CALL QBSIA(X,A,NL,ND,B)
```

where X, A and B are of type REAL*16.

Method:

For NL ≥ 0, the method of computation is a variant of Miller's backwards recurrence algorithm (see Ref. 1). The requested accuracy is obtained, when possible, by a judicious choice of the initial value of the recursion index, together with at least one repetition of the recursion with this index increased by 5. For NL < 0, only the first two functions in the sequence are computed in this way. The remaining functions are computed by the standard Bessel function recurrence relation.

Restrictions:

X > 0, $0 \leq$ A < 1, $|$NL$| \leq 100$.

Accuracy:

If X is close to a zero of one of the functions $J_{a+n}(x)$, the accuracy of that particular function will be less than ND significant digits. There may also be a loss of accuracy in any of the computed functions if A is close to 0 or 1, and in other special situations.

Error handling:

Error C343.1: X ≤ 0.
Error C343.2: A < 0 or A ≥ 1.
Error C343.3: $|$NL$| > 100$.
Error C343.4: Difficulties of convergence. Try smaller $|$ND$|$.
In all cases, a message is written on Unit 6, unless subroutine MTLSET (N002) has been called. If Error 1 to 3 occurs, the initial contents of array B is left unchanged. If the requested accuracy cannot be obtained after the initial recursion index has been increased fifty times (Error 4), the contents of array B is undefined.

Source:

The subprogram is based on Algol procedures described in Ref. 1.

References:

1. W. Gautschi, Algorithm 236, Bessel functions of the first kind, Collected Algorithms from CACM (1972)

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 24.01.1986
Language : Fortran **Revised:** 15.03.1993

Bessel Functions J with Complex Argument and Non-Integer Order

Subroutine subprograms CBSJA, WBSJA and WQBSJA calculate a sequence of Bessel functions

$$J_{a+n}(z),$$

for complex arguments z, $0 \leq a < 1$, and $n = 0, 1, 2, \ldots, N$.
The quadruple-precision version WQBSJA is available only on computers which support a COMPLEX*32 Fortran data type.

Structure:

SUBROUTINE subprograms
User Entry Names: CBSJA, WBSJA, WQBSJA
Files Referenced: Unit 6
External References: GAMMA (C302), DGAMMA (C302), QGAMMA (C302), MTLMTR (N002), ABEND (Z035)

Usage:

Single-precision version:

```
CALL CBSJA(Z,A,NL,ND,CB)
```

Z (COMPLEX) Argument z.

A (REAL) Order a of the first Bessel function in the computed sequence.

NL (INTEGER) Specifies the order $a+$ NL of the last Bessel function in the computed sequence.

ND (INTEGER) Requested number of correct significant decimal digits.

CB (COMPLEX) One-dimensional array with dimension (0:d) where d $\geq$ NL.
On exit, CB(n), (n $= 0, 1, 2, \ldots,$ NL), contains $J_{a+\mathrm{n}}(\mathrm{Z})$.

Double-precision version:

```
CALL WBSJA(Z,A,NL,ND,CB)
```

where A is of type DOUBLE PRECISION, Z and CB are of type COMPLEX*16.
On computers whose Fortran compiler does not support COMPLEX*16 arithmetic (e.g. CDC and Cray) the storage conventions for Z and CB are as follows:

Z (DOUBLE PRECISION) Array of declared dimension (2) containing Re Z in Z(1) and Im Z in Z(2).

CB (DOUBLE PRECISION) Two-dimensional array with dimensions (2,0:d) where d $\geq$ NL. On exit, CB(1,n) contains Re $J_{a+\mathrm{n}}(\mathrm{Z})$ and CB(2,n) contains Im $J_{a+\mathrm{n}}(\mathrm{Z})$, (n $= 0, 1, 2, \ldots,$ NL).

Quadruple-precision version:

```
CALL WQBSJA(Z,A,NL,ND,CB)
```

where A is of type REAL*16, Z and CB are of type COMPLEX*32.

Method:

The method is an extension to complex arguments of a variant of Miller's backwards recurrence algorithm (see Ref. 1). The requested accuracy is obtained, when possible, by a judicious choice of the initial value of the recursion index, together with at least one repetition of the recursion with this index increased by 5.

Restrictions:

Im Z $\neq 0$ if Re Z < 0, $0 \leq$ A < 1, $0 \leq$ NL ≤ 100.

Accuracy:

If Z does not lie on the real axis, the requested accuracy is usually obtained. There may be a loss of accuracy if A is close to 0 or 1, and in other special situations.

Error handling:

Error C344.1: $Z = X + iY$ with $X \leq 0$ and $Y = 0$.
Error C344.2: $A < 0$ or $A \geq 1$.
Error C344.3: $NL < 0$ or $NL > 100$.
Error C344.4: Difficulties of convergence. Try smaller $|ND|$.
In all cases, a message is written on Unit 6, unless subroutine MTLSET (N002) has been called. If Error 1 to 3 occurs, the initial contents of array CB is left unchanged. If the requested accuracy cannot be obtained after the initial recursion index has been increased fifty times (Error 4), the contents of array CB is undefined.

Source:

The subprogram is based on Algol procedures described in Ref. 1.

References:

1. W. Gautschi, Algorithm 236: Bessel functions of the first kind, Collected Algorithms from CACM (1965)

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.08.1989
Revised: 01.12.1994

Zeros of Bessel Functions J and Y

Subroutine subprograms RBZEJY and DBZEJY calculate, for real order $a \geq 0$, the first $N > 0$ zeros

$$j_{a,n},\ y_{a,n},\ j'_{a,n},\ y'_{a,n} \qquad (n = 1, 2, \ldots, N)$$

of the Bessel functions $J_a(x)$, $Y_a(x)$, $J'_a(x)$, $Y'_a(x)$, respectively. The prime denotes the derivative of the function with respect to x.

On CDC and Cray computers, the double-precision version DBZEJY is not available.

Structure:

SUBROUTINE subprograms
User Entry Names: RBZEJY, DBZEJY
Obsolete User Entry Names: BZEJY $\equiv$ RBZEJY
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tBZEJY(A,N,MODE,REL,X)
```

A — (type according to t) Order a.

N — (INTEGER) Number N of zeros wanted.

MODE — (INTEGER) defines the function for which the zeros are to be calculated:
- 1 — zeros of $J_a(x)$,
- 2 — zeros of $Y_a(x)$,
- 3 — zeros of $J'_a(x)$,
- 4 — zeros of $Y'_a(x)$.

REL — (type according to t) The requested relative accuracy.

X — (type according to t) One-dimensional array of length N at least. On exit, X(n), (n = 1, 2, ..., N) contains the first N positive (in the case A = 0 and MODE = 3, non-negative) zeros of the function defined by MODE.

Method:

Initial approximations to the zeros are computed from asymptotic expansions. These values are improved by higher-order Newton iteration making use of the differential equation for the Bessel functions. (For details see Ref. 1).

Error handling:

Error C345.1: A < 0. A message is written on Unit 6, unless subroutine MTLSET (N002) has been called. The contents of X is left unchanged. N $\leq$ 0 acts as do nothing.

Source:

The subroutine is based on Algol procedures published in the References.

References:

1. N.M. Temme, An algorithm with Algol60 program for the computation of the zeros of ordinary Bessel functions and those of their derivatives, J. Comput. Phys. **32** (1979) 270–279.
2. N.M. Temme, On the numerical evaluation of the ordinary Bessel function of the second kind, J. Comput. Phys. **21** (1976) 343–350.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Elliptic Integrals of First, Second, and Third Kind

Function subprograms RELI1, RELI2, RELI3 and DELI1, DELI2, DELI3 calculate, for real argument x, the elliptic integrals of the first, second and third kind, respectively.

On CDC and Cray computers, the double-precision versions DELI1, DELI2 and DELI3 are not available.

Mainly for reasons of numerical stability, the algorithms are based on the following definitions:

First kind:

$$\mathbf{F}_1(x,k') = \int_0^x \frac{d\xi}{\sqrt{(1+\xi^2)(1+k'^2\xi^2)}} \qquad (k'^2 \geq 0).$$

Second kind:

$$\mathbf{F}_2(x,k',a,b) = \int_0^x \frac{a+b\xi^2}{(1+\xi^2)\sqrt{(1+\xi^2)(1+k'^2\xi^2)}}\,d\xi \qquad (k'^2 \geq 0).$$

Third kind:

$$\mathbf{F}_3(x,k',p) = \int_0^x \frac{1+\xi^2}{(1+p\xi^2)\sqrt{(1+\xi^2)(1+k'^2\xi^2)}}\,d\xi \qquad (k'^2 \geq 0,\ px^2 \neq -1).$$

Note that $\mathbf{F}_1(x,k') = \mathbf{F}_2(x,k',1,1) = \mathbf{F}_3(x,k',1)$. For $p < 0$, the integral $\mathbf{F}_3$ is defined by its principal value.

For the integral of the second kind, a special entry-mode argument is provided which allows $\mathbf{F}_2(x,k',a,b)$ to be calculated when $k'^2 < 0$, i.e. when k' is imaginary.

Other common definitions of the elliptic integrals and their relations to $\mathbf{F}_1,\ \mathbf{F}_2,\ \mathbf{F}_3$ are listed here for convenience ($k^2 + k'^2 = 1$):

First kind:

$$F(k,\phi) = \int_0^\phi \frac{d\psi}{\sqrt{1-k^2\sin^2\psi}} = \mathbf{F}_1(\tan\phi,k') \qquad (|\phi| \leq \pi/2,\ |k| < 1),$$

$$\widehat{F}(y,k) = \int_0^y \frac{d\eta}{\sqrt{(1-\eta^2)(1-k^2\eta^2)}} = \mathbf{F}_1\left(y/\sqrt{1-y^2},k'\right) \qquad (|y| < 1,\ |k| < 1).$$

Second kind:

$$E(k,\phi) = \int_0^\phi \sqrt{1-k^2\sin^2\psi}\,d\psi = \mathbf{F}_2(\tan\phi,k',1,k'^2) \qquad (|\phi| \leq \pi/2,\ |k| \leq 1),$$

$$\widehat{E}(y,k) = \int_0^y \sqrt{\frac{1-k^2\eta^2}{1-\eta^2}}\,d\eta = \mathbf{F}_2\left(y/\sqrt{1-y^2},k',1,k'^2\right) \qquad (|y| < 1,\ |k| \leq 1).$$

Third kind:

$$\Pi(\phi,h,k) = \int_0^\phi \frac{d\psi}{(1+h\sin^2\psi)\sqrt{1-k^2\sin^2\psi}} = \mathbf{F}_3(\tan\phi,k',h+1)$$
$$(|\phi|\le\pi/2,\ |k|<1),$$

$$\widehat{\Pi}(y,h,k) = \int_0^y \frac{d\eta}{(1+h\eta^2)\sqrt{(1-\eta^2)(1-k^2\eta^2)}} = \mathbf{F}_3\left(y/\sqrt{1-y^2},k',h+1\right)$$
$$(|y|<1,\ |k|<1).$$

Structure:

`FUNCTION` subprograms
User Entry Names: `RELI1`, `RELI2`, `RELI3`, `DELI1`, `DELI2`, `DELI3`
Files Referenced: `Unit 6`
External References: `ASINH` (B102), `DASINH` (B102), `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression, with `AKP` $= k'$,

`RELI1(X,AKP)`	or	`DELI1(X,AKP)`	has the value	$\mathbf{F}_1(\mathtt{X},k')$,
`RELI2(X,AKP,A,B,MODE)`	or	`DELI2(X,AKP,A,B,MODE)`	has the value	$\mathbf{F}_2(\mathtt{X},k'',\mathtt{A},\mathtt{B})$,
`RELI3(X,AKP,P)`	or	`DELI3(X,AKP,P)`	has the value	$\mathbf{F}_3(\mathtt{X},k',\mathtt{P})$,

where `RELI1`, `RELI2`, `RELI3` are of type `REAL`, where `DELI1`, `DELI2`, `DELI3` are of type `DOUBLE PRECISION`, and `X`, `AKP`, `A`, `B` and `P` have the same type as the function name. `MODE` is of type `INTEGER`.

The notation k'' indicates that, when calling `RELI2` or `DELI2`, the parameters `AKP` and `MODE` must be set as follows:
If ${k'}^2 > 0$: `MODE` $= +1$ and `AKP` $= k'$,
if ${k'}^2 < 0$: `MODE` $= -1$ and `AKP` $=$ Im $k' = -ik'$ (real).

Method:

The evaluation of $\mathbf{F}_1$ and $\mathbf{F}_2$ is based on the Landen transformation, that of $\mathbf{F}_3$ on the Bartky transformation. $\mathbf{F}_2$ for ${k'}^2 < 0$ is calculated by using a transformation of the arguments. See Ref. 1 and 2 for details.

Accuracy:

The `REAL` functions (except on CDC and Cray computers) have full single-precision accuracy. The `REAL` functions on CDC and Cray, and the `DOUBLE PRECISION` functions on all computers have an accuracy approximately two significant digits less than the machine precision.

Restrictions:

1. `RELI2` and `DELI2`: `AKP*X**2` $<$ `1` if `MODE` $= -1$.
2. `RELI2` and `DELI2`: `MODE` $= +1$ or -1.
3. `RELI3` and `DELI3`: `P*X**2` $\neq -1$.

Error handling:

Error `C346.1`: Restriction 1 is not satisfied.
Error `C346.2`: Restriction 2 is not satisfied.
Error `C346.3`: Restriction 3 is not satisfied.
In all cases, the function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Source:

The subprograms are based on the Algol60 procedures *el1, el2* in Ref. 1 and *el3* in Ref. 2.

References:

1. R. Bulirsch, Numerical calculation of elliptic integrals and elliptic functions, Numer. Math. **7** (1965) 78–90.
2. R. Bulirsch, Numerical calculation of elliptic integrals and elliptic functions III, Numer. Math. **13** (1969) 305–315.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Complete Elliptic Integrals of First, Second, and Third Kind

Function subprograms `RELI1C`, `RELI2C`, `RELI3C` and `DELI1C`, `DELI2C`, `DELI3C` calculate the complete elliptic integrals of the first, second and third kind, respectively.

Function subprograms `RELIGC` and `DELIGC` calculate a general complete elliptic integral.

Function subprograms `RELIKC`, `RELIEC` and `DELIKC`, `DELIEC` calculate the complete elliptic integrals $\mathrm{K}(k)$ and $\mathrm{E}(k)$.

On CDC and Cray computers, the double-precision versions `DELI1C` etc. are not available.

Mainly for reasons of numerical stability, the algorithms are based on the following definitions:

First kind:

$$\mathbf{F}_1^*(k') = \int_0^\infty \frac{d\xi}{\sqrt{(1+\xi^2)(1+k'^2\xi^2)}} \qquad (k'^2 > 0).$$

Second kind:

$$\mathbf{F}_2^*(k',a,b) = \int_0^\infty \frac{a+b\xi^2}{(1+\xi^2)\sqrt{(1+\xi^2)(1+k'^2\xi^2)}}\,d\xi \qquad (k'^2 > 0).$$

Third kind:

$$\mathbf{F}_3^*(k',p) = \int_0^\infty \frac{1+\xi^2}{(1+p\xi^2)\sqrt{(1+\xi^2)(1+k'^2\xi^2)}}\,d\xi \qquad (k'^2 > 0,\ p \neq 0).$$

Note that $\mathbf{F}_1^*(k') = \mathbf{F}_2^*(k',1,1) = \mathbf{F}_3^*(k',1)$. For $p < 0$, the integral $\mathbf{F}_3^*$ is defined by its principal value.

The general integral:

$$\begin{aligned}\mathbf{G}(k',p,a,b) &= \int_0^\infty \frac{a+b\xi^2}{(1+p\xi^2)\sqrt{(1+\xi^2)(1+k'^2\xi^2)}}\,d\xi \\ &= \int_0^{\pi/2} \frac{a\cos^2\phi + b\sin^2\phi}{\cos^2\phi + p\sin^2\phi}\frac{d\phi}{\sqrt{\cos^2\phi + k'^2\sin^2\phi}} \qquad (k'^2 > 0).\end{aligned}$$

For $p < 0$, this integral is defined by its principal value. See **Notes** for special cases.

The functions K(k) and E(k):

$$\begin{aligned}\mathrm{K}(k) &= \int_0^{\pi/2} \frac{d\psi}{\sqrt{1-k^2\sin^2\psi}} \qquad (|k| < 1), \\ \mathrm{E}(k) &= \int_0^{\pi/2} \sqrt{1-k^2\sin^2\psi}\,d\psi \qquad (|k| \le 1).\end{aligned}$$

Other common definitions of the complete elliptic integrals and their relations to $\mathbf{F}_1^*$, $\mathbf{F}_2^*$, $\mathbf{F}_3^*$ are listed here for convenience ($k^2 + k'^2 = 1$):

First kind:

$$F(k,\pi/2) = \mathrm{K}(k) = \mathbf{F}_1^*(k') \qquad (|k|<1),$$

$$\widehat{F}(1,k) = \int_0^1 \frac{d\eta}{\sqrt{(1-\eta^2)(1-k^2\eta^2)}} = \mathbf{F}_1^*(k') \qquad (|k|<1).$$

Second kind:

$$E(k,\pi/2) = \mathrm{E}(k) = \mathbf{F}_2^*(k',1,k'^2) \qquad (|k|\le 1),$$

$$\widehat{E}(1,k) = \int_0^1 \sqrt{\frac{1-k^2\eta^2}{1-\eta^2}}\,d\eta = \mathbf{F}_2^*(k',1,k'^2) \qquad (|k|\le 1).$$

Third kind:

$$\Pi(\pi/2,h,k) = \int_0^{\pi/2} \frac{d\psi}{(1+h\sin^2\psi)\sqrt{1-k^2\sin^2\psi}} = \mathbf{F}_3^*(k',h+1) \qquad (|k|<1),$$

$$\widehat{\Pi}(1,h,k) = \int_0^1 \frac{d\eta}{(1+h\eta^2)\sqrt{(1-\eta^2)(1-k^2\eta^2)}} = \mathbf{F}_3^*(k',h+1) \qquad (|k|<1).$$

Structure:

`FUNCTION` subprograms
User Entry Names: `RELI1C`, `RELI2C`, `RELI3C`, `RELIGC`, `RELIKC`, `RELIEC`
`DELI1C`, `DELI2C`, `DELI3C`, `DELIGC`, `DELIKC`, `DELIEC`
Obsolete User Entry Names: `ELLICK` ≡ `RELIKC`, `ELLICE` ≡ `RELIEC`,
`DELLIK` ≡ `DELIKC`, `DELLIE` ≡ `DELIEC`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression, with `AK` $= k$ and `AKP` $= k'$,

`RELI1C(AKP)`	or	`DELI1C(AKP)`	has the value	$\mathbf{F}_1^*(k')$,
`RELI2C(AKP,A,B)`	or	`DELI2C(AKP,A,B)`	has the value	$\mathbf{F}_2^*(k',\mathtt{A},\mathtt{B})$,
`RELI3C(AKP,AK2,P)`	or	`DELI3C(AKP,AK2,P)`	has the value	$\mathbf{F}_3^*(k',\mathtt{P})$,
`RELIGC(AKP,P,A,B)`	or	`DELIGC(AKP,P,A,B)`	has the value	$\mathbf{G}(k',\mathtt{P},\mathtt{A},\mathtt{B})$,
`RELIKC(AK)`	or	`DELIKC(AK)`	has the value	$\mathrm{K}(k)$,
`RELIEC(AK)`	or	`DELIEC(AK)`	has the value	$\mathrm{E}(k)$,

where `RELI1C` etc are of type `REAL`, `DELI1C` etc are of type `DOUBLE PRECISION`, and `AKP`, `AK`, `AK2`, `A`, `B` and `P` have the same type as the function name.

The redundant parameter `AK2` in `RELI3C` and `DELI3C` permits improved accuracy when k^2 is small, i.e. $k' \approx 1$. In this case, `AK2` $= k^2$ should be calculated using higher-precision arithmetic and then truncated before calling the subprogram.

Method:

The evaluation of $\mathbf{F}_1^*$, $\mathbf{F}_2^*$, $\mathbf{F}_3^*$ is based on the Landen transformation, that of $\mathbf{G}$ on the Bartky transformation. For details, see Ref. 1–3. For $\mathrm{K}(k)$ and $\mathrm{E}(k)$ Chebyshev approximations are used (see Ref. 4).

Accuracy:

The `REAL` functions (except on CDC and Cray computers) have full single-precision accuracy. The `REAL` functions on CDC and Cray, and the `DOUBLE PRECISION` functions on all computers have an accuracy approximately two significant digits less than the machine precision.

Restrictions:

1. `RELI1C` and `DELI1C`: `AKP` $\neq$ `0`.
2. `RELI2C` and `DELI2C`: `AKP` $\neq$ `0` or `AKP` $=$ `0` and `B` $=$ `0`.
3. `RELI3C` and `DELI3C`: `AKP*P` $\neq$ `0`.
4. `RELIGC` and `DELIGC`: `AKP` $\neq$ `0`.
5. `RELIKC` and `DELIKC`: $|$`AK`$| \leq$ `1`,   `RELIEC` and `DELIEC`: $|$`AK`$| <$ `1`.

Error handling:

Error `C347.1`: Restriction 1 is not satisfied.
Error `C347.2`: Restriction 2 is not satisfied.
Error `C347.3`: Restriction 3 is not satisfied.
Error `C347.4`: Restriction 4 is not satisfied.
Error `C347.5`: Restriction 5 is not satisfied.
In all cases, the function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Notes:

Every linear combination of the three special complete elliptic integrals $\mathrm{K}(k)$, $\mathrm{E}(k)$, $\Pi(h,k)$ may be expressed in terms of $\mathbf{G}(k',p,a,b)$:

$$\begin{aligned}\lambda\mathrm{K}(k)+\mu\mathrm{E}(k) &= \mathbf{G}(k',1,\lambda+\mu,\lambda+\mu k'^2)\\ \lambda\mathrm{K}(k)+\mu\Pi(h,k) &= \mathbf{G}(k',h+1,\lambda+\mu,\lambda(h+1)+\mu)\end{aligned}$$

Special examples are

$$\begin{aligned}\mathrm{K}(k) &= \mathbf{G}(k',1,1,1),\\ \mathrm{E}(k) &= \mathbf{G}(k',1,1,k'^2)\\ (\mathrm{K}(k)-\mathrm{E}(k))/k^2 &= \mathbf{G}(k',1,0,1),\\ (\mathrm{K}(k)-k'^2\mathrm{E}(k))/k^2 &= \mathbf{G}(k',1,1,0),\\ \Pi(h,k) &= \mathbf{G}(k',h+1,1,1),\\ (\mathrm{K}(k)-\Pi(h,k))/h &= \mathbf{G}(k',h+1,0,1),\end{aligned}$$

If $ab \geq 0$ then $\mathbf{G}$ will evaluate any linear combination of $\mathrm{K}(k)$, $\mathrm{E}(k)$, $\Pi(h,k)$ without cancellation (such as would occur, for example, if $(\mathrm{K}(k){-}\mathrm{E}(k))/k^2$ were to be computed from values of $\mathrm{K}(k)$ and $\mathrm{E}(k)$ which had been computed separately.
Other functions which can be represented by $\mathbf{G}$ are the Jacobian Zeta function $\mathbf{Z}(\Phi,k)$ and the Heuman Lambda function $\Lambda_0(\Phi,k)$ (see Ref. 5):

$$\begin{aligned}\mathbf{Z}(\Phi,k) &= k^2\frac{\sin\Phi\cos\Phi}{\mathrm{K}(k)}\mathbf{G}(k',q,0,\sqrt{q}) \qquad (q=\cos^2\Phi+k'^2\sin^2\Phi)\\ \Lambda_0(\Phi,k) &= \frac{2}{\pi}\sqrt{q}\sin\Phi\,\mathbf{G}(k',q,1,k'^2) \qquad (q=1+k^2\tan^2\Phi).\end{aligned}$$

(Quoted from Ref. 3, slightly modified).

Source:

The subprograms for $\mathbf{F}_1^*$, $\mathbf{F}_2^*$ are based on the Algol60 procedures *cel1, cel2* in Ref. 1, those for $\mathbf{F}_3^*$ on *cel3* in Ref. 2, and those for $\mathbf{G}$ on *cel* in Ref. 3.

References:

1. R. Bulirsch, Numerical calculation of elliptic integrals and elliptic functions, Numer. Math. **7** (1965) 78–90.
2. R. Bulirsch, Numerical calculation of elliptic integrals and elliptic functions II, Numer. Math. **7** (1965) 353–354.
3. R. Bulirsch, Numerical calculation of elliptic integrals and elliptic functions III, Numer. Math. **13** (1969) 305–315.
4. W.J. Cody, Chebyshev approximations for the complete elliptic integrals K and E, Math. Comp. **19** (1965) 105–112.
5. P.F. Byrd and M.D. Friedman, Handbook of elliptic integrals for engineers and scientists, 2nd ed., Springer-Verlag Berlin (1971) 33–37.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Elliptic Integral for Complex Argument

Function subprograms `CELINT` and `WELINT` calculate, for complex argument $z = x + iy$ and real complementary modulus k' a general elliptic integral of the second kind:

$$\mathbf{F}(z, k', a, b) = \int_0^z \frac{a + b\zeta^2}{(1+\zeta^2)\sqrt{(1+\zeta^2)(1+k'^2\zeta^2)}}\, d\zeta \qquad (k'^2 \geq 0,\ \mathrm{Re}(z) \geq 0),$$

which contains the elliptic integrals of the first and second kind as special cases:

$$\mathbf{F}_1(z, k') = \int_0^z \frac{d\zeta}{\sqrt{(1+\zeta^2)(1+k'^2\zeta^2)}} = \mathbf{F}(z, k', 1, 1),$$

$$\mathbf{F}_2(z, k') = \int_0^z \frac{d\zeta}{(1+\zeta)^2}\sqrt{\frac{1+k'^2\zeta^2}{1+\zeta^2}} = \mathbf{F}(z, k', 1, k'^2).$$

The double-precision version `WELINT` is available only on computers which support a `COMPLEX*16` Fortran data type.

Structure:

`FUNCTION` subprograms
User Entry Names: `CELINT`, `WELINT`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression, with `AKP` $= k'$,

`CELINT(Z,AKP,A,B)` or `WELINT(Z,AKP,A,B)` has the value $\mathbf{F}(\mathtt{Z}, k', \mathtt{A}, \mathtt{B})$,

where `CELINT` is of type `COMPLEX`, `WELINT` is of type `COMPLEX*16`, `Z` is of the same type as the function name, and `AKP`, `A`, `B` are of type `REAL` for `CELINT` and of type `DOUBLE PRECISION` for `WELINT`.

Method:

The evaluation of **F** is based on the Gauss transformation. For details, in particular for the conformal mapping provided by **F**, see Ref. 1.

Accuracy:

`CELINT` (except on CDC and Cray computers) has full single-precision accuracy. For most values of the arguments, `WELINT` (and `CELINT` on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error `C348.1`: Re `Z` < 0. The function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Notes:

For other forms of the elliptic integrals see the write-up for `RELI1` (C346).

Source:

The subprogram is based on the Algol60 procedure *elco2* given in Ref. 1.

References:

1. R. Bulirsch, Numerical calculation of elliptic integrals and elliptic functions, Numer. Math. **7** (1965) 78–90.

•

Author(s) : G.A. Erskine **Library:** MATHLIB
Submitter : K.S. Kölbig **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Jacobian Theta Functions

Function subprograms RTHETA and DTHETA calculate the Jacobian theta functions

$$\begin{aligned}
\vartheta_0(x,q) &= 1+2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2n\pi x,\\
\vartheta_1(x,q) &= 2\sum_{n=0}^{\infty}(-1)^n q^{\left(n+\frac{1}{2}\right)^2}\sin(2n+1)\pi x,\\
\vartheta_2(x,q) &= 2\sum_{n=0}^{\infty} q^{\left(n+\frac{1}{2}\right)^2}\cos(2n+1)\pi x,\\
\vartheta_3(x,q) &= 1+2\sum_{n=1}^{\infty} q^{n^2}\cos 2n\pi x,\\
\vartheta_4(x,q) &= \vartheta_0(x,q),
\end{aligned}$$

for real arguments x and $0 \le q < 1$. $\vartheta_1(x+\frac{1}{2},1)$ and $\vartheta_2(x,1)$ are undefined if x is an integer; otherwise $\vartheta_k(x,1)=0,\ k=1,2,3,4.$

Note that several conflicting definitions of these functions occur in the literature. In particular, the argument in the trigonometric terms is often defined to be x instead of πx.

On CDC and Cray computers, the double-precision version DTHETA is not available.

Structure:

FUNCTION subprogram
User Entry Names: RTHETA, DTHETA
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

RTHETA(K,X,Q) or DTHETA(K,X,Q) has the value $\vartheta_{\mathtt{K}}(\mathtt{X},\mathtt{Q})$,

where RTHETA is of type REAL, DTHETA is of type DOUBLE PRECISION, X and Q are of the same type as the function name, and K is of type INTEGER.

Method:

If t $(0 \le t \le \frac{1}{2})$ differs from x or $-x$ by an integer, it follows from the periodicity and symmetry properties of the functions that $\vartheta_1(x,q)=\pm\vartheta_1(t,q)$ and $\vartheta_3(x,q)=\vartheta_3(t,q)$. In a region for which the approximation is sufficiently accurate, ϑ_1 is set equal to the first $(n=0)$ term of the transformed series

$$\vartheta_1(t,q) = 2(\lambda/\pi)^{1/2}e^{-\lambda t^2}\sum_{n=0}^{\infty}(-1)^n e^{-\lambda(n+\frac{1}{2})^2}\sinh(2n+1)\lambda t,$$

and ϑ_3 is set equal to the first two (i.e. $n \le 1$) terms of

$$\vartheta_3(t,q) \;=\; (\lambda/\pi)^{1/2} e^{-\lambda t^2}\left(1 + 2\sum_{n=1}^{\infty} e^{-\lambda n^2}\cosh 2n\lambda t\right),$$

where $\lambda = \pi^2/|\ln q|$. Otherwise the trigonometric series for $\vartheta_1(t,q)$ and $\vartheta_3(t,q)$ are used.
For all x, ϑ_0 and ϑ_2 are computed from $\vartheta_0(x,q) = \vartheta_3(\frac{1}{2} - |x|, q)$, $\vartheta_2(x,q) = \vartheta_1(\frac{1}{2} - |x|, q)$.

Restrictions:

1. $\mathtt{0} \le \mathtt{Q} \le \mathtt{1}$.
2. $\mathtt{K} = \mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}$.
3. If $\mathtt{Q} = \mathtt{1}$ and $\mathtt{K} = \mathtt{1}$, $\mathtt{X} - \frac{1}{2}$ must not be an integer.
   If $\mathtt{Q} = \mathtt{1}$ and $\mathtt{K} = \mathtt{2}$, `X` must not be an integer.

Error handling:

Error `C349.1:` Restriction 1 is not satisfied.
Error `C349.2:` Restriction 2 is not satisfied.
Error `C349.3:` Restriction 3 is not satisfied.
In all cases, the function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Accuracy:

For `DTHETA` (and for `RTHETA` on CDC and Cray computers), the error when `Q` is less than approximately `0.9` does not exceed two decimal digits in the last place. For larger values of `Q` (provided the computed result is non-zero), the error is at worst comparable in magnitude to the mathematical error which would be caused by one-bit rounding errors in the arguments `X` and `Q`.

On computers other than CDC and Cray, non-zero values of `RTHETA` have full machine accuracy.

Notes:

Successive references using the same value of `Q` are executed faster than those in which `Q` changes.

Many functional relations, including relations between the theta functions and the Jacobian elliptic functions, are given in Refs. 1–4.

References:

1. W. Magnus, F. Oberhettinger and R.P. Soni, Formulas and theorems for the special functions of mathematical physics, Springer-Verlag Berlin (1966) 371–377.
2. F. Tölke, Praktische Funktionenlehre, Bd. II, Springer-Verlag Berlin (1966) 1–38.
3. P.F. Byrd and M.D. Friedman, Handbook of elliptic integrals for engineers and scientists, 2nd Edition, Springer-Verlag Berlin (1971) 315–320.
4. E.T. Whittaker and G.N. Watson, A course of modern analysis, 4th Edition, Cambridge University Press, Cambridge (1946) Chapter 21.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.01.1988
Language : Fortran **Revised:** 15.03.1993

Integration by Simpson's Rule

Function subprograms SIMPS and DSIMPS use Simpson's rule to compute an approximate value of the integral

$$I = \int_A^B f(x)dx.$$

On CDC or Cray computers, the double-precision version DSIMPS is not available.

Structure:

FUNCTION subprograms
User Entry Names: SIMPS, DSIMPS
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

SIMPS(F,A,B,N) or DSIMPS(F,A,B,N)

has the approximate value of the integral I, where SIMPS is of type REAL and DSIMPS is of type DOUBLE PRECISION, and F, A, B have the same type as the function name. N is of type INTEGER.

F — One-dimensional array with dimension (0:d), where $\mathtt{d} \geq \mathtt{N}$, containing the value of $f(x)$ at $\mathtt{N}+1$ equally-spaced points x_i, $(i = 0, 1, \ldots, \mathtt{N})$, with $x_0 = \mathtt{A}$ and $x_{\mathtt{N}} = \mathtt{B}$.

A,B — End-points of integration interval.

N — As defined above. N must be positive and even.

Error handling:

Error D101.1: $\mathtt{N} \leq 0$ or N odd. The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

•

RADAPT	CERN Program Library	D102

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 01.12.1994
Language : Fortran **Revised:**

Adaptive Gaussian Quadrature

Subroutine subprograms RADAPT and DADAPT calculate, to an attempted specified accuracy, the value of the integral

$$I = \int_a^b f(x)\,dx$$

by adaptive subdivision of the interval (a, b), calculating the integrals over the subintervals using RGS56P and DGS56P (D106).

On CDC and Cray computers, the double-precision version DADAPT is not available.

Structure:

SUBROUTINE subprograms
User Entry Names: RADAPT, DADAPT
External References: RGS56P (D106), DGS56P (D106), user-supplied FUNCTION subprogram.

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tADAPT(F,A,B,NSEG,RELTOL,ABSTOL,RES,ERR)
```

F — (type according to t) Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must set F(X) $= f($X$)$.

A,B — (type according to t) End-points of integration interval. Note that B may be less than A.

NSEG — (INTEGER) Specifies how the adaptation is to be done:
= 0 : use the subdivisions as determined in the previous call to tADAPT,
= 1 : fully automatic, adapt until tolerance attained,
= n > 1 : first split interval into n equal segments, then adapt as necessary to attain tolerance.

RELTOL — (type according to t) Specified *relative* tolerance.

ABSTOL — (type according to t) Specified *absolute* tolerance.

The calculation comes to an end if either RELTOL or ABSTOL is satisfied, or the number of segments exceeds 100. Either RELTOL or ABSTOL can be set to zero, in which case only the other is used.

RES — (type according to t) The calculated approximation for I.

ERR — (type according to t) An estimated absolute uncertainty on this approximation.

Method:

The automatic adaption is done as follows: At each step, the total integral is estimated as the sum of the integrals over the subdivisions, and the squared uncertainty is estimated as the sum of the squares of the uncertainties over all subdivisions. If this uncertainty is too big (failing both the absolute and relative tolerance criteria) then the subinterval with the largest absolute uncertainty is divided in half.

Accuracy:

The true accuracy is usually very close to the uncertainty returned by the subroutine, sometimes it is much better, but very seldom worse. Even on functions with (integrable) singularities, the results are usually reliable, as long as the singularity is "wide enough" to be detected in the early stages, which can be controlled by the value of `NSEG`.

•

Author(s) : K.S. Kölbig
Library: MATHLIB

Submitter :
Submitted: 02.05.1966

Language : Fortran
Revised: 15.03.1993

Adaptive Gaussian Quadrature

Function subprograms GAUSS, DGAUSS and QGAUSS compute, to an attempted specified accuracy, the value of the integral

$$I = \int_A^B f(x) dx.$$

The quadruple-precision version QGAUSS is available only on computers which support a REAL*16 Fortran data type.

Structure:

FUNCTION subprograms
User Entry Names: GAUSS, DGAUSS, QGAUSS
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035), user-supplied FUNCTION subprogram

Usage:

In any arithmetic expression,

```
GAUSS(F,A,B,EPS),   DGAUSS(F,A,B,EPS)   or   QGAUSS(F,A,B,EPS)
```

has the approximate value of the integral I.

F   Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must set $\mathtt{F(X)} = f(\mathtt{X})$.

A,B   End-points of integration interval. Note that B may be less than A.

EPS   Accuracy parameter (see **Accuracy**).

GAUSS is of type REAL, DGAUSS is of type DOUBLE PRECISION, QGAUSS is of type REAL*16, and the arguments F, A, B, EPS and X (in F) have the same type as the function name.

Method:

For any interval $[a, b]$ we define $g_8(a, b)$ and $g_{16}(a, b)$ to be the 8-point and 16-point Gaussian quadrature approximations to

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

and define

$$r(a, b) = \frac{|g_{16}(a, b) - g_8(a, b)|}{1 + |g_{16}(a, b)|}.$$

Then, with G = GAUSS or DGAUSS,

$$G = \sum_{i=1}^{k} g_{16}(x_{i-1}, x_i),$$

where, starting with $x_0 = A$ and finishing with $x_k = B$, the subdivision points x_i $(i = 1, 2, \ldots)$ are given by

$$x_i = x_{i-1} + \lambda(B - x_{i-1}),$$

with λ equal to the first member of the sequence $1, \frac{1}{2}, \frac{1}{4}, \ldots$ for which $r(x_{i-1}, x_i) < \texttt{EPS}$. If, at any stage in the process of subdivision, the ratio

$$q = \left| \frac{x_i - x_{i-1}}{B - A} \right|$$

is so small that $1 + 0.005q$ is indistinguishable from 1 to machine accuracy, an error exit occurs with the function value set equal to zero.

Accuracy:

Unless there is severe cancellation of positive and negative values of $f(x)$ over the interval $[A, B]$, the argument `EPS` may be considered as specifying a bound on the *relative* error of I in the case $|I| > 1$, and a bound on the *absolute* error in the case $|I| < 1$. More precisely, if k is the number of sub-intervals contributing to the approximation (see Method), and if

$$I_{abs} = \int_A^B |f(x)| dx,$$

then the relation

$$\frac{|G - I|}{I_{abs} + k} < \texttt{EPS}$$

will nearly always be true, provided the routine terminates without printing an error message. For functions f having no singularities in the closed interval $[A, B]$ the accuracy will usually be much higher than this.

Error handling:

Error `D103.1`: The requested accuracy cannot be obtained (see **Method**). The function value is set equal to zero, and a message is written on `Unit 6` unless subroutine `MTLSET` (N002) has been called.

Notes:

Values of the function $f(x)$ at the interval end-points A and B are not required. The subprogram may therefore be used when these values are undefined.

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 10.08.1967
Language : Fortran | **Revised:** 01.12.1994

Cauchy Principal Value Integration

Function subprograms RCAUCH and DCAUCH compute the Cauchy principal value integral

$$I = P\int_A^B f(x)dx$$

where f has a singularity inside or outside the interval $[A, B]$ such that the Cauchy principal value exists. On computers other than CDC or Cray, only the double-precision version DCAUCH is available. On CDC and Cray computers, only the single-precision version RCAUCH is available.

Structure:

FUNCTION subprograms
User Entry Names: RCAUCH, DCAUCH
Obsolete User Entry Names: CAUCHY ≡ RCAUCH
Files Referencend: Unit 6
External References: GAUSS (D103), DGAUSS (D103), MTLMTR (N002),
ABEND (Z035), user-supplied FUNCTION subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
tCAUCH(F,A,B,S,EPS)
```

has, in any arithmetic expression, the approximate value of the integral I.

F (type according to t) Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must set F(X) = f(X).

A,B (type according to t) End-points of the integration interval. Note that B may be less than A.

S (type according to t) The absissa of the singularity.

EPS (type according to t) Accuracy parameter (see under Accuracy in the in short write-up for GAUSS (D103)).

Method:

The method described in Ref. 1 is used for decomposition of the integral. The resulting integrals are computed by GAUSS (D103).

Accuracy:

See short write-up for GAUSS (D103).

Error handling:

Error D104.1: S = A or S = B.
Error D104.2: The requested accuracy cannot be obtained (see short write-up for GAUSS (D103)).
The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. I.M. Longman, On the numerical evaluation of Cauchy principal values of integrals, MTAC (later renamed Math. Comp.) **12** (1958) 205–207.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 02.05.1966
Language : Fortran **Revised:** 01.12.1994

Integration over a Triangle

Function subprograms RTRINT and DTRINT compute an approximate value of the integral

$$I = \int\int f(x,y)\,dxdy,$$

evaluated over the interior of an arbitrary triangle Δ in the xy-plane. An attempted accuracy may, optionally, be specified.

On computers other than CDC or Cray, only the double-precision version DTRINT is available. On CDC and Cray computers, only the single-precision version RTRINT is available.

Structure:

FUNCTION subprograms
User Entry Names: RTRINT, DTRINT
Obsolete User Entry Names: TRIINT $\equiv$ RTRINT
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
tTRINT(F,NSD,NPT,EPS,X1,Y1,X2,Y2,X3,Y3)
```

has, in any arithmetic expression, the approximate value of the integral I.

F (type according to t) Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must set F(X, Y) = f(X, Y).

NSD (INTEGER)
= 0 : No subdivision of the given triangle.
= 1 : Subdivision of the given triangle (see Method).

NPT (INTEGER)
= 7 : A 7-point integration formula is used.
= 25 : A 25-point integration formula is used.
= 64 : A 64-point integration formula is used.

EPS (type according to t) Accuracy parameter (see **Accuracy**).

X1,Y1 (type according to t) The coordinates of the vertices of Δ.
X2,Y2
X3,Y3

Method:

$\texttt{NSD} = \texttt{0}:$
An approximation I_0 to I is found by computing the NPT-point formula for the triangle Δ. The value of EPS has no influence on the result.
$\texttt{NSD} = \texttt{1}:$
After computing I_0, the triangle Δ is subdivided into two subtriangles Δ' and Δ'', the corresponding approximations I' and I'' are computed, and a test is made to see whether

$$\frac{|I_0 - (I' + I'')|}{1 + |I' + I''|} < \texttt{EPS}$$

If this test is satisfied, the routine terminates by setting the function value to I_0. If it fails, the process of subdivision and testing continues according to a tree structure. The routine terminates either because the test is passed successfully by all the subtriangles at some level, or because a maximum number of subdivisions is reached (see **Error Handling**).

Accuracy:

Unless there is severe cancellation of positive and negative values of $f(x, y)$ over Δ,the argument EPS may, if $\texttt{NCD} = \texttt{1}$, be considered as specifying a bound on the relative error of I in the case $|I| > 1$, and a bound on the absolute error in the case $|I| < 1$.

Restrictions:

”Mild” singularities are permitted if they coincide with the vertices of Δ. Any other singularity lying inside Δ or on its boundaries will most likely lead to too many subdivisions (see **Error Handling**), or cause a wrong result.

Error handling:

Error D105.1: $\texttt{NPT} \neq \texttt{7}, \texttt{25}, \texttt{64}.$
Error D105.2: The number of subdivisions has reached 35 without success.
In both cases, the function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. K.S. Kölbig, A Fortran program and some numerical test results for the integration over a triangle, CERN 64–32 (1964).

•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 01.12.1994
Language : Fortran **Revised:**

Gaussian Quadrature with Five- and Six-Point Rules

Subroutine subprograms RGS56P and DGS56P calculate an approximation and uncertainty for the integral

$$I = \int_a^b f(x)\,dx$$

equal respectively to the mean value and the difference of the results I_5 and I_6 obtained by the five- and six-point Gaussian integration rules.

On CDC and Cray computers, the double-precision version DGS56P is not available.

Structure:

SUBROUTINE subprograms
User Entry Names: RGS56P, DGS56P
External References: User-supplied FUNCTION subprogram.

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tGS56P(F,A,B,RES,ERR)
```

F — (type according to t) Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must set F(X) $= f($X$)$.

A,B — (type according to t) End-points of integration interval. Note that B may be less than A.

RES — (type according to t) The calculated approximation for I, i.e. $\frac{1}{2}(I_5 + I_6)$,

ERR — (type according to t) An estimated uncertainty on this approximation, i.e. $|I_5 - I_6|$.

•

Author(s) : G.A. Erskine **Library:** MATHLIB
Submitter : K.S. Kölbig **Submitted:** 07.06.1992
Language : Fortran **Revised:**

N-Point Gaussian Quadrature

Function subprograms `RGQUAD` and `DGQUAD` calculate the approximate value of the integral

$$\int_a^b f(t)dt$$

using the N-point Gauss-Legendre quadrature formula corresponding to the interval $[a, b]$.

Subroutine subprograms `RGSET` and `DGSET` store, for subsequent use, the abscissae x_i and the weights w_i of the N-point Gauss-Legendre quadrature formula corresponding to the interval $[a, b]$.

The following values of N may be used: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 32, 40, 48, 64, 80, 96.

`RGQUAD`, `RGSET` and `DGQUAD`, `DGSET` are independent subprograms: it is not necessary, for instance, to call `DGSET` in order to use `DGQUAD`, or vice-versa.

On CDC and Cray computers, the double-precision versions `DGQUAD` and `DGSET` are not provided.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `RGQUAD`, `RGSET`, `DGQUAD`, `DGSET`
Internal Entry Names: `D107R1`, `D107D1`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035), User-supplied `FUNCTION` subprogram

Usage:

To calculate the integral:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
tGQUAD(F,A,B,N)
```

has, in any arithmetic expression, the value $\sum_{i=1}^{N} w_i f(x_i)$ which is an approximation to the given integral.

To store the abscissae x_i and the weights w_i:

```
CALL tGSET(A,B,N,X,W)
```

`F` (type according to `t`) Name of a user-supplied `FUNCTION` subprogram, declared `EXTERNAL` in the calling program. This subprogram must set `F(X)` $= f($`X`$)$.

`A,B` (type according to `t`) End-points a and b of the integration interval.

`N` (`INTEGER`) Number N of quadrature points.

`X,W` (type according to `t`) One-dimensional arrays. On exit, `X(i)` and `W(i)` contain x_i and w_i, $(i = 1, 2, \ldots, N)$, respectively.

Method:

The values of x_i and w_i are computed by linearly scaling values obtained from a stored table corresponding to the interval $[-1, +1]$.

Accuracy:

The absolute error of RGQUAD and DGQUAD is proportional to the value of the $(2N)$th derivative of f at some internal point of the interval $[a, b]$ (see Ref. 1).

Error handling:

Error D107.1: The value N does not appear in the list given above. A message is written on Unit 6, unless subroutine MTLSET (N002) has been called. If the subprogram referenced is RGQUAD or DGQUAD, the function value is set equal to zero.

References:

1. A.H. Stroud and D. Secrest, Gaussian quadrature formulas, (Prentice-Hall, Englewood Cliffs 1966).

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 01.03.1968
Language : Fortran **Revised:**

Trapezoidal Rule Integration with an Estimated Error

Let a function $f(x)$ be given by its values at certain discrete points $x_\nu(\nu = 1, 2, \ldots, n)$. Let the function values y_ν be accompanied by an estimated standard deviation ε_ν (square root of the variance). Subroutine subprogram `TRAPER` then approximates the integral

$$I = \int_A^B f(x)dx \simeq \sum_\nu w_\nu y_\nu$$

by a linear combination of the y_ν using the trapezoidal rule. It calculates the standard deviation σ of I by

$$\sigma = \sqrt{\sum_\nu w_\nu^2 \varepsilon_\nu^2}.$$

The function values $f(A)$ and $f(B)$ are calculated by linear interpolation.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `TRAPER`

Usage:

```
CALL TRAPER(X,Y,E,N,A,B,RE,SD)
```

`X,Y,E` (`REAL`) Arrays of length $\geq n$ containing x_ν, y_ν, ε_ν, respectively.

`N` (`INTEGER`) Number of function values

`A,B` (`REAL`) Limits of integration.

`RE` (`REAL`) On exit, `RE` contains an approximate value of the integral I.

`SD` (`REAL`) On exit, `SD` contains an approximate value of the standard deviation σ.

If no ε_ν are given, the array `E` should be filled with zeros.

Restrictions:

Although there are no restrictions on `A` and `B` (`B` may be less than `A`), care must be taken if one or both of them is either smaller than `X(1)` or bigger than `X(N)`. In these cases f(`A`) or f(`B`) are extrapolated linearly from `Y(1)` and `Y(2)` or `Y(N-1)` and `Y(N)` respectively, which may lead to unreasonable results. If `A` = `B` or `N` < `2`, `RE` and `SD` will be set to zero. It is assumed that all the x_ν are distinct. No test is made for this.

Notes:

This program should only be used for the problem described above. For general-purpose numerical integration to a preassigned accuracy use `GAUSS` (D103).

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.12.1988
Revised: 15.03.1993

Gaussian Quadrature for Multiple Integrals

Function subprogram packages RGMLT and DGMLT compute an approximate value of an n-dimensional integral of the form

$$I_n = \int_{a_n}^{b_n} dx_n \phi_n(x_n) \int_{a_{n-1}(x_n)}^{b_{n-1}(x_n)} dx_{n-1} \phi_{n-1}(x_{n-1}, x_n) \cdots$$

$$\phi_2(x_2, \ldots, x_n) \int_{a_1(x_2,\ldots,x_n)}^{b_1(x_2,\ldots,x_n)} dx_1 \phi_1(x_1, \ldots, x_n),$$

where $1 \leq n \leq 6$.

Each subprogram integrates over only one variable. The integral I_n is computed by means of a set of n nested calls to these subprograms.

On computers other than CDC or Cray, only the double-precision version DGMLT is available. On CDC and Cray computers, only the single-precision version RGMLT is available.

Structure:

FUNCTION subprograms
User Entry Names: RGMLT1, RGMLT2, RGMLT3, RGMLT4, RGMLT5, RGMLT6,
DGMLT1, DGMLT2, DGMLT3, DGMLT4, DGMLT5, DGMLT6
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035), user-supplied SUBROUTINE subprograms

Usage:

1. Let k be one of the integers 1, 2, . . . , 6. Then, in any arithmetic expression,

   RGMLTk(FSUBk,Ak,Bk,NIk,NGk,X) or
   DGMLTk(FSUBk,Ak,Bk,NIk,NGk,X)

   has the approximate value of the integral with respect to x_k of the function specified below.

   RGMLTk is of type REAL, DGMLTk is of type DOUBLE PRECISION, and the arguments Ak, Bk, and X have the same type as the function name. NIk and NGk are of type INTEGER.

   FSUBk — Name of a user-supplied SUBROUTINE subprogram, declared EXTERNAL in the calling program.
   Ak,Bk — End points of the integration interval for variable x_k.
   NIk — Number of equal subintervals into which the interval (Ak,Bk) is divided.
   NGk — Number of Gaussian quadrature points to be used in each of the NIk subintervals. (If NGk has any value other than 6 or 8, the value 6 is assumed).
   X — One-dimensional array of dimension $\geq n$.

2. The subroutine `FSUBk` should be of the form

   ```
   SUBROUTINE FSUBk (M,Uk,Fk,X)
   ```

   where `Uk`, `Fk` and `X` are of type `REAL` for `RGMLTk` and of type `DOUBLE PRECISION` for `DGMLTk`, and where `M` is of type `INTEGER`.

   `M` An integer ($\leq$ 64), whose value is set by `RGMLTk (DGMLTk)`.
   `Uk` One-dimensional array with declared dimension `(*)`, whose contents is set by `RGMLTk (DGMLTk)`.
   `Fk` One-dimensional array with declared dimension `(*)`, whose contents must be set by `FSUBk` as described below.
   `X` One-dimensional array which must be the same as the array `X` appearing as an actual argument in all calls to `RGMLT1`, `RGMLT2`,... (`DGMLT1`, `DGMLT2`, ...).

The subprogram `RGMLTk` (`DGMLTk`) which calls subroutine `FSUBk` sets the value of `M` and places in array `Uk` a set of `M` values of the variable x_k. Then, if $f_k(x_k, \dots, x_n)$ denotes the function which is to be integrated with respect to x_k, it is the job of subroutine `FSUBk` to set `X(k)` to the appropriate value of x_k, to compute f_k for each of these values of x_k (taking the remaining variables $x_{k+1}, \dots, x_n$ from `X(k+1)`,...,`X(n)` respectively) and place the results in array `Fk`. (See **Examples**).

Method:

Integration with respect to each variable is performed by applying the 6- or 8-point Gaussian quadrature formula to each of the equal sub-intervals.

Notes:

1. The time needed to compute an n-dimensional integral by means of these subprograms is approximately
   $$(\mathtt{NG1} * \mathtt{NG2} * \cdots * \mathtt{NGn}) * (\mathtt{NI1} * \mathtt{NI2} * \cdots * \mathtt{NIn}).$$
   This should be kept in mind when choosing the values of `NIk`.

2. The accuracy of a particular calculation can be estimated by repeating the integration with different values of `NGk` (e.g., 8 instead of 6) or by changing `NIk`, the latter usually being less economical.

Error handling:

Error `D110.1`: `NIk` $\leq$ `0`. A message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called. Execution is halted on a `STOP` instruction.

Examples:

To calculate (in type `REAL`) the integral

$$Q_2 \;=\; \int_0^1 dx_2\sqrt{x_2}e^{x_2}\int_0^{\sqrt{x_2}} dx_1 x_1\sqrt{x_1^2+x_2} \;=\; \frac{1}{3}(2\sqrt{2}-1)(e-2)$$

using 6-point Gaussian quadrature over each of $n_2 = 3$, $n_1 = 4$ subdivisions of the corresponding interval. In the main program, write for instance

```
      ...
      EXTERNAL FSUB2
      DIMENSION X(2)
      Q2=RGMLT2(FSUB2,0.,1.,3,6,X)
      ...
```

For the SUBROUTINE subprograms FSUB2, FSUB1 write for instance

```
      SUBROUTINE FSUB2(M,U2,F2,X)
      EXTERNAL FSUB1
      DIMENSION U2(*),F2(*),X(2)
      DO 1 L = 1,M
      X(2)=U2(L)
      R=SQRT(X(2))
    1 F2(L)=R*EXP(X(2))*RGMLT1(FSUB1,0.,R,4,6,X)
      RETURN
      END

      SUBROUTINE FSUB1(M,U1,F1,X)
      DIMENSION U1(*),F1(*),X(2)
      DO 1 L = 1,M
      X(1)=U1(L)
    1 F1(L)=X(1)*SQRT(X(1)**2+X(2))
      RETURN
      END
```

•

Author(s) : G.A. Erskine
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 07.12.1970
Revised: 15.03.1993

Adaptive Complex Integration Along a Line Segment

Function subprograms CGAUSS and WGAUSS compute, to an attempted specified accuracy, the value of the complex integral

$$I = \int_A^B f(z)dz.$$

The path of integration is the directed line segment AB in the complex z-plane. The function $f(z)$ must be single-valued on this segment.

The double-precision version WGAUSS is available only on computers which support a COMPLEX*16 Fortran data type.

Structure:

FUNCTION subprograms
User Entry Names: CGAUSS, WGAUSS
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035), user-supplied FUNCTION subprogram

Usage:

In any arithmetic expression,

CGAUSS(F,A,B,EPS) or WGAUSS(F,A,B,EPS)

has the approximate value of the integral I.

F	Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subroutine must set F(Z) $= f($Z$)$.
A,B	End-points of integration interval.
EPS	Accuracy parameter (see **Accuracy**).

CGAUSS is of type COMPLEX, WGAUSS is of type COMPLEX*16, and the arguments F, A, B, and Z (in F) have the same type as the function name. EPS is of type REAL for CGAUSS and of type DOUBLE PRECISION for WGAUSS.

Method:

For any line segment $[a, b]$ we define $g_8(a, b)$ and $g_{16}(a, b)$ to be the 8-point and 16-point Gaussian quadrature approximations to

$$\int_a^b f(z)dz$$

and define

$$r(a, b) = \frac{|g_{16}(a, b) - g_8(a, b)|}{1 + |g_{16}(a, b)|}.$$

Then, with G = `CGAUSS` or `WGAUSS`,

$$G = \sum_{i=1}^{k} g_{16}(z_{i-1}, z_i),$$

where, starting with $z_0 = A$ and finishing with $z_k = B$, the subdivision points z_i $(i = 1, 2, \dots)$ are given by

$$z_i = z_{i-1} + \lambda(B - z_{i-1}),$$

with λ equal to the first member of the sequence $1, 1/2, 1/4, \dots$ for which $r(z_{i-1}, z_i) <$ `EPS`. If, at any stage in the process of subdivision, the ratio

$$q = \left| \frac{z_i - z_{i-1}}{B - A} \right|$$

is so small that $1 + 0.005q$ is indistinguishable from 1 to machine accuracy, an error exit occurs with the function value set equal to zero.

Accuracy:

Unless there is severe cancellation of positive and negative values of $f(z)$ over the interval $[A, B]$, the argument `EPS` may be considered as specifying a bound on the *relative* error of I in the case $|I| > 1$, and a bound on the *absolute* error in the case $|I| < 1$. More precisely, if k is the number of sub-intervals contributing to the approximation (see **Method**), and if

$$I_{abs} = \int_A^B |f(z)| dz,$$

then the relation

$$\frac{|G - I|}{I_{abs} + k} < \texttt{EPS}$$

will nearly always be true, provided the routine terminates without printing an error message. For functions f having no singularities in the closed interval $[A, B]$ the accuracy will usually be much higher than this.

Error handling:

Error `D113.1`: The requested accuracy (see **Method**) cannot be obtained. The function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Notes:

Values of the function $f(z)$ at the end-points of the line segment A and B are not required. The subprogram may therefore be used when these values are undefined.

•

Author(s) : B. Lautrup
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 23.07.1971
Revised: 10.01.1986

Adaptive Multidimensional Monte-Carlo Integration

OBSOLETE

Please note that this routine has been obsoleted in CNL 223. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: (in part) `RADMUL` (D120)

`RIWIAD` is an adaptive multidimensional integration subroutine based on an original by G. Sheppey. It permits numerical integration of a large class of functions, in particular those that are irregular at the border of the integration region. The integral is always performed over the unit hypercube.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `RIWIAD`
Files Referenced: `Unit 6`
External References: `RNDM` (V104) user-supplied `FUNCTION` subprogram
`COMMON` Block Names and Lengths: `/ANSWER/ 2, /INTERN/ 7, /OPTION/ 3, /PARAMS/ 4, /RANDOM/ 1, /STORE/ 77, /STORE1/ 10001`

Usage:

See **Long Write-up** for a description of all features. Here only the standard use is described.
The `COMMON` block `PARAMS` must always be set by the user:

```
COMMON /PARAMS/ ACC,NDIM,NSUB,ITER
```

`ACC` (`REAL`) Relative accuracy desired.

`NDIM` (`INTEGER`) Number of dimension parameters.

`NSUB` (`INTEGER`) Number of subvolumes allowed.

`ITER` (`INTEGER`) Maximal number of iterations.

The integrand is defined by a user-supplied `FUNCTION` subprogram having the array `Q(NDIM)` as parameter, for example

```
FUNCTION EXAMPLE(Q)
REAL EXAMPLE,Q
DIMENSION Q(7)
...
END
```

This program defines EXAMPLE as a function of the 7 variables Q(1), Q(2), ..., Q(7). The sequence

```
EXTERNAL EXAMPLE
COMMON /PARAMS/ ACC,NDIM,NSUB,ITER
ACC=0.01
NDIM=7
NSUB=10000
ITER=5
CALL RIWIAD(EXAMPLE)
...
```

will then integrate EXAMPLE over the 7 variables Q(1), ..., Q(7), all in the interval from 0 to 1, i.e. over the 7-dimensional unit hypercube. The result will be printed in detail in a readily understandable form.

The program allows extensive user control via the COMMON blocks. See **Long Write-up** for details.

Method:

RIWIAD is iterative and in a given iteration it divides the unit hypercube into a certain number of subvolumes by means of a given set of intervals on each axis. Within each subvolume it estimates the mean value and variance of the integrand by random sampling, and then calculates the Riemann sum over the subvolumes. Using the variances found projected onto each axis it calculates a set of new interval divisions to be used in the next iteration. It returns when the desired accuracy is obtained or when the maximum number of iterations has been performed.

Restrictions:

There is, in principle, no limitations on the number of dimensions, although the present version only allows up to 9-dimensional integrals. The maximal dimensionality can easily be increased.

Notes:

1. The program is rather slow and should preferably be used only when other methods (for instance Gaussian quadrature) fail due to the irregular behaviour of the integrand. The time consumption is essentially proportional to the product of NSUB and ITER.
2. The non-CDC/Cray implementation of RIWIAD has IMPLICIT DOUBLE PRECISION(A-H,O-Z), i.e. all non-INTEGER variables are DOUBLE PRECISION, including the user-supplied external function.

•

Author(s) : A.C. Genz, A.A. Malik
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 15.11.1995
Revised:

Adaptive Quadrature for Multiple Integrals over N-Dimensional Rectangular Regions

Subroutine subprograms RADMUL and DADMUL compute, to an attempted specified accuracy, the value of the integral

$$I_n = \int_{a_n}^{b_n} \int_{a_{n-1}}^{b_{n-1}} \cdots \int_{a_1}^{b_1} f(x_1, x_2, \ldots, x_n)\, dx_1\, dx_2 \cdots dx_n$$

over an n-dimensional rectangular region, where $a_i,\ b_i$, $(i = 1, 2, \ldots, n)$ are constants.

On computers other than CDC and Cray, only the double-precision version DADMUL is available. On CDC and Cray computers, only the single-precision version RADMUL is available.

Structure:

SUBROUTINE subprograms
User Entry Names : RADMUL, DADMUL
External References: User-supplied SUBROUTINE subprogram

Usage:

For t $=$ R (type REAL), t $=$ D (type DOUBLE PRECISION),

```
CALL tADMUL(F,N,A,B,MINPTS,MAXPTS,EPS,WK,IWK,RESULT,RELERR,NFNEVL,IFAIL)
```

F (type according to t) Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program.

N (INTEGER) Number n of dimensions ($2 \le$ N ≤ 15).

A,B (type according to t) One-dimensional arrays of length $\ge$ N. On entry, A(i) and B(i), (i $= 1, \ldots,$ N), contain the lower and upper limits of integration, respectively. Note that $a_i,\ b_i$ correspond to x_i.

MINPTS (INTEGER) Minimum number of function evaluations requested. Must not exceed MAXPTS.

MAXPTS (INTEGER) Maximum number ($\ge$ 2**N + 2N(N + 1) + 1) of function evaluations to be allowed.

EPS (type according to t) Specified *relative* accuracy.

WK (type according to t) One-dimensional array of length IWK, used as working space.

IWK (INTEGER) Length ($\ge$ (2N + 3) $*$ (1 + MAXPTS/(2**N + 2N(N + 1) + 1))/2) of WK.

RESULT (type according to t) Contains, on exit, an approximate value of the integral I_n.

RELERR (type according to t) Contains, on exit, an estimation of the relative accuray of RESULT.

NFNEVL (INTEGER) Contains, on exit, the number of function evaluations performed.

IFAIL (INTEGER) On exit:

0 Normal exit. RELERR $<$ EPS. At least MINPTS and at most MAXPTS calls to the function F were performed.

1 MAXPTS is too small for the specified accuracy EPS. RESULT and RELERR contain the values obtainable for the specified value of MAXPTS.

2 IWK is too small for the specified number MAXPTS of function evaluations. RESULT and RELERR contain the values obtainable for the specified value of IRK.

3 N < 2, or N > 15, or MINPTS $>$ MAXPTS, or MAXPTS $<$ 2**N $+$ 2N(N $+$ 1) $+$ 1. RESULT and RELERR are set equal to zero.

The user-supplied FUNCTION subprogram F should be of the form

```
FUNCTION F(N,X)
DIMENSION X(*)
...
F =f(X(1), ..., X(N)).
RETURN
END
```

where X and F are of type t.

Method:

An integration rule of degree seven is used together with a certain strategy of subdivision. For a more detailed description of the method see **References**.

Error handling:

See description of argument IFAIL.

Notes:

1. Multi-dimensional integration is time-consuming. For each rectangular subregion, the routine requires $2^n + 2n^2 + 2n + 1$ function evaluations. Careful programming of the integrand might result in substantial saving of time.
2. Numerical integration usually works best for smooth functions. Some analysis or suitable transformations of the integral prior to numerical work may contribute to numerical efficiency.

Source:

This subroutine is an adapted version of Fortran program ADAPT published in Ref. 1.

References:

1. A.C. Genz and A.A. Malik, Remarks on algorithm 006: An adaptive algorithm for numerical integration over an N-dimensional rectangular region, J. Comput. Appl. Math. **6** (1980) 295–302.
2. A. van Doren and L. de Ridder, An adaptive algorithm for numerical integration over an n-dimensional cube, J. Comput. Appl. Math. **2** (1976) 207–217.

A copy of the text part of the References is available.

•

Author(s) : J.H. Friedman, M.H. Wright (Stanford) **Library:** MATHLIB
Submitter : F. James **Submitted:** 01.12.1981
Language : Fortran **Revised:** 14.08.1985

Multidimensional Integration or Random Number Generation

OBSOLETE

Please note that this routine has been obsoleted in CNL 223. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: (in part) `RADMUL` (D120)

`DIVON4` is designed for integration of scalar functions of several variables, especially functions not smooth enough to be integrated reliably using Gaussian quadrature. It can also be used effectively to generate random points in a multidimensional space, with point density given by any bounded function. The heart of the package is an algorithm for recursive multi-dimensional partitioning of the space into subregions of approximately constant function value (minimum range criterion).

Structure:

`SUBROUTINE` package

User Entry Names: `BUKDMP, DIVON, DVNOPT, GENPNT, INTGRL, PARTN, RANGEN, TREDMP, USRINT, USRTRM, DVNBKD, EXMBUC, SPLIT, QUASI, RECPAR, BOUNDS, TREAUD, NODAUD, BUCMVE, QUAD, FEQN, NOCUT, TSTEXT, DELSLV, FUN, BUFOPT, BNDOPT, SETTOL, BNDTST, DVCOPY, GRDCMP, DELETE, BFGS, MODCHL, NMDCHL, DVDOT, LDLSOL, SHRNK, FEASMV, ADDBND, MULCHK, DELBND, LOCSCH, ORTHVC, MXSTEP, NEWPTR, RLEN, RANUMS`

Files Referenced: Printer and optional user-defined external file

External References: `NRAN` (V105), user-supplied `FUNCTION` subprogram `DFUN`

Usage:

The function (integrand) is defined by a user-supplied `FUNCTION` subprogram which must have the name `DFUN` and must calculate the integrand in double-precision mode:

```
FUNCTION DFUN(ND,X)
DOUBLE PRECISION DFUN,X(ND)
...
DFUN = ...
...
RETURN
END
```

`ND` (`INTEGER`) Number of integration variables.

`X` (`DOUBLE PRECISION`) Array containing the coordinates of the point in the integration volume at which `DFUN` is to be evaluated.

See **Long Write-up** for details.

References:

1. J.H. Friedman and M.H. Wright, A Nested Partitioning Procedure for Numerical Multiple Integration. ACM Trans. Math. Software **7** (1981) 76–92.

•

Author(s) : G.A. Erskine **Library:** MATHLIB
Submitter : **Submitted:** 01.09.1983
Language : Fortran **Revised:** 01.03.1994

First-order Differential Equations (Runge–Kutta)

Subroutine subprograms RRKSTP and DRKSTP advance the solution of the system of $n \geq 1$ simultaneous first-order differential equations

$$\frac{dy_i}{dx} = f_i(x, y_1, \ldots, y_n), \qquad (i = 1, 2, \ldots, n)$$

by a single step of length h in the independent variable x.
On CDC and Cray computers, the double-precision version DRKSTP is not available.

Structure:

SUBROUTINE subprograms
User Entry Names : RKSTP, DRKSTP
Obsolete User Entry Names : RKSTP $\equiv$ RRKSTP
Files Referenced : Unit 6
External References: user-supplied SUBROUTINE subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tRKSTP(N,H,X,Y,SUB,W)
```

N (INTEGER) Number n of equations.

H (type according to t) The step-length h.

X (type according to t) On entry, X must be equal to the initial value of the independent variable x. On exit, X is equal to $x + h$.

Y (type according to t) One-dimensional array of length $\geq$ N. On entry, Y(i), (i = 1, ... , N), must contain $y_i(x)$. On exit, Y(i), (i = 1, ... , N), contains approximate values $y_i(x + h)$.

SUB Name of a user-supplied SUBROUTINE subprogram, declared EXTERNAL in the calling program.

W (type according to t) Array containing at least 3*N elements required as working-space.

The user-supplied subroutine SUB should be of the form

```
SUBROUTINE SUB(X,Y,F)
```

where the variable X and the one-dimensional arrays Y(*) and F(*) are of type t. This subroutine must set

$$\mathtt{F(I)} = f_{\mathtt{I}}(\mathtt{X}, \mathtt{Y(1)}, \ldots, \mathtt{Y(N)}) \qquad (\mathtt{I} = 1, 2, \ldots, \mathtt{N}).$$

Method:

Using boldface quantities to denote vectors of length n, the computational sequence is as follows:

$$\begin{aligned}
\mathbf{k}_1 &= h\mathbf{f}(x, \mathbf{y}(x)),\\
\mathbf{k}_2 &= h\mathbf{f}(x + \tfrac{1}{2}h, \mathbf{y}(x) + \tfrac{1}{2}\mathbf{k}_1),\\
\mathbf{k}_3 &= h\mathbf{f}(x + \tfrac{1}{2}h, \mathbf{y}(x) + \tfrac{1}{2}\mathbf{k}_2),\\
\mathbf{k}_4 &= h\mathbf{f}(x + h, \mathbf{y}(x) + \mathbf{k}_3); \qquad \mathbf{y}(x+h) \;=\; \mathbf{y}(x) + \tfrac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)
\end{aligned}$$

The error per step is proportional to h^5.

Error handling:

`N` $<$ `1` acts as do nothing.

References:

1. F.B. Hildebrand, Introduction to numerical analysis, (McGraw-Hill, New–York 1956) Sect. 6.16.

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 15.02.1989
Revised: 01.12.1994

First-order Differential Equations (Gragg–Bulirsch–Stoer)

Subroutine subprograms RDEQBS and DDEQBS advance the solution of the system of $n \geq 1$ simultaneous first-order differential equations

$$\frac{dy_i}{dx} = f_i(x, y_1, \ldots, y_n), \qquad (i = 1, 2, \ldots, n),$$

from a specified value x_1 to a specified value x_2 of the independent variable x.

On computers other than CDC and Cray, only the double-precision version DDEQBS is available. On CDC and Cray computers, only the single-precision version RDEQBS is available.

Structure:

SUBROUTINE subprograms
User Entry Names : RDEQBS, DDEQBS
Obsolete User Entry Names: DEQBS $\equiv$ RDEQBS
Files Referenced : Unit 6
External References: MTLMTR (N002), ABEND (Z035), user-supplied SUBROUTINE subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tDEQBS(N,X1,X2,Y,H0,EPS,SUB,W)
```

N (INTEGER) Number n of equations.

X1 (type according to t) Initial value x_1 of the independent variable x.

X2 (type according to t) Final value x_2 of the independent variable x.

Y (type according to t) One-dimensional array of length $\geq$ N. On entry, Y(i), (i = 1, ... , N), must contain $y_i(x_1)$. On exit, Y(i), (i = 1, ... , N), contains approximate values $y_i(x_2)$.

H0 (type according to t) On entry, H0 must contain the proposed initial step-length h_0. On exit, H0 contains the last computed step-length (See also **Method** and **Notes**).

EPS (type according to t) The requested absolute accuracy ε. (EPS should not be smaller than approximately 10^3 times the machine precision).

SUB Name of a user-supplied SUBROUTINE subprogram, declared EXTERNAL in the calling program.

W (type according to t) Array containing at least 36*N elements required as working-space.

The user-supplied subroutine SUB should be of the form

```
SUBROUTINE SUB(X,Y,F)
```

where the variable X and the one-dimensional arrays Y(*) and F(*) are of type t. This subroutine must set

$$\mathtt{F(I)} = f_{\mathtt{I}}(\mathtt{X}, \mathtt{Y(1)}, \ldots, \mathtt{Y(N)}) \qquad (\mathtt{I} = 1, 2, \ldots, \mathtt{N}).$$

Method:

For the first integration step, starting at $x = x_1$, the step-length h is chosen to be the smallest of the numbers $h_0,\ h_0/2,\ h_0/4, \ldots$ for which not more than 9 stages of internal extrapolation yield an estimated error less than ε. This procedure is repeated until $x = x_2$ is reached. (For details, see Ref. 1).

Error handling:

Error `D201.1`: If the requestec accuracy cannot obtained, a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.
For `N` $<$ `1`, or `X1` $=$ `X2` or `H0` $=$ `0`, control is returned to the calling program without any change in `Y`.

Notes:

For well-conditioned systems of equations any reasonable value of the initial step length h_0 may be chosen. For ill-conditioned systems, the initial value of h_0 may be important, and tests with different values are advised. An inappropriate choice may lead to wrong results in such cases.

Source:

This subroutines is based on an Algol60 procedure given in Ref. 1. The adaption for integration over a given interval (not only over one step) is due to G. Janin.

References:

1. R. Bulirsch and J. Stoer, Numerical treatment of ordinary differential equations by extrapolation methods, Numer. Math. **8** (1966) 1–13.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.02.1989
Language : Fortran **Revised:** 01.12.1994

First-order Differential Equations (Runge–Kutta–Merson)

Subroutine subprograms RDEQMR and DDEQMR advance the solution of the system of $n \geq 1$ simultaneous first-order differential equations

$$\frac{dy_i}{dx} = f_i(x, y_1, \dots, y_n), \qquad (i = 1, 2, \dots, n)$$

from a specified value x_1 to a specified value x_2 of the independent variable x. The integration step-length is automatically adjusted to keep the estimated error per step less than a specified value.

On computers other than CDC and Cray, only the double-precision version DDEQBS is available. On CDC and Cray computers, only the single-precision version RDEQBS is available.

Structure:

SUBROUTINE subprograms
User Entry Names : RDEQMR, DDEQMR
Obsolete User Entry Names: DEQMR $\equiv$ RDEQMR
Files Referenced : Unit 6
External References: MTLMTR (N002), ABEND (Z035), User-supplied SUBROUTINE subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tDEQMR(N,X1,X2,Y,H0,EPS,SUB,W)
```

N (INTEGER) Number n of equations.

X1 (type according to t) Initial value x_1 of the independent variable x.

X2 (type according to t) Final value x_2 of the independent variable x.

Y (type according to t) One-dimensional array of length $\geq$ N. On entry, Y(i), (i = 1, ..., N), must contain $y_i(x_1)$. On exit, Y(i), (i = 1, ..., N), contains approximate values $y_i(x_2)$.

H0 (type according to t) On entry, H0 must contain the proposed initial step-length h_0. On exit, H0 contains the last computed step-length (See also **Method** and **Notes**).

EPS (type according to t) The requested absolute accuracy ε. (EPS should not be smaller than approximately 10^3 times the machine precision).

SUB Name of a user-supplied SUBROUTINE subprogram, declared EXTERNAL in the calling program.

W (type according to t) Array containing at least 6*N elements required as working-space.

The user-supplied subroutine SUB should be of the form

```
SUBROUTINE SUB(X,Y,F)
```

where the variable X and the one-dimensional arrays Y(*) and F(*) are of type t. This subroutine must set

$$\mathtt{F(I)} = f_{\mathtt{I}}(\mathtt{X}, \mathtt{Y(1)}, \dots, \mathtt{Y(N)}) \qquad (\mathtt{I} = 1, 2, \dots, \mathtt{N}).$$

Method:

The method is a modification by Merson of the Runge–Kutta method. The initial value of the step-length h is taken to be the first of the numbers $h_0, h_0/2, h_0/4, \ldots$ for which the estimated relative error at the end of the step is less than ε. At each susequent step, an estimate of the integration error for that step (proportional to h^5) is computed. If the corresponding relative error exceeds ε, the current step-length is halfed; if it is less than $\varepsilon/32$ the step-length is doubled. This process is continued until x_2 is reached. (For details, see Ref. 1).

Error handling:

Error `D202.1`: If the requestec accuracy cannot obtained, a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.
For `N` $<$ `1`, or `X1` $=$ `X2` or `H0` $=$ `0`, control is returned to the calling program without any change in `Y`.

Notes:

For well-conditioned systems of equations any reasonable value of the initial step length h_0 may be chosen. For ill-conditioned systems, the initial value of h_0 may be important, and tests with different values are advised. An inappropriate choice may lead to wrong results in such cases.

References:

1. G.N. Lance, Numerical methods for high-speed computers, (Iliffe & Sons, London 1960) 56

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:** 01.12.1994

Second-order Differential Equations (Runge–Kutta–Nyström)

Subroutine subprograms RRKNYS and DRKNYS advance the solution of the system of $n \geq 1$ simultaneous second-order differential equations

$$\frac{d^2y_i}{dx^2} = f_i(x, y_1, \ldots, y_n, y'_1, \ldots, y'_n), \qquad (i = 1, 2, \ldots, n)$$

where $y'_i = dy_i/dx$, by a single step of length h in the independent variable x.

On computers other than CDC or Cray, only the double-precision version DRKNYS is available. On CDC and Cray computers, only the single-precision version RRKNYS is available.

Structure:

SUBROUTINE subprograms
User Entry Names : RRKNYS, DRKNYS
Obsolete User Entry Names: RKNYS $\equiv$ RRKNYS
External References: User-supplied SUBROUTINE subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tRKNYS(N,H,X,Y,YP,SUB,W)
```

N (INTEGER) Number n of equations.

H (type according to t) The step-length h.

X On entry, X must be equal to the initial value of the independent variable x. On exit, X is equal to $x + h$.

Y (type according to t) One-dimensional array of length $\geq$ N. On entry, Y(i), (i = 1, ..., N), must contain $y_i(x)$. On exit, Y(i), (i = 1, ..., N), contains approximate values $y_i(x + h)$.

YP (type according to t) One-dimensional array of length $\geq$ N. On entry, YP(i), (i = 1, ..., N), must contain $y'_i(x)$. On exit, YP(i), (i = 1, ..., N), contains approximate values $y'_i(x + h)$.

SUB Name of a user-supplied SUBROUTINE subprogram, declared EXTERNAL in the calling program.

W (type according to t) Array containing at least 6*N elements required as working-space.

The user-supplied subroutine SUB should be of the form

```
SUBROUTINE SUB(X,Y,YP,F)
```

where the variable X and the one-dimensional arrays Y(*), YP(*) and F(*) are of type t. This subroutine must set

$$\mathtt{F(I)} = f_{\mathtt{I}}(\mathtt{X}, \mathtt{Y(1)}, \ldots, \mathtt{Y(N)}, \mathtt{YP(1)}, \ldots, \mathtt{YP(N)}) \qquad (\mathtt{I} = 1, 2, \ldots, \mathtt{N}).$$

Method:

Using boldface quantities to denote vectors of length n, the computational sequence is as follows:

$$\begin{aligned}
\mathbf{k}_1 &= \frac{1}{2}h^2\,\mathbf{f}(x, \mathbf{y}(x), \mathbf{y}'(x)) \\
\mathbf{k}_2 &= \frac{1}{2}h^2\,\mathbf{f}(x + \frac{1}{2}h, \mathbf{y}(x) + \frac{1}{2}h\mathbf{y}'(x) + \frac{1}{4}\mathbf{k}_1, \mathbf{y}'(x) + \frac{1}{h}\mathbf{k}_1) \\
\mathbf{k}_3 &= \frac{1}{2}h^2\,\mathbf{f}(x + \frac{1}{2}h, \mathbf{y}(x) + \frac{1}{2}h\mathbf{y}'(x) + \frac{1}{4}\mathbf{k}_1, \mathbf{y}'(x) + \frac{1}{h}\mathbf{k}_2) \\
\mathbf{k}_4 &= \frac{1}{2}h^2\,\mathbf{f}(x + h, \mathbf{y}(x) + h\mathbf{y}'(x) + \mathbf{k}_3, \mathbf{y}'(x) + \frac{2}{h}\mathbf{k}_3) \\
\mathbf{y}(x+h) &= \mathbf{y}(x) + h\mathbf{y}'(x) + \frac{1}{3}(\mathbf{k}_1 + \mathbf{k}_2 + \mathbf{k}_3) \\
\mathbf{y}'(x+h) &= \mathbf{y}'(x) + \frac{1}{3h}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)
\end{aligned}$$

The error per step is proportional to h^5.

Error handling:

For `N` $\leq$ `0` or `H` $=$ `0`, control is returned to the calling program without any change in `Y` or `YP`.

References:

1. L. Collatz, The numerical treatment of differential equations, (Springer-Verlag Berlin 1960) 537

•

Author(s) : J. Hornsby **Library:** MATHLIB
Submitter : R. Keyser **Submitted:** 02.05.1966
Language : Fortran **Revised:** 30.01.1980

Elliptic Partial Differential Equation

EPDE1 solves an elliptic partial differential equation of general form (Poisson's equation being a special case) over a two-dimensional region using a finite difference method. The region may be of any shape and on its boundary either the dependent variable or a relation involving its derivative may be specified.

Structure:

SUBROUTINE subprograms
User Entry Names: EPDE1
Files Referenced: Reader, Printer, TAPE4, TAPE5
External References: User-supplied SUBROUTINES

Usage:

See **Long Write-up**.

•

Author(s) : R.C. Le Bail **Library:** MATHLIB
Submitter : **Submitted:** 20.03.1972
Language : Fortran **Revised:** 01.12.1981

Fast Partial Differential Equation Solver

ELPAHY uses fast Fourier transform techniques for the solution, over a rectangular domain, of the following elliptic, parabolic or hyperbolic part differential equation:

$$\frac{d^2\phi(x,y)}{dx^2} + c_1\frac{d^2\phi(x,y)}{dy^2} + c_2\frac{d\phi(x,y)}{dy} + c_3\phi(x,y) \;=\; \rho(x,y)$$

where $\phi(x,y)$ is the unknown function, $\rho(x,y)$ the known source term, and c_1, c_2, c_3 given coefficients. A large variety of boundary conditions can be specified on the sides of the rectangle.

Structure:

SUBROUTINE subprogram
User Entry Names: ELPAHY
Internal Entry Names: NEWRO, ELANAL, ESOLVE, SYNT, MFT
External References: RFT (D700)
COMMON Block Names and Lengths: /FW1/ 774, /FW2/ 100

Usage:

```
CALL ELPAHY(F,NX,NY,DX,DY,C,IBX,BWEST,BEAST,JBY,BSOUTH,BNORTH)
```

F (REAL) Two-dimensional array, dimensioned (NX,NY) in the calling program. On input it contains the source term $\rho(x,y)$ and on return it contains the unknown function $\phi(x,y)$.

NX (INTEGER) Number of divisions along X. NX must be of the form 2^n+1.

NY (INTEGER) Number of divisions along Y.

DX (REAL) Mesh spacing along X.

DY (REAL) Mesh spacing along Y.

C (REAL) One-dimensional array of dimension 3, containing the coefficients c_1, c_2, c_3.

IBX (INTEGER) Controls the type of boundary conditions on the left (BWEST) and right (BEAST) sides of the rectangular domain:
IBX $=1$: Imposed periodicity along x; BWEST, BEAST not given.
IBX $=2$: Given derivative on either vertical side.
IBX $=3$: Given value on either vertical side.
IBX $=4$: Given value on the left side, given derivative on the right side.

BWEST (REAL) One-dimensional array of size NY containing values or derivatives for the left side; the interpretation depends on IBX.

BEAST (REAL) One-dimensional array of size NY containing values or derivatives for the right side; the interpretation depends on IBX.

JBY (INTEGER) Controls the type of boundary conditions on the lower (BSOUTH) und upper (BNORTH) sides of the rectangular domain:

Elliptic equation ($c_1 > 0$):
JBY = 1 : Given value on both lower and upper sides.
JBY = 2 : Given derivative on both lower and upper sides.
JBY = 3 : Given value on lower side, given derivative on upper side.
JBY = 4 : Given derivative on lower side, given value on upper side.

Parabolic equation ($c_1 = 0$):
Specify BSOUTH array only. (If y=time, BSOUTH are initial values and the future BNORTH cannot be specified).
JBY = 1 : Given value on lower side.
JBY = 2 : Given derivative on lower side.

Hyperbolic equation ($c_1 < 0$):
The BSOUTH array specifies the value, the BNORTH array the derivative.
JBY = 1.

BSOUTH (REAL) One-dimensional array of size NX containing values or derivatives for the lower side; the interpretation depends on JBY.

BNORTH (REAL) One-dimensional array of size NX containing values or derivatives for the upper side; the interpretation depends on JBY.

Notes:

If NX > 65, specify COMMON /FWORK/ of length 6*NX and COMMON /FW1/ of length 6*NX in the calling program. If NY > 50, specify COMMON /FW2/ of length 2*NY. In either case, make sure your program is loaded before ELPAHY (D302) (this is automatic unless you recompile D302 in the same job).

References:

1. R.C. Le Bail, Use of fast Fourier transforms for solving partial differential equations in physics, J. Comput. Phys. **9** (1972) 440–465

A copy of Ref. 1 is available.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.02.1989
Language : Fortran **Revised:** 01.12.1994

Numerical Differentiation

Subroutine subprograms RDERIV and DDERIV compute an approximate numerical value of the derivative $f'(x)$ of a function $f(x)$ at a specified point x.

On computers other than CDC and Cray, only the double-precision version DDERIV is available. On CDC and Cray computers, only the single-precision version RDERIV is available.

Structure:

SUBROUTINE subprograms
User Entry Names : RDERIV, DDERIV
Obsolete User Entry Names: DERIV $\equiv$ RDERIV
Files Referenced : Unit 6
External References: MTLMTR (N002), ABEND (Z035), user-supplied FUNCTION subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tDERIV(F,X,DELTA,DFDX,RERR)
```

F (type according to t) Name of a user-supplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must set F(X) $= f($X$)$.

X (type according to t) The specified point x for which the derivative is to be calculated.

DELTA (type according to t) On entry, DELTA must contain a scaling factor δ, which can usually be set equal to 1. On exit, it contains the last value of this factor (see Method).

DFDX (type according to t) On exit, DFDX contains an approximation to $f'($X$)$.

RERR (type according to t) On exit, RERR contains an estimate of the relative error of DFDX.

Method:

The method is based on an extension to numerical differentiation of Romberg's principle of sequence extrapolation, originally developed for numerical integration. The subroutine starts by computing the 10 numbers

$$T_0^{(k)} = [f(x+h) - f(x-h)]/(2h), \quad (k = 0, 1, \ldots, 9),$$

with

$$h = \delta * 0.0256 * 2^{-k/2} \qquad (k \text{ even})$$
$$h = \delta * 0.0192 * 2^{-(k-1)/2} \qquad (k \text{ odd}),$$

where the scaling factor δ is initially set to DELTA. This procedure is repeated up to 9 times, with δ replaced by $\delta/10$ each time, until the sequence $T_0^{(k)}$ is found to be monotone. A Romberg-like triangular table

$$T_m^{(k)} = u_m T_{m-1}^{(k+1)} - w_m T_{m-1}^{(k)}.$$

with appropriate weights u_m, w_m is then computed for $m = 1, 2, \ldots, 9$; $k = 0, 1, \ldots, 9-m$, and DFDX is set equal to $T_9^{(0)}$.

Restrictions:

The function $f(x)$ must be differentiable at $x =$ X and in a neighbourhood of X. Misleading results will be obtained if this is not true.

Error handling:

Error D401.1: If the function $f(x)$ is such that, after 9 successive reductions of δ by a factor 1/10, the sequence $T_0^{(k)}$ is not monotone, an error message is written Unit 6, unless subroutine MTLSET (N002) has been called. DFDX is set equal to zero, RERR is set equal to one in this case.

References:

1. H. Rutishauser, Ausdehnung des Rombergschen Prinzips, Numer. Math. **5** (1963) 48–54.

•

Author(s) : W. Mönch, B. Schorr **Library:** MATHLIB
Submitter : W. Mönch **Submitted:** 15.03.1993
Language : Fortran **Revised:**

Constrained Non-Linear Least Squares and Maximum Likelihood Estimation

LEAMAX is a portable collection of subprograms for solving general non-linear least squares problems, non-linear data fitting problems, and maximum likelihood estimations.

Subroutine subprograms RSUMSQ, RFUNFT, RMAXLK and DSUMSQ, DFUNFT, DMAXLK calculate an approximation to a minimum of an objective function φ , with respect to n unknown parameters $a = (a_1, ..., a_n)^{\mathbf{T}} \in \mathbf{R^n}$:

(S) The general non-linear least squares problem

$$\varphi_S(a) \;=\; \frac{1}{2}\sum_{i=1}^{m}[f_i(a)]^2,$$

(F) The least squares data fitting problem

$$\varphi_F(a) \;=\; \frac{1}{2}\sum_{i=1}^{m}\left[\frac{y_i - f(x_i, a)}{\sigma_i}\right]^2,$$

(M) The maximum likelihood estimation

$$\varphi_M(a) \;=\; -\sum_{i=1}^{m}\ln(f(x_i, a)),$$

subject to bounds on the variables of the form

$$\underline{a}_j \le a_j \le \overline{a}_j \qquad (j = 1, 2, \ldots, n).$$

The functions $f_i : \mathbf{R^n} \to \mathbf{R^1}$ $(i = 1, ..., m)$ and $f : \mathbf{R^k} \times \mathbf{R^n} \to \mathbf{R^1}$ are arbitrary non-linear functions with respect to the argument a. In the case of the data fitting problem **(F)**, a set of observation data $\{(x_i, y_i)|x_i \in \mathbf{R^k}, y_i \in \mathbf{R^1}, i = 1, ..., m\}$ with their corresponding weights σ_i $(i = 1, ..., m)$ has to be provided, whereas for the maximum likelihood estimation **(M)**, the set of data points $\{(x_i)|x_i \in \mathbf{R^k}, i = 1, ..., m\}$ belongs to the input of the problem.

These subprograms are based on the algorithm described by Moré (Ref. 1) for finding the solution of a general non-linear least squares problem by using the Levenberg-Marquardt algorithm. The method is completed by an active set strategy for handling simple box constraints to the unknown parameters (see **Long Write-up** for details). The necessary derivatives can either be supplied by the user (subprogram SUB) or are approximated numerically. In the case of a non-linear data fitting problem, approximations to the covariance matrix and the standard deviations of the model parameter estimators are also provided.

On computers other than CDC or Cray, only the double-precision versions DSUMSQ, DFUNFT, DMAXLK are available. On CDC and Cray computers, only the single-precision versions RSUMSQ, RFUNFT, RMAXLK are available.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RSUMSQ`, `RFUNFT`, `RMAXLK`, `DSUMSQ`, `DFUNFT`, `DMAXLK`
Internal Entry Names: `D501L1`, `D501L2`, `D501SF`, `D501P1`, `D501P2`, `D501N1`, `D501N2`

External References:	`RGEQPF` (F001),	`RORMQR` (F001),	`RTRTRS` (F001),	`DGEQPF` (F001),
	`DORMQR` (F001),	`DTRTRS` (F001),	`RVSET` (F002),	`RVSCL` (F002),
	`RVSUB` (F002),	`RVCPY` (F002),	`RVMPY` (F002),	`DVSET` (F002),
	`DVSCL` (F002),	`DVSUB` (F002),	`DVCPY` (F002),	`DVMPY` (F002),
	`RMSET` (F003),	`RMSCL` (F003),	`RMCPY` (F003),	`RMMPY` (F003),
	`RMBIL` (F003),	`DMMLT` (F003),	`DMSET` (F003),	`DMSCL` (F003),
	`DMCPY` (F003),	`DMMPY` (F003),	`DMBIL` (F003),	
	`RMMLT` (F004),	`DMMLT` (F004),	`RSINV` (F012),	`DSINV` (F012)
	User-supplied `SUBROUTINE` subprogram			

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`):

(S) General non-linear least squares problem

```
      CALL tSUMSQ(SUB,M,N,NC,A,AL,AU,MODE,EPS,MAXIT,IPRT,
     +            MFR,IAFR,PHI,DPHI,COV,STD,W,NERROR)
```

(F) Least squares data fitting problem

```
      CALL tFUNFT(SUB,K,M,N,NX,NC,X,Y,SY,A,AL,AU,MODE,EPS,MAXIT,IPRT,
     +            MFR,IAFR,PHI,DPHI,COV,STD,W,NERROR)
```

(M) Maximum likelihood estimation

```
      CALL tMAXLK(SUB,K,M,N,NX,X,A,AL,AU,MODE,EPS,MAXIT,IPRT,
     +            MFR,IAFR,PHI,DPHI,W,NERROR)
```

`SUB` Name of user-supplied `SUBROUTINE` subprogram, declared `EXTERNAL` in the calling program. This subprogram must provide the values of the functions $f_i(a)$ $(i = 1, \ldots, m)$, $f(\cdot, a)$, and, if `MODE` $= 1$, the values of the derivatives $\partial f_i(a)/\partial a_j$ and $\partial f(\cdot, a)/\partial a_j$ $(i = 1, \ldots, m; j = 1, \ldots, n)$ (see **Examples**).

`K` (`INTEGER`) Cases **(F)** and **(M)**: dimension k of a data point (observation) $x_i \in \mathbf{R^k}$.

`M` (`INTEGER`) Case **(S)**: number of non-linear functions f_i; cases **(F)** and **(M)**: number of data points (observations).

`N` (`INTEGER`) Number of unknown parameters a.

`NX` (`INTEGER`) Cases **(F)** and **(M)**: declared first dimension of array `X` in the calling program, `NX` $\geq$ `K`.

`NC` (`INTEGER`) Cases **(S)** and **(F)**: declared first dimension of array `COV` in the calling program, `NC` $\geq$ `N`.

`X` (Type according to `t`) Cases **(F)** and **(M)**: two-dimensional array of dimension (`NX`, $\geq$ `M`). On entry, `X` must contain the data set $\{ x_i \}$ (the i-th column of `X` belongs to the data point $x_i \in \mathbf{R^k}$, $i = 1, \ldots, m$).

`Y` (type according to `t`) Case **(F)**: one-dimensional array of length $\geq$ `M`, contains, on entry, the data set $\{ y_i \}$.

`SY` (type according to `t`) Case **(F)**: one-dimensional array of length $\geq$ `M`, contains, on entry, the weights $\{ \sigma_i \}$ of the data points.

A (Type according to t) One-dimensional array of length $\geq$ N. On entry, A(J) must contain the starting value of a_j for the Levenberg-Marquardt algorithm. On exit, A(J) contains an approximation to a_j of a minimum point (if the algorithm was successful).

AL (Type according to t) One-dimensional array of length $\geq$ N. On entry, AL(J) must contain the lower bound $\underline{a}_j$ of a_j .

AU (Type according to t) One-dimensional array of length $\geq$ N. On entry, AU(J) must contain the upper bound $\overline{a}_j$ of a_j .

MODE (INTEGER)

= 0 : The derivatives are approximated by divided differences.

= 1 : The derivatives are to be supplied by subprogram SUB.

Other values for MODE are treated as MODE = 0.

EPS (Type according to t) User-supplied tolerance used to control the termination criterion. EPS should be chosen according to the accuracy required by the problem and the machine accuracy t (recommended value on entry: between 10^{-6} for t = R , and 10^{-12} for t = D, respectively).

MAXIT (INTEGER) Maximum permitted number of iterations.

IPRT (INTEGER) Printing control.

= 0 : no printing of intermediate results,

= ±L : printing of intermediate results at every |L|-th iteration; if IPRT < 0, printing of all input parameters of tSUMSQ, tFUNFT, tMAXLK, respectively, in addition.

MFR (INTEGER) On exit, MFR contains the number of free variables at the solution point.

IAFR (INTEGER) One-dimensional array of length $\geq$ 2 * N for cases **(S)** and **(F)**, and of length $\geq$ N for case **(M)**, used as working space. On exit, the first MFR elements of IAFR contain the indices of the free variables at the solution point.

PHI (Type according to t) On exit, PHI contains the value of the objective function φ at the solution point.

DPHI (Type according to t) One-dimensional array of length $\geq$ N. On exit, DPHI(J) contains the derivative $\partial\varphi/\partial a_j$ of φ with respect to a_j (j-th component of the gradient of φ) at the solution point.

COV (Type according to t) Cases **(S)** and **(F)**: two-dimensional array of dimension (NC, $\geq$ N). On exit, COV contains an approximation to the covariance matrix.

STD (Type according to t) Cases **(S)** and **(F)**: one-dimensional array of length $\geq$ N. On exit, STD(J) contains an approximation to the standard deviation of the estimator of the model parameter a_j.

W (Type according to t) One-dimensional array of length $\geq$ 9 * N + 4 * M + 2 * M * N + 3 * N * N for cases **(S)** and **(F)**, and of length $\geq$ 7 * N + 2 * N * N for case **(M)**, used as working space.

NERROR (INTEGER) Error indicator. On exit:

= 0 : No error or warning detected.

= 1 : At least one of the constants K, M, N, NX, NC, MAXIT is illegal or at least for one j the relation $\underline{a}_j \leq \overline{a}_j$ is not true.

= 2 : The maximum number MAXIT of iterations has been reached.

= 3 : The objective function φ or its derivative is not defined for the current values of the parameter vector a.

= 4 : Cases **(S)** and **(F)**: The routines tGEQPF, tORMQR, tTRTRS in the Linear Algebra package LAPACK (F001) were unable to solve the linear least squares problem or the subprogram tSINV (F012) was unable to compute the covariance matrix.
Case **(M)**: the routine tSINV (F012) was unable to solve the normal equations.

Examples:

For the user-supplied `SUBROUTINE` subprogram `SUB` write for instance in the case `t` = `D`:

(S) Objective function (Brown badly-scaled function, $n = 2,\ m = 3$):

$$\varphi_S(a) \;=\; \frac{1}{2}\,\left[(a_1 - 10^6)^2 + (a_2 - 2\cdot 10^{-6})^2 + (a_1 a_2 - 2)^2\right]$$

.

```
      SUBROUTINE SUB(N,A,M,F,DF,MODE,NERROR)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (ZO = 0)
      DIMENSION A(*),F(*),DF(M,*)
      NERROR=0
      F(1)=A(1)-1D6
      F(2)=A(2)-2D-6
      F(3)=A(1)*A(2)-2
      IF(MODE .NE. 1) RETURN
      CALL DMSET(M,N,ZO,DF(1,1),DF(1,2),DF(2,1))
      DF(1,1)=1
      DF(2,2)=1
      DF(3,1)=A(2)
      DF(3,2)=A(1)
      RETURN
      END
```

(F) Objective function (Bard function, $n = 3,\ m = 15,\ k = 3$):

$$\varphi_F(a) \;=\; \frac{1}{2}\sum_{i=1}^{m}\left[y_i - \left(a_1 + \frac{x_{1,i}}{x_{2,i}\,a_2 + x_{3,i}\,a_3}\right)\right]^2$$

.

```
      SUBROUTINE SUB(K,X,N,A,F,DF,MODE,NERROR)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(*),X(*),DF(*)
      T=X(2)*A(2)+X(3)*A(3)
      IF (T .EQ. 0) THEN
       NERROR=3
      ELSE
       NERROR=0
       F=A(1)+X(1)/T
       IF(MODE .NE. 1) RETURN
       DF(1)=1
       DF(2)=-X(1)*X(2)/T**2
       DF(3)=-X(1)*X(3)/T**2
      ENDIF
      RETURN
      END
```

(M) Objective function ($n = 1,\ m = 100,\ k = 1$):

$$\varphi_M(a) \;=\; -\sum_{i=1}^{m} \ln \left\{ \frac{1}{a_1\sqrt{\pi}} \exp\left[-\frac{1}{2}\left(\frac{x_{1,i}-1}{a_1}\right)^2 \right] \right\}$$

.

```
SUBROUTINE SUB(K,X,N,A,F,DF,MODE,NERROR)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION A(*),X(*),DF(*)
PARAMETER (PIR = 0.56418 95835 47756 287D0)
NERROR=3
IF(A(1) .LE. 0) RETURN
T=0.5D0*((X(1)-1)/A(1))**2
F=PIR*EXP(-T)/A(1)
NERROR=0
IF(MODE .EQ. 1) DF(1)=-F*(1-2*T)/A(1)**2
RETURN
END
```

In all three cases the parameters `K` , `N` , `A` , `M` , `MODE` , `NERROR` are as declared above. The other parameters are the following:

`F` (Type according to `t`) Case **(S)**: one-dimensional array of length $\geq$ `M`. `F(I)` must contain the function value $f_i(a)$ at a $(i = 1, \ldots, m)$, on exit.
Cases **(F)** and **(M)**: `F` must contain the function value $f(x, a)$ at (x, a), on exit.

`DF` (Type according to `t`) If `MODE = 1` values of `DF` are supplied by `SUB`. For other values of `MODE` the parameter `DF` is not referenced.
Case **(S)**: two-dimensional array of dimension (`M`, $\geq$ `N`). `DF(I,J)` must contain the value of the partial derivative $\partial f_i(a)/\partial a_j$ at a, $(i = 1, \ldots, m; j = 1, \ldots, n)$, on exit.
Cases **(F)** and **(M)**: one-dimensional array of length $\geq$ `N`. `DF(J)` must contain the value of the partial derivative $\partial f(x, a)/\partial a_j$, $(j = 1, \ldots, n)$, on exit.

`X` (Type according to `t`) Cases **(F)** and **(M)**: one-dimensional array of length $\geq$ `K` for **one** data point $x_i \in \mathbf{R^k}$ (in contrast to above declaration).

References:

1. J.J. Moré, The Levenberg-Marquardt algorithm: Implementation and theory, In: Numerical Analysis, G.A. Watson (Ed.), Lecture Notes in Mathematics 630, Springer-Verlag, New York (1977) 105-116.

2. Å.Björck: Solution of Equations in $\mathbf{R^n}$ (Part 1: Least Squares Methods). In: Handbook of Numerical Analysis, P.G.Ciarlet, J.L.Lions (Eds.), North-Holland, Amsterdam, New York, Oxford, Tokyo, 1990, 467-636.

3. R.Fletcher: Practical Methods of Optimization. John Wiley and Sons, Chichester, 2nd Edition, 1987.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.11.1993
Language : Fortran **Revised:**

Minimum of a Function of One Variable

Subroutine subprograms `RMINFC` and `DMINFC` calculate, to a limited specified accuracy, the abscissa of a single local minimum of a real-valued function $f(x)$ lying in a given interval (a,b), together with the function value at the minimum. Although this subprogram may find a minimum under other conditions (see **Notes**), the search interval should contain exactly one local minimum point x with $a < x < b$.

On CDC and Cray computers, the double-precision version `DMINFC` is not available.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RMINFC`, `DMINFC`
External References: User-supplied `FUNCTION` subprogram

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
    CALL tMINFC(F,A,B,EPS,DELTA,X,Y,LLM)
```

`F` (type according to `t`) Name of a user-supplied `FUNCTION` subprogram, declared `EXTERNAL` in the calling program. This function must set `F(X)` $= f($`X`$)$.

`A,B` (type according to `t`) On entry, `A` and `B` must specify the end-points a, b of the search interval.

`EPS` (type according to `t`) On entry, `EPS` must be equal to the accuracy parameter ε (see **Accuracy**).

`DELTA` (type according to `t`) On entry, `DELTA` must be equal to the parameter δ specifying a tolerance interval near `A` and `B` (see **Accuracy**).

`X` (type according to `t`) On exit, `X` is the computed approximation to the abscissa of a minimum of the function $f(x)$.

`Y` (type according to `t`) Contains, on exit, the value of $f($`X`$)$.

`LLM` (`LOGICAL`) On exit, `LLM` is `.TRUE.` if the relations $|\mathtt{X} - \mathtt{A}| > \delta$ and $|\mathtt{X} - \mathtt{B}| > \delta$ are both true (i.e. if `X` is the abscissa of a local minimum lying inside the interval $[\mathtt{A},\mathtt{B}]$), and `.FALSE.` otherwise (see **Notes**).

Method:

The so-called *golden section search* is applied (see **References**). This method uses a fixed number n of function evaluations, where $n = [\,2.08 \times \ln{(|a-b|/\varepsilon)} + \frac{1}{2}\,] + 1$.

Accuracy:

The accuracy depends on the behaviour of the function and is difficult to measure. For example, a flat minimum results in poor accuracy. This implies that the subprograms are not intended to replace the usual procedures when a minimum of a function is needed in the exact mathematical sense. In any case, a choice of $\varepsilon > 10^{-8}$ in double-precision and of $\varepsilon > 10^{-4}$ in single-precision mode usually results in a relative error of `X` which is smaller than or in the order of ε. A suggested value of δ is $\delta = 10\varepsilon$.

Notes:

1. As a rule, the specified interval (a, b) should contain strictly one local minimum.
2. If this is not the case, and if $f(x)$ is monotonous in (a, b), the subprograms find the minimum at the correct endpoint a or b. `LLM` is set to `.FALSE.` in this case.
3. In all other possible cases, the behaviour of the subprograms is not easy to predict. In particular, in the case of several minimal points inside (a, b), one of them is found, but not necessarily the one with the smallest value of the function.

References:

1. R. Fletcher, Practical methods of optimization (John Wiley & Sons, Chichester 1987) 39–40.
2. W. Krabs, Einführung in die lineare und nichtlineare Optimierung für Ingenieure (BSB B.G. Teubner, Leipzig 1983) 84–86

•

MINUIT	CERN Program Library	D506

Author(s) : F. James, M. Roos **Library:** PACKLIB
Submitter : F. James **Submitted:** 03.05.1967
Language : Fortran **Revised:** 15.01.1994

Function Minimization and Error Analysis

The MINUIT package performs minimization and analysis of the shape of a multi-parameter function. It is intended to be used on Chisquare or likelihood functions for fitting data and finding parameter errors and correlations. The more important options offered are:

- Variable metric (Fletcher) minimization
- Monte Carlo minimization
- Simplex (Nelder and Mead) minimization
- Parabolic error analysis (error matrix)
- MINOS (non-linear) error analysis
- Contour plotting
- Fixing and restoring parameters
- Global minimization

Structure:

SUBROUTINE subprograms
User Entry Names: MINTIO, MINUIT, MNCOMD, MNCONT, MNERRS, MNEXCM, MNINPU, MNINTR, MNPARS, MNREAD
Internal Entry Names: MNAMIN, MNBINS, MNCALF, MNCLER, MNCNTR, MNCRCK, MNCROS, MNCUVE, MNDERI, MNDXDI, MNEIG, MNEMAT, MNEVAL, MNEXIN, MNFIXP, MNFREE, MNGRAD, MNHELP, MNHESS, MNHES1, MNIMPR, MNINEX, MNINIT, MNLIMS, MNLINE, MNMATU, MNMIGR, MNMNOS, MNMNOT, MNPARM, MNPFIT, MNPINT, MNPLOT, MNPOUT, MNPRIN, MNPSDF, MNRAZZ, MNRN15, MNRSET, MNSAVE, MNSCAN, MNSEEK, MNSET, MNSETI, MNSIMP, MNSTAT, MNSTIN, MNTINY, MNUNPT, MNWARN, MNWERR, MNVERT, STAND

Usage:

MINUIT can be used either

as a “master” batch program which reads and executes commands appearing in the input data stream;

or

as a “master” interactive program which reads and executes commands given from the terminal;

or

as a Fortran callable “slave” package, called from the user program or from an intermediate package such as PAW or HBOOK;

or

any combination of the above.

See **Long Write-up** for details.

•

FUMILI	CERN Program Library	D510

Author(s) : I. Silin
Submitter : A. Kobine
Language : Fortran

Library: MATHLIB
Submitted: 05.04.1971
Revised: 18.11.1985

Fitting Chisquare and Likelihood Functions

OBSOLETE

Please note that this routine has been obsoleted in CNL 211. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: `LEAMAX` (D501)

`FUMILI` minimizes the objective functions $\chi^2/2$ and ML defined by:

$$\frac{1}{2}\chi^2 = \frac{1}{2}\sum_{j=1}^{N}\left[\frac{Y_j^* - Y(X_j^{(1)},\ldots,X_j^{(L)};A_1,\ldots,A_M)}{\Delta Y_j^*}\right]^2$$

and

$$ML = \sum_{j=1}^{N} -\ln[Y(X_j^{(1)},\ldots,X_j^{(L)};A_1,\ldots,A_M)]$$

with respect to the M parameters A where, for each j, $1 \le j \le N, Y_j^*$ is a data-point with user estimated error, $\pm\Delta Y_j$, the X_j are L co-ordinates of that point and Y is a theoretical function predicting Y_j^* for a given set of X_j and A.

The method makes use of a particular property concerning the dependence of the objective function ($\chi^2/2$ or ML) on the theoretical function (Y) for faster convergence.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `FUMILI`, `LIKELM`, `ERRORF`
Internal Entry Names: `ARITHM`, `D510BD`, `FUNCT`, `MCONV`, `MONITO`, `SCAL`, `SGZ`
Files Referenced: Printer
External References: User-supplied `FUNCTION` and (optional) `SUBROUTINE` subprograms
`COMMON` Block Names and Lengths: `/A/ 100`, `/AL/ 100`, `/AU/ 100`, `/DA/ 100`, `/DF/ 100`, `/ENDFLG/ 7`, `/ERROR/ 500`, `/EXDA/ 1500`, `/G/ 100`, `/NED/ 2`, `/PL/ 100`, `/PLU/ 100`, `/R/ 100`, `/SIGMA/ 100`, `/X/ 10`, `/Z/ 2485`, `/Z0/ 2485`

Usage:

See **Long Write-up**.

References:

1. Preprint YINDR-810, 1961 (Dubna) (CERN Library, preprint P. 810).

•

Author(s) : G.A. Erskine and K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Solution of a Linear Fredholm Integral Equation of Second Kind

Subroutine subprograms `RFRDH1`, `DFRDH1` and function subprograms `RFRDH2`, `DFRDH2` calculate an approximation to the solution y of the Fredholm integral equation

$$y(x) \;=\; F(x) + \int_a^b G(x,t)\,y(t)\,dt \qquad (1)$$

over the interval $[a,b]$. The function F must not be identically zero. The interval $[a,b]$ may be divided into m subintervals $[t_{i-1},t_i]$, $(i=1,2,\ldots,m)$, with $a=t_0<t_1<\cdots<t_m=b$.
The order N_i (number of abscissae) of the Gaussian quadrature formula used for integrating over $[t_{i-1},t_i]$ is specified separately for each subinterval.

Function subprograms `RFRDH3` and `DFRDH3` evaluate numerically integrals of the form $\int_a^b H(t)\,y(t)\,dt$ where H is an arbitrary function and y is the solution of (1).
The following values of N_i may be used: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 32, 40, 48, 64, 80, 96.
On computers other than CDC and Cray, only the double-precision versions `DFRDH1` etc. are available. On CDC and Cray computers, only the single-precision versions `RFRDH1` etc. are available.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `RFRDH1`, `RFRDH2`, `RFRDH3`, `DFRDH1`, `DFRDH2`, `DFRDH3`
Files Referenced: `Unit 6`
External References: `RGSET` (D107), `DGSET` (D107), `REQN` (F010), `DEQN` (F010), `MTLMTR` (N002), `ABEND` (Z035), user-supplied `FUNCTION` subprograms.

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`), the first step in the solution of (1) must be the execution of a statement of the form:

```
CALL tFRDH1(F,G,M,T,NG,WS,IDIM,N)
```

`F,G` (type according to `t`) Names of user-supplied `FUNCTION` subprograms, declared `EXTERNAL` in the calling program. Subprogram `F` must set `F(X)` $= F($`X`$)$, subprogram `G` must set `G(X,T)` $= G($`X,T`$)$.

`M` (`INTEGER`) Number $m \ge 1$ of subintervals in $[a,b]$.

`T` (type according to `t`) One-dimensional array of dimension `(0:d)` where `d` $\ge$ `M`. On entry, `T` must contain the $m+1$ ordered points of subdivision t_i, $(i=0,1,\ldots,m)$, with $t_0=a$ and $t_m=b$.

`NG` (`INTEGER`) One-dimensional array of length $\ge$ `M`. On entry, `NG` must contain the order (number of absissae) N_i of the Gaussian quadrature formula to be used in the interval $t_{i-1}\le t\le t_i$, $(i=1,2,\ldots,m)$.

`WS` (type according to `t`) Two-dimensional array of dimensions (`IDIM`, $\ge$ `IDIM` $+4$). Used as working space and for communication between the subprograms.

`IDIM` (`INTEGER`) A number $\ge \sum_{i=1}^m N_i$.

`N` (`INTEGER`) On exit, `N` $= \sum_{i=1}^m N_i$.

Once tFRDH1 has been called, the function subprograms tFRDH2 and tFRDH3 may be referenced any number of times without any further call to tFRDH1.

In any arithmetic expression,

```
tFRDH2(F,G,X,WS,IDIM,N)
```

has the value $y(\texttt{X})$, where y is the approximate solution of (1).

In any arithmetic expression,

```
tFRDH3(H,WS,IDIM,N)
```

has the approximate value of $\int_a^b H(t)\, y(t)\, dt$ where y is the approximate solution of (1).

H (type according to t) is the name of a user-defined FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must set $\texttt{H(X)} = H(\texttt{X})$.

Method:

Let the sets $\{w_k\}$ and $\{z_k\}$ be defined by

$$\begin{aligned}\{w_k\} &= \{w_1^{(1)},\ldots,w_{N_1}^{(1)},\ldots,w_1^{(m)},\ldots,w_{N_m}^{(m)}\},\\ \{z_k\} &= \{z_1^{(1)},\ldots,z_{N_1}^{(1)},\ldots,z_1^{(m)},\ldots,z_{N_m}^{(m)}\}.\end{aligned}$$

$w_j^{(i)}$ and $z_j^{(i)}$ are respectively the weights and abscissae of the N_i-point Gaussian quadrature formulae corresponding to the interval $[t_{i-1}, t_i]$. Subprograms RFRDH1 or DFRDH1 sets up and solves the following system of simultaneous linear equations with unknowns $y(z_k)$:

$$y(z_k) = F(z_k) + \sum_{j=1}^{N} w_j G(z_k, z_j) y(z_k) \qquad (k = 1, 2, \ldots, N)$$

where $N = \sum_{i=1}^{m} N_i$.

Function subprogram tFRDH2 calculates $\quad y(\texttt{X}) = \sum_{k=1}^{N} w_k\, G(\texttt{X}, z_k)\, y(z_k).$

Function subprogram tFRDH3 calculates $\quad I = \sum_{k=1}^{N} w_k\, H(z_k)\, y(z_k).$

Accuracy:

The accuracy depends upon the extend to which the product $G(x,t)y(t)$ can be represented by a polynomial of degree $2N_i - 1$ for all x in the interval $[t_{i-1}, t_i]$, $(i = 1, 2, \ldots, m)$.

Error handling:

Error D601.1: In tFRDH1, the system of linear equations is singular. A message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

If any of the values N_i does not appear in the list given above, a message is written on Unit 6 by RGSET or DGSET (D107) unless subroutine MTLSET (N002) has been called.

•

Author(s) : C. Iselin
Submitter :
Language : Fortran
Library: MATHLIB
Submitted: 04.09.1972
Revised: 15.01.1977

Real Fast Fourier Transform

Let the discrete Fourier transform be defined by

$$y_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \exp\left(\frac{2\pi ijk}{N}\right) x_k, \qquad (j = 0, 1, \ldots, N).$$

The subroutines of package `RFT` compute this transform or its inverse

$$x_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \exp\left(\frac{-2\pi ijk}{N}\right) y_j, \qquad (k = 0, 1, \ldots, N)$$

for real functions, with the restriction that N is a power of 2.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RFT`, `RCA`, `RPA`, `RPS`, `RSA`
Internal Entry Names: `D700SU`
Files Referenced: Printer
`COMMMON` Block Names and Lengths: `/D700DT/ 6`, `/FWORK/ 321`

Usage:

```
CALL RFT(M,X,IX,Y,IY,MODE)
```
or
```
CALL RCA(M,X,IX,Y,IY)
```
or
```
CALL RPA(M,X,IX,Y,IY)
```
or
```
CALL RPS(M,X,IX,Y,IY)
```
or
```
CALL RSA(M,X,IX,Y,IY)
```

`M` (`INTEGER`) Number m (such that $n = 2^m$) of input values (full period or half period).

`X` (`REAL`) Input array. The input values are taken from $\texttt{X}(k{*}\texttt{IX}+1)$ for $k = 0, 1, \ldots, n$.

`Y` (`REAL`) Output array. The results are stored in $\texttt{Y}(k{*}\texttt{IY}+1)$ for $j = 0, 1, \ldots, n$.

`MODE` (`INTEGER`) Selects the mode of operation for `RFT` as follows:

`MODE = 1`: **Analysis of a general real function.**

```
CALL RFT(M,X,IX,Y,IY,1)
```
or
```
CALL RPA(M,X,IX,Y,IY)
```

assumes $x_k = \texttt{X}(k{*}\texttt{IX}+1)$ $(k = 0, 1, \ldots, n-1)$; $n = 2^m = N$ to define a full period of the function to be analysed. The value x_n is ignored. The results are returned in the following order:

$$y_0 = y_n = \texttt{Y}(1)$$
$$y_j = y_{n-j} = \texttt{Y}(j{*}\texttt{IY}+1) + i\texttt{Y}((j+n/2){*}\texttt{IY}+1), \quad (j = 1, 2, \ldots, n/2).$$

The other values in `Y` are not changed.

`MODE = 4`: **Synthesis of a general real function.**

```
CALL RFT(M,X,IX,Y,IY,4)      or
CALL RPS(M,X,IX,Y,IY)
```

is exactly the inverse of `MODE=1` as described above. The value x_n is set equal to x_0.

`MODE=2/5`: **Analysis/Synthesis of a real even function.**
For an even function, the transform is identical to its inverse.

```
CALL RFT(M,X,IX,Y,IY,2)      or
CALL RFT(M,X,IX,Y,IY,5)      or
CALL RCA(M,X,IX,Y,IY)
```

all assume that $x_k = \mathtt{X}(k{*}\mathtt{IX}+1),\ (k=0,1,\dots,n),\ n=2^m=N/2$ define a *half-period* of the function to be analysed and that the other half period is generated by *even* continuation. The results returned are the cosine terms

$$y_j = y_{2n-j} = \mathtt{Y}(j{*}\mathtt{IY}+1), \quad (j=0,1,\dots,n).$$

Note that the full period has $2n = N$ points.

`MODE = 3/6`: **Analysis/Synthesis of a real odd function.**
For an odd function the transform is also identical to its inverse. All assume that $x_k = \mathtt{X}(k{*}\mathtt{IX}+1)$, $(k = 1,2,\dots,n)$;

```
CALL RFT(M,X,IX,Y,IY,3)      or
CALL RFT(M,X,IX,Y,IY,6)      or
CALL RSA(M,X,IX,Y,IY)
```

$n = 2^m = N/2$ define a *half-period* of the function to be analysed and that the other half period is generated by *odd* continuation. The results returned are the sine terms

$$y_j = -y_{2n-j} = \mathtt{Y}(j{*}\mathtt{IY}+1), \quad (j=1,2,\dots,n).$$

Note that $y_0 = y_n = 0$ and that the values returned are $\mathtt{Y(1)} = \mathtt{X(1)}$ and $\mathtt{Y}(n{*}\mathtt{IY}+1) = \mathtt{X}(n{*}\mathtt{IX}+1)$. Again the full period has $2n = N$ points.

Restrictions:

These subroutines work for any input such that the *full period* has at least four points, i.e., $m \geq 2$ for general functions, or $m \geq 1$ for odd or even functions. If the number of data points exceeds 129 ($m \leq 7$), the calling program must provide sufficient working storage by using the statement

```
COMMON /FWORK/ W(nnn)
```

where $\mathtt{nnn} = 5 * 2^m$.

References:

1. C. Iselin, An approach to fast Fourier transform, CERN 71-19.

A copy of Ref. 1 is available.

•

Author(s) : R.C. Singeton (Stanford)
Submitter : B. Fornberg
Language : Fortran

Library: MATHLIB
Submitted: 03.05.1968
Revised: 01.10.1974

Complex Fast Fourier Transform

A discrete Fourier transform is defined by:

$$Y(n) \;=\; \sum_{j=0}^{N-1} X(j)\, \exp\left(\frac{-2\pi i j n}{N}\right), \qquad (n = 0, 1, \ldots, N-1).$$

and the inverse

$$Z(j) \;=\; \sum_{n=0}^{N-1} Y(n)\, \exp\left(\frac{2\pi i j n}{N}\right), \qquad (j = 0, 1, \ldots, N-1)$$

satisfying $Z(j) = NX(j), (j = 0, 1, \ldots, N-1)$. CFT evaluates these sums using fast Fourier technique. It is not required that N is a power of 2. One-, two- and three-dimensional transforms can be performed.

Structure:

SUBROUTINE subprogram
User Entry Names: CFT
Files Referenced: Printer

Usage:

```
CALL CFT(A,B,NTOT,N,NSPAN,ISN)
```

Arrays A and B originally hold the real and imaginary components of the data, and return the real and imaginary components of the resulting Fourier coefficients.

Multivariate data is indexed according to the Fortran array element successor function, without limit on the number of implied multiple subscripts. The SUBROUTINE is called once for each variate. The calls for a multivariate transform may be in any order. NTOT is the total number of complex data values. N is the dimension of the current variable. NSPAN/N is the spacing of consecutive data values while indexing the current variable. The sign of ISN determines the sign of the complex exponential, and the magnitude of ISN is normally one.

For a single-variate transform, NTOT = N = NSPAN = (number of complex data values), e.g.

```
CALL CFT(A,B,N,N,N,1)
```

A tri-variate transform with A(N1,N2,N3), B(N1,N2,N3) is computed by

```
CALL CFT(A,B,N1*N2*N3,N1,N1,1)
CALL CFT(A,B,N1*N2*N3,N2,N1*N2,1)      and
CALL CFT(A,B,N1*N2*N3,N3,N1*N2*N3,1)
```

The data may alternatively be stored in a single COMPLEX array A, then the magnitude of ISN changed to two to give the correct indexing increment and the second parameter used to pass the initial address for the sequence of imaginary values, for example:

```
REAL
EQUIVALENCE (A,S)
...
CALL CFT (A,S(2),NTOT,N,NSPAN,2)
```

Arrays AT(MAXF), CK(MAXF), BT(MAXF), SK(MAXF), and NP(MAXP) are used for temporary storage. If the available storage is insufficient the program is terminated by a STOP.

MAXF must be $\geq$ the maximum prime factor of N. MAXP must be $>$ the number of prime factors of N. In addition, if the square-free portion K of N has two or more prime factors, then MAXP must be $\geq$ K $- 1$. Storage in NFAC allows for a maximum of 11 factors of N. If N has more than one square-free factor, the product of the square-free factors must be ≤ 210.

Notes:

CFT is very general since the number of points is not restricted to powers of two, as is the case for RFT (D700) and FFTRC (D701). For N $=$ 16, 32, 64, 128 the routines in FFTRC (D701) are considerably faster.

References:

1. R.C. Singleton, An Algorithm for Computing the Mixed Radix F.F.T., IEEE Trans. Audio Electroacoust., AU–1(1969) 93–107.
2. Reprinted in: L.R. Rabiner and C.M. Rader: Digital Signal Processing, IEEE Press New York (1972) 294.

•

Author(s) : K.S. Kölbig, H.-H. Umstätter **Library:** MATHLIB
Submitter : **Submitted:** 22.04.1996
Language : Fortran **Revised:**

Real Fast Fourier Transform

Subroutine RFSTFT calculates the finite Fourier transform of a real periodic sequence $y_0, y_1, \ldots, y_{n-1}$, whose period n must be a power of two. Either the direct transform

$$C_j = \frac{1}{n}\sum_{k=0}^{n-1} y_k \exp\left(\frac{-i2\pi jk}{n}\right), \qquad (j = 0, 1, \ldots, n/2), \tag{1}$$

or the inverse transform

$$y_k = \sum_{j=0}^{n-1} C_j \exp\left(\frac{i2\pi jk}{n}\right), \qquad (k = 0, 1, \ldots, n-1), \tag{2}$$

where y_k are real and C_j are complex numbers, may be calculated. Note that $C_j = \overline{C_{n-j}}$, $(j = n/2 + 1, \ldots, n-1)$, where $\overline{\alpha}$ denotes the complex conjugate of α. Thus, only the numbers C_j for which $0 \le j \le n/2$ are calculated.

Structure:

SUBROUTINE subprogram
User Entry Names: RFSTFT
External References: CFSTFT (D706)

Usage:

```
COMPLEX C(0:..)
REAL Y(0:..)
EQUIVALENCE (C,Y)
...
CALL RFSTFT(M,C)
...
```

M (INTEGER) On entry, n is determined by the absolute value of M via $n = 2^{|\mathtt{M}|}$.
$\mathtt{M} < 0$: The direct transform (1) is calculated.
$\mathtt{M} \ge 0$: The inverse transform (2) is calculated.
Unchanged on exit.

C (COMPLEX) One dimensional array of dimension (0:d), where $\mathtt{d} \ge \mathtt{n}/2$.

Y (REAL) One dimensional array of dimension (0:d), where $\mathtt{d} \ge \mathtt{n} + 1$.
$\mathtt{M} < 0$:
On entry, $\mathtt{Y}(k) = y_k$, $(k = 0, 1, \ldots, n-1)$.
On exit, $\mathtt{C}(j) = C_j$, $(j = 0, 1, \ldots, n/2)$, as defined by (1).
$\mathtt{M} \ge 0$:
On entry, $\mathtt{C}(j) = C_j$, $(j = 0, 1, \ldots, n/2)$.
On exit, $\mathtt{Y}(k) = y_k$, $(k = 0, 1, \ldots, n-1)$, as defined by (2).

Method:

The subroutine uses `CFSTFT` (D705) with sequences reduced to half of their length as explaind in Ref. 1.

References:

1. E.O. Brigham, The fast Fourier transform, (Prentice-Hall, Englewood Cliffs, 1974) Ch. 10, Sect. 10, Fig. 10-10.

•

Author(s) : K.S. Kölbig, H.-H. Umstätter
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 22.04.1996
Revised:

Complex Fast Fourier Transform

Subroutine CFSTFT calculates the finite Fourier transform of a complex periodic sequence $a_0, a_1, \ldots, a_{n-1}$, whose period n must be a power of two. Either the direct transform

$$A_j = \sum_{k=0}^{n-1} a_k \exp\left(\frac{-i2\pi jk}{n}\right), \qquad (j = 0, 1, \ldots, n-1), \tag{1}$$

or the unscaled inverse transform

$$\alpha_k = \sum_{j=0}^{n-1} A_j \exp\left(\frac{i2\pi jk}{n}\right), \qquad (k = 0, 1, \ldots, n-1), \tag{2}$$

where a_k, α_k and A_j are complex numbers, may be calculated.

If the A_j in (2) have the values defined by (1), then $a_k = \alpha_k / n$, $(k = 0, 1, \ldots, n-1)$. To ensure optimum use of storage, the same array is used for input and output, and all intermediate calculations are carried out in this array.

Structure:

SUBROUTINE subprogram
User Entry Names: CFSTFT

Usage:

```
CALL CFSTFT(M,A)
```

M (INTEGER) On entry, n is determined by the absolute value of M via $n = 2^{|\mathtt{M}|}$.
M $<$ 0 : The direct transform (1) is calculated.
M $\geq$ 0 : The inverse transform (2) is calculated.
Unchanged on exit.

A (COMPLEX) One dimensional array of dimension (0:d), where d $\geq$ n $-$ 1.
M $<$ 0 :
On entry, $\mathtt{A}(k) = a_k$, $(k = 0, 1, \ldots, n-1)$.
On exit, $\mathtt{A}(j) = A_j$, $(j = 0, 1, \ldots, n-1)$, as defined by (1).
M $\geq$ 0 :
On entry, $\mathtt{A}(j) = A_j$, $(j = 0, 1, \ldots, n-1)$.
On exit, $\mathtt{A}(k) = a_k$, $(k = 0, 1, \ldots, n-1)$, as defined by (2).

Method:

The method is based on an algorithm of Cooley, Lewis and Welch (see **References**), with the following modifications which increase speed for small values of M: multiplications by $\exp(ip\pi)$ are replaced by addition or subtraction, and terms of the form $\exp(i2\pi/q)$, $(q = 2, 4, \ldots, n)$ are calculated recursively using only square roots and divisions.

References:

1. G. Dahlquist and Å. Björck, Numerical methods (Prentice-Hall, Englewood Cliffs, 1974) 416.

2. L.R. Rabiner and B. Gold, Theory and application of digital signal processing (Prentice-Hall, Englewood Cliffs, 1975) 332.

•

Author(s) : F. James **Library:** KERNLIB
Submitter : **Submitted:** 05.09.1966
Language : Fortran **Revised:** 18.11.1985

Polynomial Interpolation

Subroutine `POLINT` interpolates in a table of arguments a_j and function values $f_j = f(a_j)$, using an interpolating polynomial of specified degree $K-1$ which passes through K successive tabular points. The table arguments a_j need not be equidistant. Meaningful results can usually be obtained only for small values of K (typically less than 10).

Structure:

`SUBROUTINE` subprogram
User Entry Names: `POLINT`
Files Referenced: Printer
External References: `KERMTR` (N001), `ABEND` (Z035)

Usage:

```
CALL POLINT(F,A,K,X,R)
```

F (`REAL`) One-dimensional array. `F(j)` must be equal to the value at `A(j)` of the function to be interpolated, ($j = 1, 2, \ldots,$ `K`).

A (`REAL`) One-dimensional array. `A(j)` must be equal to the table argument a_j, ($j = 1, 2, \ldots,$ `K`).

K (`INTEGER`) `K-1` is the degree of the interpolating polynomial.

X (`REAL`) Argument at which the interpolating polynomial is to be evaluated.

R (`REAL`) On exit, `R` is set equal to the value at `X` of the polynomial passing through the points (a_j, f_j), ($j = 1, 2, \ldots,$ `K`).

If `X` lies outside the range of the points `A(1)`, ..., `A(K)`, the interpolation becomes an extrapolation, with consequent loss of accuracy.

Method:

Newton's divided difference formula is used.

Restrictions:

$2 \leq$ `K` $\leq$ `20`. If `K` $>$ `20`, the interpolation is performed as if `K` had the value `20`. The original value of `K` is unchanged on exit.

Error handling:

Error `E100.1`: `K` $<$ `1`. A message is printed unless subroutine `KERSET` (N001) has been called.

Notes:

`POLINT` is intended for interpolation using *all* the tabulated points in the array `A`. To use only the tabulated points around the value of the argument `X`, use `DIVDIF` (E105).

•

Author(s) : K.S. Kölbig, H. Lipps **Library:** MATHLIB
Submitter : **Submitted:** 29.08.1968
Language : Fortran **Revised:** 01.12.1994

Maximum and Minimum Elements of Arrays

Function subprograms `MAXIZE`, `MAXRZE`, `MAXDFZ` and `MINIZE`, `MINRZE`, `MINDFZ` give give the positions of the maximum and minimum elements of a one-dimensional array.

On CDC and Cray computers, the double-precision versions `MAXDZE` and `MINDZE` are not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `MAXIZE`, `MAXRZE`, `MAXDZE`, `MINIZE`, `MINRZE`, `MINDZE`
Obsolete User Entry Names: `MAXFZE` ≡ `MAXRZE`, `MINFZE` ≡ `MINRZE`

Usage:

In any arithmetic expression, for `t` = `I` (type `INTEGER`), `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

`MAXtZE(A(J),N)`   and   `MINtZE(A(J),N)`

has the `INTEGER` value of the location of, respectively, the maximum and minimum elements of the `N` successive elements of the array `A`, **relative to the element** `A(J)`, where `A` is of type `t`.

Notes:

1. If there is more than one location at which the maximum or minimum is attained, the first location is returned as the function value in each case.
2. If `N` < `1` the function value is `1`.
3. Clearly, `N+J` should not exceed the dimension of the array `A`.
4. The obsolete older entries `MAXFZE` (for `MAXRZE`) and `MINFZE` (for `MINRZE`) are kept for a transitional period. They will eventually disappear.

•

Author(s) : J. Zoll
Submitter : C. Letertre
Language : Fortran

Library: KERNLIB
Submitted: 01.09.1969
Revised:

Largest Absolute Number in Scattered Vector

`AMAXMU` looks for the largest absolute value in a scattered vector of real numbers.

Structure:

`FUNCTION` subprogram
User Entry Names: `AMAXMU`

Usage:

In any arithmetic expression,

```
AMAXMU(A,IDO,IW,NA)
```

is set to the largest absolute value of numbers in any of the subsets of `A` as specified by `IDO`, `IW` and `NA`.

`A`	`(REAL)` One-dimensional array, containing a number of subsets of real numbers.
`IDO`	`(INTEGER)` Number of subsets to be examined.
`IW`	`(INTEGER)` Number of words in each subset.
`NA`	`(INTEGER)` Specifies the distance between the first elements of consecutive subsets.

Notes:

To find the largest element in a continuous vector, `VMAXA` (F121) is faster than `AMAXMU`.

Examples:

```
X=AMAXMU(A,4,1,2)
```

sets `X` equal to the largest absolute value of `A(1)`, `A(3)`, `A(5)` and `A(7)`.

•

Author(s) : C. Letertre
Submitter : B. Schorr
Language : Fortran

Library: KERNLIB
Submitted: 17.05.1971
Revised: 27.11.1984

Multidimensional Linear Interpolation

Function subprogram `FINT` uses repeated linear interpolation to evaluate a function $f(x_1, x_2, \ldots, x_n)$ of n variables which has been tabulated at the nodes of an n-dimensional rectangular grid. It is not necessary that the table arguments corresponding to any coordinate x_i be equally spaced.

Structure:

`FUNCTION` subprogram
User Entry Names: `FINT`
Files Refernced: Printer
External References: `KERMTR` (N001), `ABEND` (Z035)

Usage:

In any arithmetic expression,

```
FINT(N,X,NA,A,F)
```

has an approximate value of $f(\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_\mathtt{n})$.

`N` (`INTEGER`) Number of coordinates n required to define the function f.

`X` (`REAL`) One-dimensional array. `X(j)`, $(j = 1, 2, \ldots, \mathtt{N})$, must contain the coordinates of the point at which the interpolation is to be performed.

`NA` (`INTEGER`) One-dimensional array. For $j = 1, 2, \ldots, \mathtt{N}$, `NA(j)` must be equal to the number of numerical values of variable x_j which are stored in array `A`.

`A` (`REAL`) One-dimensional array of length not less than the sum of `NA(1)`, ..., `NA(N)`. The first `NA(1)` elements of `A` must contain numerical values $a_{11}, a_{12}, \ldots$ of the first variable x_1 in strictly increasing order, the next `NA(2)` elements of `A` must contain numerical values $a_{21}, a_{22}, \ldots$ of the second variable x_2 in strictly increasing order, and so on.

`F` (`REAL`) Multidimensional array with dimensions `NA(1), NA(2), ...,NA(N)`, containing values of the function f at the nodes of the rectangular grid defined by `A`:
$\mathtt{F(i, j, \ldots, m)} = f(a_{1\mathtt{i}}, a_{2\mathtt{j}}, \ldots, a_{n\mathtt{m}})$, $(i = 1, 2, \ldots, \mathtt{NA(1)}, \ldots; m = 1, 2, \ldots, \mathtt{NA(N)})$.

If any coordinate x_i of the given point X lies outside the range of the corresponding table arguments, the interpolation for this coordinate is replaced by an extrapolation based on the two nearest table arguments, with consequent loss of accuracy.

Method:

Repeated linear interpolation with respect to variables $x_1, x_2, \ldots$ within the grid cell which contains the given point X. For $n = 2$, with (x_1, x_2) replaced by (x, y) for clarity, the procedure is equivalent to the following:

Let $a_1, a_2, \ldots$ be the tabulated values of x. Let $b_1, b_2, \ldots$ be the tabulated values of y.
Let i and j be the subscripts for which $a_i \le x < a_{i+1}, b_j \le y < b_{j+1}$.

Then compute:

$$
\begin{aligned}
t &= (x-a)/(a_{i+1}-a), \\
g_j &= (1-t)f(a_i,b_j)+tf(a_{i+1},b_j), \\
g_{j+1} &= (1-t)f(a_i,b_{j+1}+tf(a_{i+1},b_{j+1}), \\
u &= (y-b)/(b_{j+1}-b), \\
f_{appr} &= (1-u)g_j+ug_{j+1}
\end{aligned}
$$

Restrictions:

1. $\mathtt{1} \le \mathtt{N} \le \mathtt{5}$. `FINT` is set equal to zero if `N` is not in this range.
2. $\mathtt{NA(j)} \ge \mathtt{1}$, $(\mathtt{j} = \mathtt{1}, \mathtt{2}, \ldots, \mathtt{N})$.
3. The table arguments for each variable must be in strictly increasing order.

There is no test for conditions 2 or 3.

Error handling:

`E104.1`: $\mathtt{N} < \mathtt{1}$ or $\mathtt{N} > \mathtt{5}$. `FINT` is set equal to zero, and a message is printed unless subroutine `KERSET (N001)` has been called.

Examples:

Given a function of two variables $g(x,y)$ defined by a `FUNCTION` subprogram `G`, to construct a table of values of $f_{km} = g(\sqrt{k}, \log m)$ for $k = 1, 2, \ldots, 10$, $m = 1, 2, \ldots, 15$, and to interpolate in this table to set `GINT` equal to an approximate value of $g(1.7, 2.9)$. The program is written in a form which allows generalization to functions of more than two variables.

```
      PARAMETER (NA1=10,NA2=15)
      DIMENSION X(2),NA(2),A(NA1+NA2),F(NA1,NA2)
      DATA NA/NA1,NA2/
C     STORE ARGUMENT ARRAY
      K1=0
      K2=K1+NA1
      DO 1 J = 1,MAX(NA1,NA2)
        IF (J .LE. NA1) A(J+K1)=SQRT(FLOAT(J))
        IF (J .LE. NA2) A(J+K2)=LOG(FLOAT(J))
    1 CONTINUE
C     STORE FUNCTION ARRAY
      DO 3 J1 = 1,NA1
        DO 2 J2 = 1,NA2
          F(J1,J2)=G(A(J1+K1),A(J2+K2))
    2   CONTINUE
    3 CONTINUE
C     INTERPOLATE IN TABLE
      X(1)=1.7
      X(2)=2.9
      GINT=FINT(2,X,NA,A,F)
      ...
```

•

Author(s) : F. James
Submitter : G.A. Erskine
Language : Fortran
Library: KERNLIB
Submitted: 19.07.1973
Revised: 27.11.1984

Function Interpolation

Function subprogram `DIVDIF` interpolates in a table of arguments a_j and function values $f_j = f(a_j)$, using an interpolating polynomial of specified degree which passes through tabular points which are symmetrically-positioned around the interpolation argument. The table arguments a_j need not be equidistant.

Structure:

`FUNCTION` subprogram
User Entry Names: `DIVDIF`
Files Referenced: Printer
External References: `KERMTR` (N001), `ABEND` (Z035)

Usage:

In any arithmetic expression,

```
DIVDIF(F,A,N,X,M)
```

has an approximate value of $f(\mathtt{X})$.

- `F` (`REAL`) One-dimensional array. `F(j)` must be equal to the value at `A(j)` of the function to be interpolated, ($j = 1, 2, \ldots, \mathtt{N}$).
- `A` (`REAL`) One-dimensional array. `A(j)` must be equal to the table argument a_j, ($j = 1, 2, \ldots, \mathtt{N}$).
- `N` (`INTEGER`) Number of values in arrays `F` and `A`.
- `X` (`REAL`) Argument at which the interpolating polynomial is to be evaluated.
- `M` (`INTEGER`) Requested degree of the interpolating polynomial. If `M` exceeds $M_{max} = \min(10, \mathtt{N}-1)$ the interpolation is carried out using a polynomial of degree M_{max} instead of `M`. The original value of `M` is unchanged on exit.

Method:

Newton's divided difference formula is used. Except when `X` lies near one end of the table (in which case unsymmetrically-situated interpolation points are used), the interpolation procedure is as follows:
`M` **odd**:
An interpolating polynomial passing through $\mathtt{M}+1$ successive points (a_j, f_j) symmetrically placed with respect to `X` is used.
`M` **even**:
The mean of two interpolating polynomials is used, each passing through $\mathtt{M}+1$ successive points (a_j, f_j), one polynomial having an extra point to the left of `X`, the other having an extra point to the right of `X`.
If `X` lies too close to either end of the table for symmetrically-positioned tabular values to be used, the $\mathtt{M}+1$ values at the end of the table are used. If `X` lies outside the range of the table, the interpolation becomes an extrapolation, with corresponding loss of accuracy.

Restrictions:

The argument values $\mathtt{A}(1), \mathtt{A}(2), \ldots$ must be in either strictly increasing order or strictly decreasing order. No error message is printed if this is not true.

Error handling:

Error `E105.1`: `N` $<$ `2` or `M` $<$ `1`. `DIVDIF` is set equal to zero and a message is printed unless subroutine `KERSET (N001)` has been called.

Notes:

See also the write-up for `POLINT` (E100).

•

Author(s) : F. James, K.S. Kölbig
Submitter :
Language : Fortran
Library: KERNLIB
Submitted: 18.10.1974
Revised: 15.11.1995

Binary Search for Element in Ordered Array

Integer function subprograms LOCATI and LOCATR perform a binary search in an array of non-decreasing integer or real numbers $a_1 \leq a_2 \leq \ldots \leq a_n$ to locate a specified value t.

Structure:

FUNCTION subprograms
User Entry Names: LOCATI, LOCATR
Obsolete User Entry Names: LOCATF $\equiv$ LOCATR
On CDC or Cray computers, the double-precision version LOCATD is not available.

Usage:

In any arithmetic expression, for $\mathtt{t} = \mathtt{I}$ (type INTEGER), $\mathtt{t} = \mathtt{R}$ (type REAL), $\mathtt{t} = \mathtt{D}$ (type DOUBLE PRECISION),

```
LOCATt(tA,N,tT)
```

has the INTEGER value according to the description below.

tA	(type according to t) One-dimensional array. The numbers tA(j) must form a non-decreasing sequence for $j = 1, 2, \ldots,$ N.
N	(INTEGER) Number n of elements in array tA.
tT	(type according to t) Search value t.

Depending on four possible outcomes of the search, each subprogram returns the following value L ($a =$ tA, $t =$ tT):

$a_j = t$ for some j with $1 \leq j \leq \mathtt{N}$	$\mathtt{L} = j$
$t < a_1$	$\mathtt{L} = 0$
$a_k < t < a_{k+1}$ for some k with $1 \leq k \leq \mathtt{N} - 1$	$\mathtt{L} = -k$
$a_n < t$	$\mathtt{L} = -\mathtt{N}$

If the value t occurs more than once in the array a, the result L may correspond to any of the occurrences.

Method:

Repeated bisection of the subscript range.

Notes:

1. The number of comparisons performed is approximately proportional to $\ln \mathtt{N}$. Therefore, for large N the binary search is considerably faster than a sequential search using a DO loop. However, for N less than about 40 a DO loop is faster.
2. The obsolete older entry LOCATF is kept for a transitional period. It will eventually disappear.

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 01.12.1994
Language : Fortran | **Revised:**

Least Squares Polynomial Fit

Subroutine subprograms RLSQPM and DLSQPM fit a polynomial

$$p_m(x) \;=\; \sum_{j=0}^{m} a_j x^j$$

of degree m to n equally-weighted data points (x_i, y_i). The calculated coefficients a_j are such that

$$S_m^2 \;=\; \sum_{i=1}^{n} \bigl(y_i - p_m(x_i)\bigr)^2 \;=\; \min .$$

Subroutine subprograms RLSQP1 and DLSQP1 fit a straight line $p_1(x) \;=\; a_0 + a_1 x$ to n such points.
Subroutine subprograms RLSQP2 and DLSQP2 fit a parabola $p_2(x) \;=\; a_0 + a_1 x + a_2 x^2$ to n such points.
An estimate $s \;=\; \sqrt{S_m^2/(n-m-1)}$ of the standard deviation σ is calculated.
On CDC and Cray computers, the double-precision versions DLSQPM, DLSQP1 and DLSQP2 are not available.

Structure:

SUBROUTINE subprograms
User Entry Names: RLSQPM, RLSQP1, RLSQP2, DLSQPM, DLSQP1, DLSQP2
External References: RVSET (F002), DVSET (F002), DVSUM (F002), DVMPY (F002), DSEQN (F012)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tLSQPM(N,X,Y,M,A,SD,IFAIL)
CALL tLSQP1(N,X,Y,A0,A1,SD,IFAIL)
CALL tLSQP2(N,X,Y,A0,A1,A2,SD,IFAIL)
```

N — (INTEGER) Number n of data points.

X — (type according to t) One-dimensional array of length $\geq$ N. On entry, X(i) contains the abscissas $x_i,\ (i = 1, 2, \ldots, n)$.

Y — (type according to t) One-dimensional array of length $\geq$ N. On entry, Y(i) contains the ordinates $y_i,\ (i = 1, 2, \ldots, n)$.

M — (INTEGER) Degree m of the polynomial to be fitted.

A — (type according to t) One-dimensional array of dimension (0:d), where d $\geq$ M. Contains, on exit, in A(j) the coefficients $a_j,\ (j = 0, 1, \ldots, m)$.

A0,A1,A2 — (type according to t) Contain, on exit, the coefficients a_0, a_1 for $p_1(x) = a_0 + a_1 x$ or a_0, a_1, a_2 for $p_2(x) = a_0 + a_1 x + a_2 x^2$, respectively.

SD — (type according to t) Contains, on exit, the estimate s.

IFAIL — (INTEGER) Error flag.
= 0 : Normal case,
= 1 : N $\leq$ 1 or M < 0 or M $\geq$ N or M > 20,
= −1 : The matrix of normal equations is numerically singular.

In the case IFAIL $\neq$ 0: M = 0, A(j) = 0 and A0 = A1 = A2 = 0 on exit.

Method:

The normal equations are solved. On computers other than CDC or Cray, double-precision mode arithmetic is used internally for `RLSQPM`, `RLSQP1` and `RLSQP2`.

Notes:

Meaningful results can usually be obtained only for small values of m (typically < 10).

References:

1. D.H. Menzel, Fundamental formulas of physics, v. 1, (Dover, New York 1960) 116–122.

•

Author(s) : E. Keil | **Library:** KERNLIB
Submitter : B. Schorr | **Submitted:** 01.12.1969
Language : Fortran | **Revised:** 27.11.1984

Least Squares Polynomial Fit

OBSOLETE

Please note that this routine has been obsoleted in CNL 218. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: RLSQPM (E201)

Subroutine LSQ fits a polynomial of degree $m-1$ to n equally-weighted data points (x_i, y_i). The computed coefficients a_j of the fitted polynomial have values which minimize

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{m} a_j x_i^{j-1}\right)^2 .$$

For the case $m = 2$ (straight line fit), subroutine LLSQ is faster and easier to use than LSQ.

Meaningful results can usually be obtained only for small values of m (typically less than 10).

Structure:

SUBROUTINE subprograms
User Entry Names: LSQ, LLSQ
Files Referenced: Printer
External References: RVSUM (F002), RSEQN (F012), DSEQN (F012), KERMTR (N001), ABEND (Z035)

Usage:

```
CALL LSQ(N,X,Y,M,A)
CALL LLSQ(N,X,Y,A1,A2,IFAIL)
```

N — (INTEGER) Number n of data points.

X — (REAL) One-dimensional array. X(i) must be equal to the data coordinate x_i, $(i = 1, 2, \ldots, \texttt{N})$.

Y — (REAL) One-dimensional array. Y(i) must be equal to the observed value y_i, $(i = 1, 2, \ldots, \texttt{N})$.

M — (INTEGER) On entry, M must be equal to the number m of coefficients of the polynomial to be fitted. On exit, the value of M may differ from this (see **Error Handling**).

A — (REAL) One-dimensional array. On exit from LSQ, A(j) is equal to the coefficient of x^{j-1} in the fitted polynomial, $(j = 1, 2, \ldots, \texttt{M})$.

A1,A2 — (REAL) On exit from LLSQ, A1 and A2 are equal to the coefficients of the fitted straight line $a_1 + a_2 x$.

IFAIL — (INTEGER) On exit from LLSQ, IFAIL is equal to -2 if N $<$ 2, to -1 if the matrix of normal equations is numerically singular, and to zero otherwise.

Method:

Normal equations.

Error handling:

Error `E208.1:` $M < 1$ or $M > N$ or $M > 20$ (subroutine `LSQ`). `M` is replaced by zero.
Error `E208.2:` The normal equations matrix is numerically singular (subroutine `LSQ`).
For each error, a message is printed unless subroutine `KERSET (N001)` has been called.

Notes:

On computers other than Cray and CDC double-precision arithmetic is used internally.
•

NORBAS	CERN Program Library	E210

Author(s) : W. Mönch, B. Schorr **Library:** MATHLIB
Submitter : W. Mönch **Submitted:** 15.03.1993
Language : Fortran **Revised:**

Polynomial Splines / Normalized B-Splines

`NORBAS` (`NOR`malized `BA`sis `S`plines) is a portable collection of subprograms for various calculations with polynomial splines in one dimension (**1D**) and in two dimensions (**2D**). The polynomial splines are represented as linear combinations of normalized basis splines (B-splines).

On computers other than CDC or Cray, only the double-precision versions `DSPKN1`, etc. are available. On CDC and Cray computers, only the single-precision versions `RSPKN1`, etc. are available.

The following outline provides the background material and the notations needed for describing the subprograms and their parameters. For further information about splines and their applications see **References**, in particular Ref. 7.

Case (1D):

k Degree (order -1) of the B-spline $(0 \le k \le 25)$.

m Number of spline-knots $(m \ge 2\,k+2)$.

i Index of the B-spline $(1 \le i \le m-k-1)$.

τ Set of m spline-knots $\tau = \{t_1, t_2, \ldots, t_m\}$, in non-decreasing order, with multiplicity $\le k+1$ (i.e. no more than $k+1$ knots coincide).

$[a,b]$ Interval, defined by $a = t_{k+1}$, $b = t_{m-k}$.

$B_i(x)$ Normalized B-spline of degree k over τ with index i. The value of $B_i(x)$ is identically zero outside the interval $t_i \le x \le t_{i+k+1}$, and the normalization of $B_i(x)$ is such that

$$\int_{-\infty}^{+\infty} B_i(x)\,dx \;=\; \frac{t_{i+k+1}-t_i}{k+1} \qquad (i = 1, \ldots, m-k-1).$$

$s(x)$ Polynomial spline at $x \in [a,b]$ in B-spline representation

$$y \;=\; s(x) \;=\; \sum_{i=1}^{m-k-1} c_i\, B_i(x)\,.$$

Spline interpolation to a data set:

Given a data set $\{x_l, y_l\}_{l=1,\ldots,n}$; determine coefficients $\{c_i\}_{i=1,\ldots,n}$ of a polynomial interpolation spline $y = s(x)$ in B-spline representation with degree k over a set τ of $m = n+k+1$ knots, such that the following relations (interpolation conditions) hold:

$$s(x_l) \;=\; y_l \qquad (l = 1, \ldots, n).$$

The existence of a solution of this interpolation problem depends on an appropriate choice of the spline-knots (Ref. 7, Theorem XIII.1 (Schoenberg-Whitney)).

Least squares spline approximation to a data set:

Given a data set $\{x_l, y_l\}_{l=1,\ldots,n}$; determine coefficients $\{c_i\}_{i=1,\ldots,m-k-1}$ of a polynomial approximation spline $y = s(x)$ in B-spline representation with degree k over a set τ of $m \le n+k+1$ knots, such that following least squares problem is solved:

$$\phi(c_1, \ldots, c_{m-k-1}) \;=\; \sum_{l=1}^{n} \left(y_l - s(x_l)\right)^2 \;=\; \texttt{min!}$$

Variation diminishing spline approximation to a function (Schoenberg):
For a given function $y = f(x)$ on $[a, b]$ this spline approximation is defined by $y = s(x)$, with (Ref. 7, p. 158-162)

$$c_i \;=\; f(x_i); \qquad x_i \;=\; (t_{i+1} + \cdots + t_{i+k})/k \quad (i = 1, \ldots, m-k-1;\; k \geq 1).$$

Case (2D):

kx	Degree of one-dimensional B-splines in x-direction $(0 \leq kx \leq 25)$.
ky	Degree of one-dimensional B-splines in y-direction $(0 \leq ky \leq 25)$.
mx	Number of spline-knots in x-direction $(mx \geq 2\,kx + 2)$.
my	Number of spline-knots in y-direction $(my \geq 2\,ky + 2)$.
i	Index of B-spline $(1 \leq i \leq mx - kx - 1)$ in x-direction.
j	Index of B-spline $(1 \leq j \leq my - ky - 1)$ in y-direction.
τ_x	Set of mx spline-knots $\tau_x = \{t_{x,1}, t_{x,2}, \ldots, t_{x,mx}\}$ in x-direction, in non-decreasing order, with multiplicity $\leq kx + 1$ (i.e. no more than $kx + 1$ knots coincide).
τ_y	Set of my spline-knots $\tau_y = \{t_{y,1}, t_{y,2}, \ldots, t_{y,my}\}$ in y-direction, in non-decreasing order, with multiplicity $\leq ky + 1$ (i.e. no more than $ky + 1$ knots coincide).
$[a_x, b_x]$	Interval in x-direction, defined by $a_x = t_{x,kx+1}$, $b_x = t_{x,mx-kx}$.
$[a_y, b_y]$	Interval in y-direction, defined by $a_y = t_{y,ky+1}$, $b_y = t_{y,my-ky}$.
$B_i(x)$	B-spline of degree kx over τ_x with index i.
$B_j(y)$	B-spline of degree ky over τ_y with index j.
$B_{i,j}(x,y)$	Product $B_{i,j}(x,y) = B_i(x)\,B_j(y)$ of two one-dimensional B-splines.
$s(x,y)$	Two-dimensional polynomial spline at $(x,y) \in [a_x, b_x] \times [a_y, b_y]$ in B-spline representation

$$z \;=\; s(x,y) \;=\; \sum_{i=1}^{mx-kx-1} \sum_{j=1}^{my-ky-1} c_{i,j}\, B_{i,j}(x,y).$$

Spline interpolation to a data set:
Given a data set $\{x_{lx}, y_{ly}, z_{lx,ly}\}_{lx=1,\ldots,nx;ly=1,\ldots,ny}$; determine coefficients $\{c_{i,j}\}_{i=1,\ldots,nx;j=1,\ldots,ny}$ of a two-dimensional polynomial interpolation spline $z = s(x,y)$ in B-spline representation with degrees kx , ky over the sets τ_x of $mx = nx + kx + 1$ knots in x-direction and τ_y of $my = ny + ky + 1$ knots in y-direction, such that following relations (interpolation conditions) hold:

$$s(x_{lx}, y_{ly}) = z_{lx,ly} \qquad (lx = 1, \ldots, nx\,;\; ly = 1, \ldots, ny).$$

The existence of a solution of this interpolation problem depends on an appropriate choice of the spline-knots τ_x , τ_y in the two-dimensional interpolation area $[a_x, b_x] \times [a_y, b_y]$.

Least squares spline approximation to a data set:
Given a data set $\{x_{lx}, y_{ly}, z_{lx,ly}\}_{lx=1,\ldots,nx;ly=1,\ldots,ny}$; determine coefficients $\{c_{i,j}\}_{i=1,\ldots,nx;j=1,\ldots,ny}$ of a two-dimensional polynomial approximation spline $z = s(x,y)$ in B-spline representation with degrees kx, ky over the sets τ_x of $mx \leq nx + kx + 1$ knots in x-direction and τ_y of $my \leq ny + ky + 1$ knots in y-direction, such that following least squares problem is solved:

$$\phi(c_{1,1}, \ldots, c_{mx-kx-1,my-ky-1}) = \sum_{lx=1}^{nx} \sum_{ly=1}^{ny} \left(z_{lx,ly} - s(x_{lx}, y_{ly})\right)^2 \;=\; \texttt{min!}$$

Variation diminishing spline approximation to a function:
For a given function $z = f(x,y)$ on $[a_x, b_y] \times [a_y, b_y]$ this two-dimensional spline approximation is defined by $z = s(x,y)$ on $[a_x, b_x] \times [a_y, b_y]$, with

$$\begin{aligned} c_{i,j} = f(x_i, y_j); \qquad & x_i = (t_{x,i+1} + \cdots + t_{x,i+kx})/kx \quad (i = 1, \ldots, mx - kx - 1;\; kx \geq 1), \\ & y_j = (t_{y,j+1} + \cdots + t_{y,j+ky})/ky \quad (j = 1, \ldots, my - ky - 1;\; ky \geq 1). \end{aligned}$$

The package `NORBAS` contains `FUNCTION` and `SUBROUTINE` subprograms for solving the following problems.
To calculate:

(K) A set τ of m spline-knots in the interval $[a, b]$ for normalized B-splines $B_i(x)$ of degree k, use subprogram `tSPKN1` (**1D**). The knots are in non-decreasing order and determined in such a way that the first $k+1$ knots coincide with a, the last $k+1$ knots coincide with b, and the remaining $(m-2\,k-2)$ knots are equidistant in (a, b).
Two sets τ_x , τ_y of spline-knots in $[a_x, b_x]$ and $[a_y, b_y]$ for B-splines $B_{i,j}(x, y)$ of degrees kx and ky in x- and y-direction, use subprogram `tSPKN2` (**2D**). τ_x and τ_y, are calculated by the same formulae in x-, and y-direction, as in case (**1D**).

(B) The function $B_i(x)$,

$$\text{the } n\text{-th derivative} \quad \frac{d^n B_i(x)}{dx^n}, \qquad \text{or the integral} \quad \int_{-\infty}^{x} B_i(\xi)\, d\xi$$

of a normalized B-spline $B_i(x)$, with fixed degree k and index i over a set τ of spline-knots, use subprogram: `tSPNB1` (**1D**).
The function $B_{i,j}(x, y)$,

$$\text{the partial derivative} \quad \frac{\partial^{nx}\,\partial^{ny}\,B_{i,j}(x, y)}{\partial x^{nx}\,\partial y^{ny}}, \qquad \text{or the integral} \quad \int_{-\infty}^{x}\int_{-\infty}^{y} B_{i,j}(\xi, \eta)\, d\eta\, d\xi$$

of a two-dimensional B-spline $B_{i,j}(x, y)$, with fixed degrees kx, ky and indices i, j over the sets τ_x, τ_y of spline-knots, use subprogram `tSPNB2` (**2D**).

(P) The function $s(x)$,

$$\text{the } n\text{-th derivative} \quad \frac{d^n s(x)}{dx^n}, \qquad \text{or the integral} \quad \int_{-\infty}^{x} s(\xi)\, d\xi$$

of a polynomial spline $y = s(x)$ in B-spline representation with given coefficients c_i, use subprogram `tSPPS1` (**1D**).
The function $s(x, y)$,

$$\text{the partial derivative} \quad \frac{\partial^{nx}\,\partial^{ny}\,s(x, y)}{\partial x^{nx}\,\partial y^{ny}}, \qquad \text{or the integral} \quad \int_{-\infty}^{x}\int_{-\infty}^{y} s(\xi, \eta)\, d\eta\, d\xi$$

of a two-dimensional polynomial spline $z = s(x, y)$ in B-spline representation with given coefficients $c_{i,j}$, use subprogram `tSPPS2` (**2D**).

(I) The coefficients c_i of a one-dimensional polynomial interpolation spline $y = s(x)$ in B-spline representation to a user-supplied data set $\{x_l, y_l\}$, use subprogram `tSPIN1` (**1D**).
The coefficients $c_{i,j}$ of a two-dimensional polynomial interpolation spline $z = s(x, y)$ in B-spline representation to a user-supplied data set $\{x_{lx}, y_{ly}, z_{lx,ly}\}$, use subprogram `tSPIN2` (**2D**).

(A) The coefficients c_i of a one-dimensional polynomial least squares approximation spline $y = s(x)$ in B-spline representation to a user-supplied data set $\{x_l, y_l\}$, use subprogram `tSPAP1` (**1D**).
The coefficients $c_{i,j}$ of a two-dimensional polynomial least squares approximation spline $z = s(x, y)$ in B-spline representation to a user-supplied data set $\{x_{lx}, y_{ly}, z_{lx,ly}\}$, use subprogram `tSPAP2` (**2D**).

(V) The coefficients c_i of a one-dimensional polynomial variation diminishing spline approximation $y = s(x)$ in B-spline representation to a user-supplied function $y = f(x)$, use subprogram `tSPVD1` (**1D**).
The coefficients $c_{i,j}$ of a two-dimensional polynomial variation diminishing spline approximation $z = s(x, y)$ in B-spline representation to a user-supplied function $z = f(x, y)$, use subprogram `tSPVD2` (**2D**).

(D) From given coefficients c_i of a one-dimensional polynomial spline $y = s(x)$ in B-spline representation, the corresponding coefficients d_i of its n-th derivative $d^n s(x)/dx^n$ in B-spline representation, use subprogram `tSPCD1` (**1D**).
From given coefficients $c_{i,j}$ of a two-dimensional polynomial spline $z = s(x, y)$ in B-spline representation, the corresponding coefficients $d_{i,j}$ of its partial derivative $\partial^{nx}\,\partial^{ny}\,s(x, y)/\{\partial x^{nx}\,\partial y^{ny}\}$ in B-spline representation, use subprogram `tSPCD2` (**2D**).

Structure:

SUBROUTINE and FUNCTION subprograms
User Entry Names: RSPKN1, RSPKN2, RSPNB1, RSPNB2, RSPPS1, RSPPS2, RSPIN1, RSPIN2, RSPAP1, RSPAP2, RSPVD1, RSPVD2, RSPCD1, RSPCD2, DSPKN1, DSPKN2, DSPNB1, DSPNB2, DSPPS1, DSPPS2, DSPIN1, DSPIN2, DSPAP1, DSPAP2, DSPVD1, DSPVD2, DSPCD1, DSPCD2,
Internal Entry Names: RSPAS1, RSPAS2, RSPLKK, DSPAS1 DSPAS2, DSPLKK
Files Referenced: Unit 6
External References: RGBTRF (F001), RGBTRS (F001), RGESVD (F001), DGBTRF (F001), DGBTRS (F001), DGESVD (F001), RVSET (F002), RVSUM (F002), RVCPY (F002), RVMPY (F002), DVSET (F002), DVSUM (F002), DVCPY (F002), DVMPY (F002), RMCPY (F003), RMMPY (F003), DMCPY (F003), DMMPY (F003), MTLMTR (N002), ABEND (Z035).
User-supplied FUNCTION subprogram

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION):

(K) Knots

```
CALL tSPKN1(K,M,A,B,T,NERR)
CALL tSPKN2(KX,KY,MX,MY,AX,BX,AY,BY,TX,TY,NERR)
```

(B) Normalized B-splines

```
tSPNB1(K,M,I,NDER,X,T,NERR)
tSPNB2(KX,KY,MX,MY,I,J,NDERX,NDERY,X,Y,TX,TY,NERR)
```

(P) Polynomial spline

```
tSPPS1(K,M,NDER,X,T,C,NERR)
tSPPS2(KX,KY,MX,MY,NDERX,NDERY,X,Y,TX,TY,C,NDIMC,W,NERR)
```

(I) Spline interpolation

```
CALL tSPIN1(K,N,XI,YI,KNOT,T,C,W,IW,NERR)
CALL tSPIN2(KX,KY,NX,NY,XI,YI,ZI,NDIMZ,KNOT,TX,TY,C,NDIMC,W,IW,NERR)
```

(A) Least squares spline approximation

```
CALL tSPAP1(K,M,N,XI,YI,KNOT,T,C,W,NW,NERR)
CALL tSPAP2(KX,KY,MX,MY,NX,NY,XI,YI,ZI,NDIMZ,KNOT,TX,TY,C,NDIMC,W,NW,NERR)
```

(V) Variation diminishing spline approximation

```
CALL tSPVD1(F,K,M,T,C,NERR)
CALL tSPVD2(F,KX,KY,MX,MY,TX,TY,C,NDIMC,NERR)
```

(D) Coefficients of derivatives

```
CALL tSPCD1(K,M,NDER,T,C,D,NERR)
CALL tSPCD2(KX,KY,MX,MY,NDERX,NDERY,TX,TY,C,NDIMC,D,NERR)
```

Case (1D):

F Name of a user-upplied `FUNCTION` subprogram, declared `EXTERNAL` in the calling program. This subprogram must provide the value of the function $y = f(x)$ for variation diminishing spline approximation.

K (`INTEGER`) Degree of B-splines ($1 \leq$ K ≤ 25 for tSPVD1, $0 \leq$ K ≤ 25 otherwise).

M (`INTEGER`) Number of knots ($\geq 2 *$ K $+ 2$).

I (`INTEGER`) Index of B-splines ($1 \leq$ I $\leq$ M $-$ K $- 1$).

N (`INTEGER`) Number of data points $\{x_l, y_l\}$ ($\geq$ K $+ 1$ for spline interpolation (tSPIN1); $\geq$ M $-$ K $- 1$ for spline approximation (tSPAP1)).

NDER (`INTEGER`) Selects one out of three cases: (i) integral, (ii) function value, or (iii) derivative. ($-1 \leq$ NDER $\leq$ K for tSPNB1 and tSPPS1; $1 \leq$ NDER $\leq$ K for tSPCD1).

$= -1$: Calculation of the integral of $B_i(x)$ (tSPNB1), or the integral of $s(x)$ (tSPPS1).

$= 0$: Calculation of the function value $B_i(x)$ (tSPNB1), or the function value $s(x)$ (tSPPS1).

≥ 1 : Calculation of the NDER-th derivative of $B_i(x)$ (tSPNB1), or the NDER-th derivative of $s(x)$ (tSPPS1).

X (Type according to t) Independent variable x of polynomial spline $s(x)$ or B-spline $B_i(x)$.

XI (Type according to t) One-dimensional array of length $\geq$ N. On entry, XI(L) must contain the l-th data point x_l for spline interpolation (tSPIN1) or spline approximation (tSPAP1). The data points must be in ascending order.

YI (Type according to t) One-dimensional array of length $\geq$ N. On entry, YI(L) must contain the l-th data point y_l for spline interpolation (tSPIN1) or spline approximation (tSPAP1).

KNOT (`INTEGER`) Controls the mode of supplying the knots for spline interpolation or approximation.

$= 1, 2$: The knots are computed by the subprograms tSPIN1 and tSPAP1. At the left and right end-point of the interpolation (approximation) interval $[x_1, x_n]$ arise multiple knots. The remaining knots are either equidistant (KNOT $= 1$) or are computed by using the data points $\{x_l\}$ of interpolation (approximation) (KNOT $= 2$).

$\neq 1, 2$: The knots must be supplied by the user.

A,B (Type according to t) On entry, A and B must contain the left (right) end-point of the interval $[a, b]$ for the calculation of a set of spline knots (tSPKN1).

T (Type according to t) One-dimensional array of length $\geq$ M .
For tSPKN1 and for tSPINT1, tSPAP1 if KNOT $= 1, 2$: On exit, T(J) contains the j-th knot. In the other cases, on entry, T(J) must contain the j-th knot. The knots must be in non-decreasing order with multiplicity $\leq$ K $+ 1$.

C (Type according to t) One-dimensional array of length $\geq$ M $-$ K $- 1$.
For tSPPS1 and tSPCD1: On entry, C(I) must contain the i-th coefficient c_i of the polynomial spline $s(x)$ in B-spline representation.
For tSPIN1, tSPAP1 and tSPVD1: On exit, C(I) contains the i-th coefficient c_i of the polynomial interpolation or approximation spline $s(x)$ in B-spline representation.

D (Type according to t) One-dimensional array of length $\geq$ M $-$ K $- 1$.
On exit, D(I) contains the coefficient d_i of the NDER-th derivative of a polynomial spline $s(x)$ in B-spline representation.

W (Type according to t) One-dimensional array of length $\geq (3 *$ K $+ 1) *$ N (tSPIN1), and of length $\geq$ NW (tSPAP1); used as working space.

NW (`INTEGER`) Length of working array W, (NW $\geq$ N $* (n_0{+}5){+}n_0{*}(n_0{+}1)$; $n_0 =$ M $-$ K $- 1$).

IW (`INTEGER`) One-dimensional array of length $\geq$ N, used as working space.

NERR (INTEGER) Error indicator. On exit:

$= 0$: No error or warning detected.

$= 1$: At least one of the parameters I, K, M, N, NDER is not in range or $A \le B$ is not true.

$= 2$: The subprograms tGEQPF, tORMQR, tTRTRS in the Linear Algebra package LAPACK (F001) were unable to solve the linear system of equations for calculating the coefficients of the spline interpolation to a given data set.

Case (2D):

F Name of a user-upplied FUNCTION subprogram, declared EXTERNAL in the calling program. This subprogram must provide the value of the function $z = f(x, y)$ for variation diminishing spline approximation.

KX,KY (INTEGER) Degree of one-dimensional B-splines in x- (y-)direction ($1 \le KX \le 25$, $1 \le KY \le 25$ for tSPVD2; $0 \le KX \le 25$, $0 \le KY \le 25$ otherwise).

MX,MY (INTEGER) Number of knots in x- (y-)direction ($MX \ge 2 * KX + 2$, $MY \ge 2 * KY + 2$).

I,J (INTEGER) Indices of B-splines ($1 \le I \le MX - KX - 1$, $1 \le J \le MY - KY - 1$).

NX,NY (INTEGER) Number of data points x_{lx} (y_{ly}) in x- (y-)direction ($NX \ge KX + 1$, $NY \ge KY + 1$ for spline interpolation tSPIN2; $NX \ge MX - KX - 1$, $NY \ge MY - KY - 1$ for spline approximation tSPAP2).

NDERX, NDERY (INTEGER) Selects one out of three cases: (i) integral, (ii) function value, or (iii) derivative. ($-1 \le NDERX \le KX$, $-1 \le NDERY \le KY$ for tSPNB2 and tSPPS2; $1 \le NDERX \le KX$, $1 \le NDERY \le KY$ for tSPCD2).

$= -1$: Calculation of the integral of $B_{i,j}(x, y)$ (tSPNB2), or the integral of $s(x, y)$ (tSPPS2).

$= 0$: Calculation of the function value $B_{i.j}(x, y)$ (tSPNB2), or the function value $s(x, y)$ (tSPPS2).

≥ 1 : Calculation of the NDERX-th partial derivative with respect to x and the NDERY-th partial derivative with respect to y of $B_{i,j}(x, y)$ (tSPNB2), or the calcultion of these derivatives of $s(x, y)$ (tSPPS2).

Note that in the first two cases $NDERX = NDERY = -1$, $NDERX = NDERY = 0$, respectively.

X,Y (Type according to t) Independent variables x, y of polynomial spline $s(x, y)$ or B-spline $B_{i,j}(x, y)$.

XI,YI (Type according to t) One-dimensional arrays of length $\ge$ NX and $\ge$ NY, respectively. On entry, XI(LX) and YI(LY) must contain the lx-th data point x_{lx} and the ly-th data point y_{ly} for spline interpolation (tSPIN2) or spline approximation (tSPAP2). The data points must be in ascending order.

ZI (Type according to t) Two-dimensional array of dimension (NDIMZ, $\ge$ NY). On entry, ZI(LX, LY) must contain the (lx, ly)-th data point $z_{lx,ly}$ for spline interpolation (tSPIN2) or spline approximation (tSPAP2).

NDIMZ (INTEGER) Declared first dimension of a two-dimensional array ZI in the calling program ($\ge$ NX).

KNOT (INTEGER) Controls the mode of supplying the knots for spline interpolation or approximation.

$= 1, 2$: The set of knots are computed by subprograms tSPIN2 and tSPAP2. At the left and right end-points of the interpolation (approximation) intervals $[x_1, x_{nx}]$, $[y_1, y_{ny}]$ arise multiple knots. The remaining knots are either equidistant ($KNOT = 1$) or are computed by using the data points $\{x_{lx}, y_{ly}\}$ of interpolation (approximation) ($KNOT = 2$).

$\ne 1, 2$: The knots must be supplied by the user.

AX,BX; AY,BY (Type according to t) On entry, AX, BX; AY, BY must contain the left (right) end-points of the intervals $[a_x, b_x]$; $[a_y, b_y]$ for the calculation of a set of spline knots in x- (y-)direction, respectively, by tSPKN2.

TX,TY (Type according to t) One-dimensional arrays of length $\geq$ MX and $\geq$ MY, repectively.
For tSPKN2 and for tSPIN2, tSPAP2 if KNOT $= 1, 2$: On exit, TX(J) and TY(J) contain the j-th knot in x- (y-)direction. In the other cases, on entry, TX(J) and TY(J) must contain the j-th knot in x- (y-)direction. The knots must be in non-decreasing order with multiplicity $\leq$ KX + 1 and $\leq$ KY + 1, respectively.

C (Type according to t) Two-dimensional array of dimension (NDIMC, $\geq$ MY − KY − 1).
For tSPPS2, tSPCD2: On entry, C(I,J) must contain the (i, j)-th coefficient $c_{i,j}$ of the polynomial spline $s(x, y)$ in B-spline representation.
For tSPIN2, tSPAP2, tSPVD2: On exit, C(I,J) contains the (i, j)-th coefficient $c_{i,j}$ of the polynomial interpolation or approximation spline $s(x, y)$ in B-spline representation.

NDIMC (INTEGER) Declared first dimension of a two-dimensional array C in the calling program ($\geq$ MX − KX − 1).

D (Type according to t) Two-dimensional array of dimension (NDIMC, $\geq$ MY − KY − 1).
On exit, D(I,J) contains the coefficient $d_{i,j}$ of the partial derivative of order nx, ny with respect to x and y of a polynomial spline $s(x, y)$ in B-spline representation.

W (Type according to t) One-dimensional array of length $\geq$ MY − KY − 1 (tSPPS2), $\geq$ (3 ∗ KX ∗ NY + 2) ∗ NX ∗ NY (tSPIN2), and of length $\geq$ NW (tSPAP2), used as working space.

NW (INTEGER) Length of a one-dimensional array W, used as working space ($\geq$ NX ∗ NY ∗ (n_0+6)+n_0∗(n_0+1); n_0= (MX − KX − 1) ∗ (MY − KY − 1)).

IW (INTEGER) One-dimensional array of length $\geq$ NX ∗ NY, used as working space.

NERR (INTEGER) Error indicator. On exit:

= 0 : No error or warning detected.

= 1 : At least one of the parameters I, J, KX, KY, MX, MY, NX, NY, NDERX, NDERY is not in range or at least one of the relations AX $\leq$ BX, AY $\leq$ BY is not true.

= 2 : The routines tGEQPF, tORMQR, tTRTRS in the Linear Algebra package LAPACK (F001) were unable to solve the linear system of equations for calculation coefficients of spline interpolation to a given data set.

Examples:

Calculate

1. The coefficients c_i of a polynomial interpolation spline $y = s(x)$ of degree $k = 2$ in B-spline representation to a given data set $\{(x_l, y_l)\}_{l=1,\ldots,6}$;
2. The corresponding coefficients d_i of the first derivative $y' = \dfrac{ds(x)}{dx}$;
3. The values of $s(x)$, $\dfrac{ds(x)}{dx}$ and $\displaystyle\int_0^x s(\xi)d\xi$ for $x = 0(0.1)1$.

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION XI(6),YI(6),T(9),C(6),D(6),W(42),IW(6)
DATA K,N,NDER,KNOT / 2,6,1,1 /
DATA XI / 0D0,0.20D0,0.40D0,0.60D0,0.80D0,1.00D0 /
DATA YI / 1D0,0.66D0,0.47D0,0.38D0,0.35D0,0.34D0 /
M=N+K+1

CALL DSPIN1(K,N,XI,YI,KNOT,T,C,W,IW,NERR)
CALL DSPCD1(K,M,NDER,T,C,D,NERR)
```

```
      WRITE(6,1000) K,N,(T(I),I=1,M)
      DO 20 J=0,10
      X=J/1D1

      SPL0=DSPPS1(K,M, 0,X,T,C,NERR)
      SPL1=DSPPS1(K,M, 1,X,T,D,NERR)
      SPLI=DSPPS1(K,M,-1,X,T,C,NERR)

   20 WRITE(6,1010) J,X,SPL0,SPL1,SPLI
 1000 FORMAT(...)
 1010 FORMAT(...)
      END
```

```
DEGREE OF POLYNOMIAL SPLINE:  2          NUMBER OF DATA POINTS:  6
KNOTS T :        0.00  0.00  0.00  0.25  0.50  0.75  1.00  1.00  1.00

    J       X        S(X)         DS(X)         IN(X)
    0      0.00     1.000000     -2.119921      0.000000
    1      0.10     0.809004     -1.700000      0.090100
    2      0.20     0.660000     -1.280079      0.163201
    3      0.30     0.550992     -0.940017      0.223467
    4      0.40     0.470000     -0.679816      0.274299
    5      0.50     0.415028     -0.419615      0.318334
    6      0.60     0.380000     -0.280953      0.357970
    7      0.70     0.358838     -0.142290      0.394796
    8      0.80     0.350000     -0.065306      0.430174
    9      0.90     0.344235     -0.050000      0.464873
   10      1.00     0.340000     -0.034694      0.499072
```

Error handling:

Error E210.1: K|KX,KY not in range,
Error E210.2: M|MX,MY not in range,
Error E210.3: I|I,J not in range,
Error E210.4: N|NX,NY not in range,
Error E210.5: NDER|NDERX,NDERY not in range,
Error E210.6: A,B|AX,BX;AY,BY inconsistent,
Error E210.7: NDERX|NDERY inconsistent.

In all cases, NERR is set = 1 (see above). A message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. J.H. Ahlberg, E.N. Nilson, J.L. Walsh, The Theory of Splines and their Applications, Academic Press, New York, 1967.

2. M.J. Marsden, An identity for spline functions with applications to variation diminishing spline approximation, J. Appr. Theory 3 (1970), 7-49.

3. C. de Boor, On calculating with B-splines, J. Appr. Theory 6 (1972), 50-62.

4. M.G. Cox, The numerical evaluation of B-splines, J. Inst. Maths Applics 10 (1972), 134-149.

5. J.G. Hayes, J. Halliday, The least-squares fitting of cubic spline surfaces to general data sets, J. Inst. Maths Applics 14 (1974), 89-103.

6. M.G. Cox, An algorithm for spline interpolation, J. Inst. Maths Applics 15 (1975), 95-108.

7. C. de Boor, A Practical Guide to Splines, Springer-Verlag, Berlin (1978).

8. P. Lancester, K. Salkauskas, Curve and Surface Fitting - An Introduction, Academic Press, New York, 1986.

9. J.C. Mason, M.S. Cox (Eds.), Algorithms for Approximation, Clarendon Press, Oxford, 1987.

10. J.W. Schmidt, H. Späth (Eds.), Splines in Numerical Analysis, Akademie-Verlag, Berlin, 1989.

11. H. Späth, Eindimensionale Spline-Interpolations-Algorithmen; Zweidimensionale Spline-Interpolations-Algorithmen, (2 Bd.) R. Oldenbourg, München 1990/1991.

•

Author(s) : K.S. Kölbig, H. Lipps **Library:** MATHLIB
Submitter : K.S. Kölbig **Submitted:** 01.05.1990
Language : Fortran **Revised:**

Cubic Splines and their Integrals

Subroutines `RCSPLN` and `DCSPLN` compute a (vector-valued) cubic spline function $F(x)$ which interpolates between a given set of points. Entries `RCSPNT` and `DCSPNT` compute the first and second integral over $F(x)$. On computers other than CDC or Cray, only the double-precision versions `DCSPLN` and `DCSPNT` are available. On CDC and Cray computers, only the single-precision versions `RCSPLN` and `RCSPNT` are available.

Given an interval $[a, b]$, a subdivision of this interval into $n \geq 2$ subintervals

$$a = x_0 < x_1 < ... < x_{n-1} < x_n = b,$$

and $n+1$ function values $Y_k = \{y_{k1}, \ldots, y_{km}\}$ on the $n+1$ abscissae (called 'knots') x_k $(k = 0, 1, \ldots, n)$, `RCSPLN` and `DCSPLN` compute a function $F(x)$ of class C^2, defined on $[a, b]$, which assumes the given value Y_k at the knot x_k (i.e. $F(x_k) = Y_k$), and which, when restricted to the ith sub-interval $[x_{i-1}, x_i]$ is identical with a set of m polynomials $\{p_{i1}, ..., p_{im}\}$, each of degree at most 3. Any function $F(x)$ which satisfies the above two conditions is called a 'cubic spline' through the $n + 1$ points (x_k, Y_k). To define the spline function $F(x)$ uniquely the subroutines impose an additional boundary condition, specified by their `MODE` parameter:

`MODE = 0`: $F''(x_0) = F''(x_n) = 0$ (the so-called *natural spline*).

`MODE = 1`: $F''(x_0) = F''(x_1)$ and $F''(x_{n-1}) = F''(x_n)$.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RCSPLN`, `RCSPNT`, `DCSPLN`, `DCSPNT`
Files referenced: `Unit 6`

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

Spline: `CALL tCSPLN(N,X,M,Y,NDIM,MODE,A,B,C,D)`
Integrals: `CALL tCSPNT(N,X,M,Y,NDIM,MODE,A,B,C,D)`

`N` (`INTEGER`) Number n of subintervals $[x_{i-1}, x_i]$. Must contain a value of at least 2 on entry. Unchanged on return.

`X` (type according to `t`) One-dimensional array of dimension `(0:d)` of at least $n+1$ elements. On entry, `X(k)` must contain the abscissa x_k, $(k = 0, 1, \ldots, n)$. Unchanged on return.

`M` (`INTEGER`) Number m of components of the vector-valued spline function $F(x)$. Must contain a value of at least 1 on entry. Unchanged on return.

`Y` (type according to `t`) Two-dimensional array of dimension `(0:NDIM,d)` where `d` is a value not less than m. On entry, `Y(k,j)` must contain the value y_{kj} of the jth component of the vector Y_k, $(k = 0, 1, \ldots, n; j = 1, \ldots, m)$. Unchanged on return.

`NDIM` (`INTEGER`) Upper bound of the first dimension of arrays `A`, `B`, `C`, `D` and `Y`. Must contain a value of at least n on entry. Unchanged on return.

`MODE` (`INTEGER`) Type of boundary condition (see description above). Must contain the value 0 or 1 on entry. Unchanged on return.

`A,B,C,D` (type according to `t`) Two-dimensional arrays of dimension `(NDIM,d)`, where `d` $\geq m$.
On return from `RCSPLN`, `A(i,j)`, `B(i,j)`, `C(i,j)` and `D(i,j)` will contain the four coefficients a_{ij}, b_{ij}, c_{ij}, and d_{ij} of the polynomial

$$p_{ij} = a_{ij} + b_{ij}(x - x_{i-1}) + c_{ij}(x - x_{i-1})^2 + d_{ij}(x - x_{i-1})^3$$

that determines the jth component $f_j(x)$ of the spline in the ith subinterval $[x_{i-1}, x_i]$, $i = l, \ldots, n$, $j = 1, \ldots, m$.
On return from `RCSPNT`,

$$\texttt{A(i,j) =} \int_a^{x_i} f_j(t)\,dt \quad \text{and} \quad \texttt{B(i,j) =} \int_a^{x_i} \int_a^{x} f_j(t)\,dt\,dx,$$

with $i = 1, \ldots, n$; $j = 1, \ldots, m$.
Arrays `C` and `D` have been used as working space.

Restrictions:

`N` $\geq$ `2`, `M` $\geq$ `1`, `NDIM` $\geq$ `N`, `MODE` $=$ `0` or `1`.

Error handling:

Error `E211.1`: `N` $<$ `2`.
Error `E211.2`: `M` $<$ `1`.
Error `E211.3`: `NDIM` $<$ `N`.
Error `E211.4`: `MODE` $\neq$ `0` and `MODE` $\neq$ `1`.
A message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.
•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 01.12.1994
Language : Fortran **Revised:**

Solution of Overdetermined Linear System in the Chebychev Norm

Subroutine subprograms `RCHEBN` and `DCHEBN` find the Chebyshev or minimax solution to a set of overdetermined linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, i.e. the vector **x** which minimizes

$$c = \max_{1 \le i \le m} c_i = \max_{1 \le i \le m} \left| b_i - \sum_{j=1}^{n} a_{ij} x_j \right| .$$

On computers other than CDC or Cray, only the double-precision version `DCHEBN` is available. On CDC and Cray computers, only the single-precision version `RCHEBN` is available.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RCHEBN`, `DCHEBN`
External References: `RVSCA` (F002), `RVSCL` (F002), `RVSCS` (F002), `RVSET` (F002), `RVXCH` (F002), `DVSCA` (F002), `DVSCL` (F002), `DVSCS` (F002), `DVSET` (F002), `DVXCH` (F002)

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
CALL tCHEBN(M,N,A,MDIM,B,TOL,RELERR,X,RESMAX,IRANK,ITER,ICODE)
```

`M` (`INTEGER`) Number m of equations.

`N` (`INTEGER`) Number n $(\le m)$ of unknowns.

`A` (type according to `t`) Two-dimensional array of dimension `(MDIM,d)`, where `d` $\ge n+3$. On entry, `A(I,J)` must contain the coefficients a_{ij} $(i = 1,\dots,m;\ j = 1,\dots,n)$ of matrix **A**. The contents of `A` is destroyed during execution.

`MDIM` (`INTEGER`) Declared first dimension of array `A`, where `MDIM` $\ge m+1$.

`B` (type according to `t`) One-dimensional array of length $\ge m+1$. On entry, the first m elements of `B` must contain the vector **b**. On exit, these elements contain the residuals c_i.

`TOL` Tolerance parameter which should be set to a value somewhat greater than the machine precision.

`RELERR` (type according to `t`) On entry, `RELERR` should be set to zero if the true minimax solution is required. (For `RELERR` non-zero see **Notes**).

`X` (type according to `t`) One-dimensional array of length $\ge n+3$. On exit, the first n elements of `X` contain the solution vector **x**.

`RESMAX` (type according to `t`) On exit, `RESMAX` contains the value c of the maximum residual.

`IRANK` (`INTEGER`) On exit, `IRANK` contains an estimate of the rank of the matrix `A`. (This estimate may depend on `TOL`).

`ITER` (`INTEGER`) On exit, `ITER` contains the number of simplex iterations performed.

`ICODE` (`INTEGER`) On exit, `ICODE` contains one of the following:
$=$ `0` : Solution **x** is not unique,
$=$ `1` : Solution **x** is unique,
$=$ `2` : Calculation terminated prematurely because of rounding error.

Method:

Modified simplex method of linear programming applied to the dual of the stated minimax problem.

Notes:

1. If RELERR on entry contains a non-zero positive value r, RELERR on exit contains a value $r' < r$, and the computed solution $\mathbf{x}'$ in X and the maximum residual c' in RESMAX are such that $(c' - c)/c < r'$, where c is the maximum residual corresponding to the true minimax solution $\mathbf{x}$. By setting RELERR non-zero (e.g. RELERR = 0.1), the number of simplex iterations is usually reduced.
2. If RESMAX is within one or two orders of magnitude of TOL, the computed residuals in B on exit may contain few significant digits, and may have been set to zero if RESMAX $<$ TOL.

Source:

The subprograms are based on a Fortran algorithm given in Ref. 1.

References:

1. I. Barrodale and C. Phillips, Algorithm 495: Solution of an overdetermined system of linear equations in the Chebyshev norm, ACM Trans. Math. Software **1** (1975) 264–270.

•

Author(s) : W. Hart, W. Matt
Submitter :
Language : Fortran

Library: KERNLIB
Submitted: 01.01.1975
Revised: 04.02.1986

Constrained and Unconstrained Linear Least Squares Fitting

The `TL` package finds the least squares solution to a set of unweighted linear equations, possibly subject to a set of equality constraints. The solution is found by Householder triangularisation (see Ref. 1 for details) with parameter elimination if constraints are present. This write-up ends with a few words on generalised least squares fitting (unequal weighting) which is a simple application of the `TL` package.

All matrices are assumed to be stored **row-wise and without gaps**, **contrary** to the Fortran convention, i.e., if the Fortran statement `DIMENSION A(NJ,NI)` reserves memory for the matrix **A** the element A_{ij} is found in word `A(J,I)`.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `TLSC`, `TLS`, `TLERR`, `TLRES`
Internal Entry Names: `TLSMSQ`, `TLSWOP`, `TLUK`, `TLSTEP`, `TLPIV`

Usage:

General Description

Consider the set of M linear equations

$$\sum_{j=1}^{N} A_{ij} x_j = b_i \qquad (i = 1, 2, \ldots, M \text{ with } N \leq M)$$

to be solved such that the Euclidian norm $||\mathbf{Ax} - \mathbf{b}||_2 = S^2$ is minimised. Instead of determining **x** from the Normal Equation $\mathbf{x} = (\mathbf{A'A})^{-1}\mathbf{A'b}$ it is found by applying successive Householder transformations (**Q**) which reduce **A** to upper triangular form without changing the norm of the columns of **A** or the vector **b**. This is beneficial from the point of view of stability and flexibility of application. Writing

$$\mathbf{QA} = \mathbf{R} = \begin{bmatrix} \mathbf{R_1} \\ \mathbf{O} \end{bmatrix} \begin{matrix} \} \ N \text{ rows} \\ \} \ M - N \text{ rows} \end{matrix} \qquad \text{and} \qquad \mathbf{Qb} = \mathbf{y} = \begin{bmatrix} \mathbf{y_1} \\ \mathbf{y_2} \end{bmatrix} \begin{matrix} \} \ N \text{ rows} \\ \} \ M - N \text{ rows} \end{matrix}$$

we have that $||\mathbf{Rx} - \mathbf{y}||_2 = ||\mathbf{Ax} - \mathbf{b}||_2$ and the vector **x** is obtained by backward substitution in $\mathbf{R_1x} = \mathbf{y_1}$. As a byproduct, the sum of squares of residuals is directly calculated as $S^2 = ||\mathbf{y_2}||_2$.

Now consider **A** and **b** to be composed of M_1 constraints to be satisfied exactly, followed by $M - M_1$ equations to be minimised. Writing

$$\mathbf{A} = \begin{bmatrix} \mathbf{A_1} \\ \mathbf{A_2} \end{bmatrix} \begin{matrix} \} \ M_1 \text{ rows} \\ \} \ M - M_1 \text{ rows} \end{matrix} \quad , \quad \mathbf{b} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \end{bmatrix} \begin{matrix} \} \ M_1 \text{ rows} \\ \} \ M - M_1 \text{ rows} \end{matrix}$$

then $||\mathbf{A_2x} - \mathbf{b_2}||_2 = S^2$ has to be minimized subject to $\mathbf{A_1x} - \mathbf{b_1} = 0$.

This problem is solved by eliminating M_1 parameters and then evaluating the reduced set of parameters (see Ref. 2 for details).

An attractive feature of the unitary Householder transformations is that when each parameter is eliminated ("solved for") column pivoting allows the selection of that parameter which gives the maximum reduction in the current value of S^2. Thus it is possible to terminate the calculation whenever S^2 or its current reduction become acceptably small. This can be exploited when iterating. If there is more than one RHS vector, then **x** and **b** become $N \times L$ and $M \times L$ matrices with the pivoting strategy applied to the first column of **b**.

The triangular form of $\mathbf{R_1}$ allows the error matrix, **E**, of the fitted parameters to be derived directly from $\mathbf{R_1}$ itself without inverting. The equation is

$$\mathbf{E} = \mathbf{R_1^{-1}}(\mathbf{R_1^{-1}})'.$$

Moreover, the vector of fitted residuals is most easily computed by applying the inverse Householder transformation to $\mathbf{y_2}$, i.e.

$$\mathbf{Ax} - \mathbf{b} = \mathbf{Q}^{-1}\begin{bmatrix}\mathbf{O}\\ \mathbf{y_2}\end{bmatrix}.$$

Note that these residuals do *not have to be calculated* to find the fitted S^2 which is output from the fitting routines.

In all routines described below, the dimensionality of the problem is transmitted via the common block

```
COMMON /TLSDIM/ M1,M,N,L,IER
```

The parameter `IER` returns the number of parameters solved for, or else `-1001` if either `M1` $>$ `N`, `N` $>$ `M` or **A** has rank less than `N`.

Constrained Least Squares Fitting

```
CALL TLSC(A,B,AUX,IPIV,EPS,X)
```

`A`	(`REAL`) The combined constraint / derivative matrix of dimension `M` $\times$ `N`, the upper `M1` rows being the constraints.
`B`	(`REAL`) The combined constraint / measurement matrix of dimension `M` $\times$ `L`, the upper `M1` rows being the constraints.
`X`	(`REAL`) The matrix of dimension `N` $\times$ `L` returning the `L` least squares solutions.
`AUX`	(`REAL`) Working array of length `N` $+ \max($`N`, `L`$)$. On output `AUX(J),(J=1,L)` contain the minimised sum of squares.
`IPIV`	(`INTEGER`) Working array of length `N` which holds the exchange information (column pivoting is employed if necessary).
`EPS`	(`REAL`) Parameter specifying a pivoting criterium. There is no exchange of columns I and 1 unless `EPS` $*$ `PIVOT(I)` $>$ `PIVOT(1)`. Typically `EPS` $\simeq$ `0.1`.

Subroutines called: `TLSMSQ`, `TLSWOP`, `TLUK`, `TLSTEP`.

When constraint equations are present, the full pivoting strategy cannot be adopted and so all parameters are solved for, i.e., `IER` returns the value `N` or `-1001`. Under these circumstances `EPS` is used to reduce the amount of pivoting to those cases where it is felt to be absolutely necessary.

Unconstrained Least Squares Fitting

```
CALL TLS(A,B,AUX,IPIV,EPS,X)
```

A (REAL) M $\times$ N derivative matrix.

B (REAL) M $\times$ L matrix of measurements.

X (REAL) N $\times$ L parameter solution matrix.

AUX (REAL) Working array as for TLSC.

IPIV (INTEGER) Working array as for TLSC.

EPS (REAL) Input parameter used for prematurely terminating the calculation:
$>$ 0 : Termination when r.m.s. residual $<$ |EPS|,
$<$ 0 : Termination when the reduction in the residual $<$ |EPS|,
$=$ 0 : Unconditionally solve for all X_j.

Subroutines called: TLSMSQ, TLSWOP, TLUK, TLSTEP, TLPIV.

As previously indicated, full pivoting is possible without constraints, hence the allowance for premature exit.

Fitted Error Matrix

```
CALL TLERR(A,E,AUX,IPIV)
```

The parameter and subroutine arguments defined previously in COMMON /TLSDIM/ require the output values from a call to TLS or TLSC. E is an $N \times N$ matrix which, upon return, will contain the unnormalised covariance matrix of the fitted parameters, $(\mathbf{A}'\mathbf{A})^{-1}$. A may be overwritten by E and the routine may be called independently from TLS/TLSC by setting IER to zero.

Subroutines called: TLUK, TLSTEP.

Fitted Residuals

```
CALL TLRES(A,B,AUX)
```

All the arguments and common variables require the output values from a call to TLS or TLSC. Upon return, B will give the matrix of residuals, i.e., for each set of least squares equations the column vector $\mathbf{Ax} - \mathbf{b}$.

Subroutine called: TLSTEP.

Notes:

1. The pivoting and exit criteria of TLS are calculated using the first vector of measurements; therefore it is wise to have EPS $=$ 0 if L $>$ 1.
2. TLERR and/or TLRES may be called in any order after TLS or TLSC.
3. TLS or TLSC may be used for solving simultaneous linear equations by setting M $=$ N or M1 $=$ N.
4. Useful examples in the application of these routines can be found in the HYDRA Geometry / Kinematics processors.

Generalized Least Squares Fitting

The problem is to minimise $(\mathbf{Ax} - \mathbf{b})'\mathbf{G}(\mathbf{Ax} - \mathbf{b})$ where **G**, the weight matrix, is the inverse of the error matrix of the measurement vector **b**. Once again Householder triangularisation offers an attractive alternative to the Normal Equation solution $\mathbf{x} = (\mathbf{A}'\mathbf{GA})^{-1}\mathbf{A}'\mathbf{Gb}$. The first step is to perform the Choleski decomposition of **G**, which is positive semi-definite (see `TR` (F112)), such that $\mathbf{G} = \mathbf{U}'\mathbf{U}$, **U** being upper triangular. The problem is then reduced to minimising $||\mathbf{A^1x} - \mathbf{b^1}||_2$, where $\mathbf{A^1} = \mathbf{UA}$ and $\mathbf{b^1} = \mathbf{Ub}$, which is just the unweighted case previously described. This has the feature that if **A** has already been triangularised then the product **UA** remains triangular and only back substitution is necessary to find the weighted least squares solution.

References:

1. G. Golub, Numerical methods for solving linear least squares problems, Numer. Math. **7** (1965) 206–216.

2. Å. Björck and G. Golub, Iterative refinement of linear least square solutions by Householder transformation, BIT **7** (1967) 322–337.

•

Author(s) : M. Metcalf
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.05.1977
Revised: 27.11.1984

Least-Squares Fit to Straight Line

Given a vector of values Y measured at the points X, `LFIT` and `LFITW` find the best least-squares fit to the linear relationship $Y = aX + b$. `LFIT` performs an unweighted fit and `LFITW` takes account of a given vector of weights. Both subroutines have an option for skipping missing points without shifting the points of the vector X.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `LFIT`, `LFITW`

Usage:

```
CALL LFIT(X,Y,L,KEY,A,B,VAR)      or
CALL LFITW(X,Y,W,L,KEY,A,B,VAR)
```

`X` (`REAL`) Vector of abscissae.

`Y` (`REAL`) Vector of values corresponding to points `X`.

`W` (`REAL`) Vector of weights (for `LFITW` only).

`L` (`INTEGER`) Length of vectors `X`, `Y` and `W`.

`KEY` (`INTEGER`)
$=$ `0` : indicates that any points where `Y` $=$ `0` are to be skipped,
$=$ `1` : indicates that all `L` points are to be used.

`A` (`REAL`) Fitted slope a.

`B` (`REAL`) Fitted constant term b.

`VAR` (`REAL`) Residual sum of squares divided by (`L` $-$ 2) indicating the badness of fit.

References:

1. D.H. Menzel, Fundamental Formulas of Physics, Dover Publ., New York (1960) 116.

•

Author(s) : H. Grote
Submitter : M. Metcalf
Language : Fortran

Library: MATHLIB
Submitted: 01.05.77
Revised:

Least-Squares Fit to Parabola

OBSOLETE
Please note that this routine has been obsoleted in CNL 218. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: RLSQP2 (E201)

Given a vector of values Y measured at the points X, PARLSQ finds the best least-squares fit to the parabola $Y = c_1 + c_2 x + c_3 x^2$.

Structure:

SUBROUTINE subprogram
User Entry Names: PARLSQ

Usage:

```
CALL PARLSQ(X,Y,L,C,VAR)
```

X	(REAL) Vector of abscissae.
Y	(REAL) Vector of values corresponding to points X.
L	(INTEGER) Length of vectors X and Y.
C	(REAL) Array of dimension 3 in the calling program. On exit, it contains the coefficients c_1, c_2, c_3.
VAR	(REAL) Residual sum of squares divided by L − 3.

Notes:

If L < 3, C and VAR are set to zero.

References:

1. D.H. Menzel, Fundamental Formulas of Physics, Dover Publ., New York (1960) 122

•

RCHECF	CERN Program Library	E406

Author(s) : T. Håvie
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 24.01.1986
Revised: 01.12.1994

Chebyshev Series Coefficients of a Function

Subroutine subprograms `RCHECF`, `DCHECF` and `QCHECF` calculate coefficients for a finite sum of Chebyshev polynomials approximating a function $f(x)$ over an interval $a \leq x \leq b$ to accuracy ε. It returns an integer n and coefficients $c_0, c_1, \ldots, c_n$ such that the sum

$$f^*(x) = \sum_{j=0}^{n} c_j T_j(t) \tag{1}$$

where $t = (2x - a - b)/(b - a)$ and $T_j(t)$ is the Chebyshev polynomial of degree j, satisfies for $a \leq x \leq b$ the relation

$$|f^*(x) - f(x)| < \varepsilon. \tag{2}$$

Subsequent evaluation of the approximation (1) can be done by calling `CHSUM` (E407) with the appropriate value of its argument `MODE`.

On computers other than CDC and Cray, only the double- and quadruple-precision versions `DCHECF` and `QCHECF` are available. On CDC and Cray computers, only the single- and double-precision versions `RCHECF` and `DCHECF` are available.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `RCHECF`, `DCHECF`, `QCHECF`
Obsolete User Entry Names: `CHECF` ≡ `RCHECF`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035), user-supplied `FUNCTION` subprogram

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`), `t` = `Q` (type `REAL*16`),

```
CALL tCHECF(F,A,B,EPS,C,N,DELTA)
```

`F` (type according to `t`) Name of a user-supplied `FUNCTION` subprogram, declared `EXTERNAL` in the calling program.

`A,B` (type according to `t`) End-points a, b of the approximation interval.

`EPS` (type according to `t`) Requested accuracy.

`C` (type according to `t`) One-dimensional array with dimension `(0:d)`, `d` $\geq$ `128`. On exit, `C(j)` $= c_j$, ($j = 0, 1, \ldots,$ `N`).

`N` (`INTEGER`) On exit, `N` is equal to the subscript of the last computed coefficient.

`DELTA` (type according to `t`) On exit, `DELTA` is such that the relation $|f^*(x) - f(x)| <$ `DELTA` is almost certainly true for $x \in [a, b]$. (See Error Handling.)

Method:

The interval $[a, b]$ is subdivided successively into sets of subintervals of length $2^{-k}(b-a)$, $(k = 0, 1, 2 \ldots)$. After each subdivision the orthogonality properties of the Chebyshev polynomials with respect to summation over equally-spaced points are used to compute two sets of approximate values of the coefficients c_j: one set computed using the end-points of the subintervals, and one set using the mid-points. The mean of these two values is taken as the best estimate of the c_j, which are then tested to see (a) whether certain rate-of-convergence criteria are satisfied, (b) whether there is some n for which the sum for $j > n$ of the available c_j is less than ε. If both conditions are satisfied the subroutine terminates.

Error handling:

Error `E406.1`: If the requested accuracy cannot be obtained with 65 coefficients (i.e., $\texttt{N} = \texttt{64}$) a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called. In this case, values of f^* computed from (1) with $\texttt{N} = \texttt{64}$ should still be in error by less than `DELTA`.

Notes:

1. This subroutine is intended for use with functions $f(x)$ which can be computed to full machine accuracy, and which are sufficiently smooth to ensure fairly rapid decrease of the c_j with increasing j. Functions defined by experimental data can usually be approximated better by least-squares methods, using ordinary polynomials.
2. Note that some authors use a different definition for the constant term in (1), i.e. $c_0/2$ instead of c_0. Here, the definition of Ref. 1 is used.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press, New York 1975)

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 24.01.1986
Language : Fortran **Revised:** 15.11.1995

Summation of Chebyshev Series

Function subprograms `RCHSUM` and `DCHSUM` compute, for real arguments x in the specified intervals, one of the following four sums:

$$S(x) = \sum_{n=0}^{N} c_n T_n(x) \qquad (-1 \leq x \leq 1) \qquad (1)$$
$$S(x) = \sum_{n=0}^{N} c_n T_{2n}(x) \qquad (-1 \leq x \leq 1) \qquad (2)$$
$$S(x) = \sum_{n=0}^{N} c_n T_{2n+1}(x) \qquad (-1 \leq x \leq 1) \qquad (3)$$
$$S(x) = \sum_{n=0}^{N} c_n T_n^*(x) \qquad (0 \leq x \leq 1) \qquad (4)$$

where $T_n(x)$ is the Chebyshev polynomial of degree n and $T_n^*(x) = T_n(2x-1)$.
On CDC and Cray computers, the double-precision version `DCHSUM` is not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RCHSUM`, `DCHSUM`
Obsolete User Entry Names: `CHSUM` $\equiv$ `RCHSUM`

Usage:

In any arithmetic expression,

`RCHSUM(MODE,C,N,X)` or `DCHSUM(MODE,C,N,X)`

has the value of the sum selected by `MODE`. `RCHSUM` is of type `REAL`, and `DCHSUM` is of type `DOUBLE PRECISION`. `C` and `X` have the same type as the function name. `MODE` and `N` are of type `INTEGER`.

`MODE` Type of sum to be evaluated (`MODE` $=$ `1`, `2`, `3`, `4`).

`C` One-dimensional array with dimension `(0:d)`, `d` $\geq$ `N`, containing the coefficients $c_0, c_1, \ldots, c_N$.

`N` Limit N of summation.

`X` Argument x.

Notes:

Note that some authors use a different definition for the constant term in (1), (2) and (4), i.e. $c_0/2$ instead of c_0. Here, the definition of Ref. 1 is used.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press, New York 1975)
2. C.W. Clenshaw, Chebyshev series for mathematical functions, Mathematical Tables, Vol.5 (National Physical Laboratory, London, 1962).

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.02.1994
Language : Fortran **Revised:**

Conversion of Chebyshev to Power and Power to Chebyshev Series

Subroutine subprograms `RCHPWS`, `RPWCHS` and `DCHPWS`, `DPWCHS` perform the conversion of a finite Chebyshev series to a finite power series (i.e. a polynomial) and *vice versa*.

Thus, given the coefficients c_j, $(j = 0, 1, \ldots, n)$ of a finite Chebyshev series, `RCHPWS` and `DCHWPS` calculate the coefficients a_j, $(j = 0, 1, \ldots, n)$ of the equivalent polynomial:

$$c_0 + c_1 T_1(x) + \cdots + c_n T_n(x) \;=\; a_0 + a_1 x + \cdots + a_n x^n.$$

Conversely, given the coefficients a_j, $(j = 0, 1, \ldots, n)$ of a power series, `RPWCHS` and `DPWCHS` calculate the coefficients c_j, $(j = 0, 1, \ldots, n)$ of the equivalent finite Chebyshev series:

$$a_0 + a_1 x + \cdots + a_n x^n \;=\; c_0 + c_1 T_1(x) + \cdots + c_n T_n(x).$$

In both cases, $T_j(x)$ is the Chebyshev polynomial of degree j.

Note that sometimes the constant term in the Chebyshev series is defined differently, i.e. $c_0/2$ instead of c_0. Here, the definition of Ref. 1 is used.

On computers other than CDC or Cray, only the double-precision versions `DCHPWS` and `DPWCHS` are available. On CDC and Cray computers, only the single-precision versions `RCHPWS` and `RPWCHS` are available.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RCHPWS`, `RPWCHS`, `DCHPWS`, `DPWCHS`
Files referenced: `Unit 6`

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
CALL tCHPWS(N,C,A)
```

`N` (`INTEGER`) Degree n of last Chebyshev polynomial in the expansion.

`C` (type according to `t`) One-dimensional array of dimension `(0:d)`, where `d` $\geq$ `N`. On entry, `C` must contain the coefficients c_j, $(j = 0, 1, \ldots, n)$ of the Chebyshev expansion.

`A` (type according to `t`) One-dimensional array of dimension `(0:d)`, where `d` $\geq$ `N`. On exit, `A` contains the coefficients a_j, $(j = 0, 1, \ldots, n)$ of the power series expansion.

```
CALL tPWCHS(N,A,C)
```

`N` (`INTEGER`) Degree n of the polynomial.

`A` (type according to `t`) One-dimensional array of dimension `(0:d)`, where `0` $\geq$ `N`. On entry, `A` must contain the coefficients a_j, $(j = 0, 1, \ldots, n)$ of the polynomial.

`C` (type according to `t`) One-dimensional array of dimension `(0:d)`, where `0` $\geq$ `N`. On exit, `C` contains the coefficients c_j, $(j = 0, 1, \ldots, n)$ of the Chebyshev expansion.

Error handling:

Error E408.1: $\mathtt{N} < 0$ or $\mathtt{N} > 100$.
A message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. Y.L. Luke, Mathematical functions and their approximations, (Academic Press, New York 1975)

•

Author(s) : T. Håvie, K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.12.1994
Revised:

Summation of Trigonometric Series

Function subprograms RTRGSM and DTRGSM compute the sum of the trigonometric series

$$f(x) \;=\; a_0 + \sum_{k=1}^{n} a_k \cos kx + \sum_{k=1}^{m} b_k \sin kx$$

for a given argument x in the range $-\pi \le x \le \pi$ and given coefficients a_k, b_k.
On CDC and Cray computers, the double-precision version DTRGSM is not available.

Structure:

FUNCTION subprogram
User Entry Names: RTRGSM, DTRGSM

Usage:

In any arithmetic expression, for t = R (type REAL), t = D (type DOUBLE PRECISION),

```
tTRGSM(X,A,N,B,M,IOP)
```

has the value $f(x)$.

- X — (Type according to t) Argument x.
- A — (Type according to t) One-dimensional array of dimension (0:d) where d $\geq$ N, containing the constant coefficient a_0 in A(0) and the cosine coefficients a_k $(k = 1, \ldots, n)$ in A(k).
- N — (INTEGER) The number n of cosine coefficients.
- B — (Type according to t) One-dimensional array of length $\geq$ M, containing the sine coefficients b_k $(k = 1, \ldots, n)$ in B(k).
- M — (INTEGER) The number m of sine coefficients.
- IOP — (INTEGER) An option number:
  = 1 : the general case,
  = 2 : all b_k are zero, i.e. $f(x) = f(-x)$,
  = 3 : all a_k are zero, i.e. $f(x) = -f(-x)$.

Method:

Standard recurrence relations are used for calculating the sum (see Ref. 1).

Notes:

For a function $f(z)$ given in the range $a \le z \le b$, use the transformation

$$x \;=\; \frac{2\pi}{b-a}\left(z - \frac{b+a}{2}\right) \qquad \text{for IOP} = 1,$$

$$x \;=\; \pi\,\frac{z-a}{b-a} \qquad \text{for IOP} = 2 \text{ or IOP} = 3.$$

References:

1. W. Clenshaw, A note on the summation of Chebyshev series, MTAC (later renamed Math. Comp.) **9** (1955) 118–120.

•

Author(s) : see below **Library:** MATHLIB
Submitter : B. Damgaard **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Linear Algebra Package

Authors: E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen.

LAPACK is a package of subroutines written in Fortran for solving the most common problems in numerical linear algebra: systems of linear equations, linear least squares problems, eigenvalue problems, and singular value problems. LAPACK is intended to supersede LINPACK and EISPACK. It extends the functionality of these packages by including equilibration, iterative refinement, error bounds, and driver routines for linear systems, routines for computing and re-ordering the Schur factorization, and condition estimation routines for eigenvalue problems. LAPACK improves on the accuracy of the standard algorithms in EISPACK by including high accuracy algorithms for finding singular values and eigenvalues of bidiagonal and tridiagonal matrices respectively that arise in SVD and symmetric eigenvalue problems. The algorithms and software are structured to achieve high efficiency on vector processors, high-performance "superscalar" workstations, and shared-memory multi-processors.

Structure:

SUBROUTINE subprograms

Usage:

It is highly recommended to obtain a copy of the LAPACK Users' Guide published by SIAM. This Users' Guide gives a detailed description of the philosophy behind LAPACK as well as an explanation of its usage. European users must order from the distributors of SIAM books in Europe:

> STM Distribution Ltd.
> Sunbury International Business Centre
> Middlesex TW16 7DX, England
> Tel. +44 932 765119, FAX +44 932 765429

or from booksellers. Other users should contact SIAM directly in order to find out the address of the local retailer:

> SIAM
> 3600 University City Science Center
> Philadelphia, PA 19104-2688
> Tel. +1 215 382 9800, FAX +1 215 386 7999 .

Availability
CERN is distributing the package only in compiled form, suited for the CERN-supported platforms. Source code is directly available via `netlib` (use `find netlib` for details). Alternatively, NAG offers the distribution via magnetic tapes for a nominal handling charge. NAG can be contacted at

> NAG Response Centre
> Tel. +44 865 311744, FAX +44 865 311755

•

Author(s) : H. Lipps **Library:** KERNLIB
Submitter : **Submitted:** 18.12.1979
Language : Fortran or Assembler or COMPASS **Revised:** 27.05.1987

Elementary Vector Processing

These subprograms perform elementary vector operations.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: RVADD, RVCPY, RVDIV, RVMPA, RVMPY, RVMUL, RVMULA, RVMUNA, RVRAN, RVSCA, RVSCL, RVSCS, RVSET, RVSUB, RVSUM, RVXCH, DVADD, DVCPY, DVDIV, DVMPA, DVMPY, DVMUL, DVMULA, DVMUNA, DVRAN, DVSCA, DVSCL, DVSCS, DVSET, DVSUB, DVSUM, DVXCH, CVADD, CVCPY, CVDIV, CVMPA, CVMPY, CVMUL, CVMULA, CVMUNA, CVRAN, CVSCA, CVSCL, CVSCS, CVSET, CVSUB, CVSUM, CVXCH, CVMPYC, CVMPAC
External References: `LOCF` (N100), `RANF` (G900), `DRANF` (G900) (some Fortran versions only).

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`), `t` = `C` (type `COMPLEX`):

`CALL tVSET (N,S,Z1,Z2)`	$z_j = s$
`CALL tVRAN (N,A,B,Z1,Z2)`	$z_j =$ random (see **Note** 2)
`CALL tVCPY (N,X1,X2,Z1,Z2)`	$z_j = x_j$
`CALL tVXCH (N,X1,X2,Y1,Y2)`	interchanges x_j with y_j
`CALL tVADD (N,X1,X2,Y1,Y2,Z1,Z2)`	$z_j = x_j + y_j$
`CALL tVSUB (N,X1,X2,Y1,Y2,Z1,Z2)`	$z_j = x_j - y_j$
`CALL tVMUL (N,X1,X2,Y1,Y2,Z1,Z2)`	$z_j = x_j y_j$
`CALL tVMULA(N,X1,X2,Y1,Y2,Z1,Z2)`	$z_j = x_j y_j + z_j$
`CALL tVMUNA(N,X1,X2,Y1,Y2,Z1,Z2)`	$z_j = -x_j y_j + z_j$
`CALL tVDIV (N,X1,X2,Y1,Y2,Z1,Z2,IFAIL)`	$z_j = x_j / y_j$ (see **Note** 3)
`CALL tVSCL (N,S,X1,X2,Z1,Z2)`	$z_j = s x_j$
`CALL tVSCA (N,S,X1,X2,Y1,Y2,Z1,Z2)`	$z_j = s x_j + y_j$
`CALL tVSCS (N,S,X1,X2,Y1,Y2,Z1,Z2)`	$z_j = s x_j - y_j$
`F = tVSUM (N,X1,X2)`	$f = x_1 + \cdots + x_n$
`F = tVMPY (N,X1,X2,Y1,Y2)`	$f = x_1 y_1 + \cdots + x_n y_n$
`F = tVMPA (N,X1,X2,Y1,Y2,S)`	$f = x_1 y_1 + \cdots + x_n y_n + s$
`F = CVMPYC(N,X1,X2,Y1,Y2)`	$f = x_1 \bar{y}_1 + \cdots + x_n \bar{y}_n$
`F = CVMPAC(N,X1,X2,Y1,Y2,S)`	$f = x_1 \bar{y}_1 + \cdots + x_n \bar{y}_n + s$

where $\bar{y}_j$ is the complex conjugate of y_j.

N	(INTEGER) The mathematical dimension of the vectors ($j = 1, 2, \ldots, \mathtt{N}$).
S,A,B	(Type according to t) The scalar values s, a, and b, respectively.
X1,X2	(Type according to t) Array elements. They must contain the elements x_1, x_2 of the vector (x_j).
Y1,Y2	(Type according to t) Array elements. They must contain the elements y_1, y_2 of the vector (y_j).
Z1,Z2	(Type according to t) Array elements. On exit, they will contain the elements z_1, z_2 of the result vector (z_j).
IFAIL	(INTEGER) On exit, IFAIL is set to zero if all elements y_j are non-zero. Otherwise IFAIL is set to the smallest index k for which $y_k = 0$.

For N < 1 all subroutines return control without action; functions tVSUM, tVMPY and CVMPYC assume the value zero, and tVMPA and CVMPAC assume the value S.

Restrictions:

If vector (z_j) overlaps with vector (x_j) or (y_j), results will be correct provided each element z_j coincides with an element x_k or y_k, where $k < j$.

Accuracy:

On computers with IBM 370 architecture, RVMPY, RVMPA, CVMPY and CVMPA accumulate the inner product using double-precision arithmetic internally; the final result is then rounded to single precision.

Notes:

1. The vectors (x_j) etc. need not be packed: any equidistant spacing of their elements is permitted. The subprograms determine the location of the vector element x_j from the actual arguments X1 and X2.
2. tVRAN sets z_j to a random value of type t that is uniformly distributed in the interval (A,B). For CVRAN, the real and imaginary parts of z_j are distributed uniformly and independently in (REAL(A),REAL(B)) and in (AIMAG(A),AIMAG(B)).
3. If $y_k = 0$ and $y_1, \ldots, y_{k-1}$ are non-zero, tVDIV computes only $z_1, \ldots, z_{k-1}$ and sets IFAIL $= k$.
4. The use of an in-line DO loop will be more efficient than calling the equivalent vector processing subprogram when the vector length is sufficiently small, due to the overhead of the subprogram call.

•

Author(s) : H. Lipps
Submitter :
Language : Fortran or Assembler or COMPASS

Library: KERNLIB
Submitted: 18.12.1979
Revised: 15.11.1995

Elementary Matrix Processing

These subprograms perform elementary matrix operations.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms

User Entry Names: `RMADD`, `RMBIL`, `RMCPY`, `RMDMP`, `RMMNA`, `RMMNS`, `RMMPA`, `RMMPS`, `RMMPY`, `RMRAN`, `RMSCL`, `RMSET`, `RMSUB`, `RMUTL`, `RUMNA`, `RUMNS`, `RUMPA`, `RUMPS`, `RUMPY`,
`DMADD`, `DMBIL`, `DMCPY`, `DMDMP`, `DMMNA`, `DMMNS`, `DMMPA`, `DMMPS`, `DMMPY`, `DMRAN`, `DMSCL`, `DMSET`, `DMSUB`, `DMUTL`, `DUMNA`, `DUMNS`, `DUMPA`, `DUMPS`, `DUMPY`,
`CMADD`, `CMBIL`, `CMCPY`, `CMDMP`, `CMMNA`, `CMMNS`, `CMMPA`, `CMMPS`, `CMMPY`, `CMRAN`, `CMSCL`, `CMSET`, `CMSUB`, `CMUTL`, `CUMNA`, `CUMNS`, `CUMPA`, `CUMPS`, `CUMPY`, `CMMPYC`, `CCMMPY`, `CUMPYC`, `CCUMPY`

External References: `LOCF` (N100), `RANF` (G900), `DRANF` (G900) (some Fortran versions only).

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`), `t` = `C` (type `COMPLEX`):

`CALL tMSET (M,N,S,Z11,Z12,Z21)`	$z_{ij} = s$
`CALL tMRAN (M,N,A,B,Z11,Z12,Z21)`	$z_{ij} =$ random (see **Note** 2)
`CALL tMCPY (M,N,X11,X12,X21,Z11,Z12,Z21)`	$z_{ij} = x_{ij}$
`CALL tMUTL (N,X11,X12,X21)`	$x_{jk} = x_{kj}$ $(j > k)$ (see **Note** 3)
`CALL tMSCL (M,N,S,X11,X12,X21,Z11,Z12,Z21)`	$z_{ij} = sx_{ij}$
`CALL tMDMP (M,N,D1,D2,X11,X12,X21,Z11,Z12,Z21)`	$z_{ij} = d_i x_{ij}$
`CALL tMADD (M,N,X11,X12,X21,Y11,Y12,Y21,Z11,Z12)`	$z_{ij} = x_{ij} + y_{ij}$
`CALL tMSUB (M,N,X11,X12,X21,Y11,Y12,Y21,Z11,Z12)`	$z_{ij} = x_{ij} - y_{ij}$
`CALL tMMPY (M,N,X11,X12,X21,Y1,Y2,Z1,Z2)`	$z_i = x_{i1}y_1 + \cdots + x_{in}y_n$
`CALL tMMPA (M,N,X11,X12,X21,Y1,Y2,Z1,Z2)`	$z_i = x_{i1}y_1 + \cdots + x_{in}y_n + z_i$
`CALL tMMPS (M,N,X11,X12,X21,Y1,Y2,Z1,Z2)`	$z_i = x_{i1}y_1 + \cdots + x_{in}y_n - z_i$
`CALL tMMNA (M,N,X11,X12,X21,Y1,Y2,Z1,Z2)`	$z_i = -x_{i1}y_1 - \cdots - x_{in}y_n + z_i$
`CALL tMMNS (M,N,X11,X12,X21,Y1,Y2,Z1,Z2)`	$z_i = -x_{i1}y_1 - \cdots - x_{in}y_n - z_i$
`CALL tUMPY (N,U11,U12,U22,Y1,Y2,Z1,Z2)`	$z_j = u_{jj}y_j + \cdots + u_{jn}y_n$
`CALL tUMPA (N,U11,U12,U22,Y1,Y2,Z1,Z2)`	$z_j = u_{jj}y_j + \cdots + u_{jn}y_n + z_j$
`CALL tUMPS (N,U11,U12,U22,Y1,Y2,Z1,Z2)`	$z_j = u_{jj}y_j + \cdots + u_{jn}y_n - z_j$
`CALL tUMNA (N,U11,U12,U22,Y1,Y2,Z1,Z2)`	$z_j = -u_{jj}y_j - \cdots - u_{jn}y_n + z_j$
`CALL tUMNS (N,U11,U12,U22,Y1,Y2,Z1,Z2)`	$z_j = -u_{jj}y_j - \cdots - u_{jn}y_n - z_j$
`F = tMBIL (N,V1,V2,X11,X12,X21,Y1,Y2)`	$f = \sum_{k,j=1}^{n} v_k x_{kj} y_j$
`CALL CMMPYC(M,N,X11,X12,X21,Y1,Y2,Z1,Z2)`	$z_i = x_{i1}\bar{y}_i + \cdots + x_{in}\bar{y}_n$
`CALL CCMMPY(M,N,X11,X12,X21,Y1,Y2,Z1,Z2)`	$z_i = \bar{x}_{i1}y_i + \cdots + \bar{x}_{in}y_n$
`CALL CUMPYC(N,U11,U12,U22,Y1,Y2,Z1,Z2)`	$z_j = u_{jj}\bar{y}_j + \cdots + u_{jn}\bar{y}_n$
`CALL CCUMPY(N,U11,U12,U22,Y1,Y2,Z1,Z2)`	$z_j = \bar{u}_{jj}y_j + \cdots + \bar{u}_{jn}y_n$

where $\bar{x}_{ij}$, $\bar{u}_{jk}$, $\bar{y}_j$ are the complex conjugates of x_{ij}, u_{jk}, y_j, respectively.

`M,N`	(`INTEGER`) The mathematical dimensions of the matrices and vectors ($i = 1, 2, \ldots,$ `M`; $j, k = 1, 2, \ldots,$ `N`).
`S,A,B`	(Type according to `t`) The scalar values s, a, and b, respectively.
`X11,X12,X21`	(Type according to `t`) Array elements. They must contain the elements x_{11}, x_{12}, x_{21} of the matrix (x_{ij}).
`Y11,Y12,Y21`	(Type according to `t`) Array elements. They must contain the elements y_{11}, y_{12}, y_{21} of the matrix (y_{ij}).
`Y1,Y2`	(Type according to `t`) Array elements. They must contain the elements y_1, y_2 of the vector (y_j).
`D1,D2`	(Type according to `t`) Array elements. They must contain the elements d_1, d_2 of the vector (d_i).
`V1,V2`	(Type according to `t`) Array elements. They must contain the elements v_1, v_2 of the vector (v_k).
`U11,U12,U22`	(Type according to `t`) Array elements. They must contain the elements u_{11}, u_{12}, u_{22} of the upper-triangular matrix (u_{jk}).
`Z11,Z12,Z21`	(Type according to `t`) Array elements. On exit, they will contain the elements z_{11}, z_{12}, z_{21} of the result matrix (z_{ij}).
`Z1,Z2`	(Type according to `t`) Array elements. On exit, they will contain the elements z_1, z_2 of the result vector (z_j).

For `M` $<$ `1` or `N` $<$ `1` all subroutines return control without action and all functions assume the value zero.

Accuracy:

On computers with IBM 370 architecture, all routines that accumulate the inner product of type `REAL` or `COMPLEX` use double-precision arithmetic internally; the final result is then rounded to single precision.

Notes:

1. The vectors (y_j) etc. need not be packed: any equidistant spacing of their elements is permitted. The subprograms determine the location of the vector element y_j from the actual arguments `Y1` and `Y2`. Similarly, the matrices (x_{ij}) etc. need not be stored according to the Fortran convention; any equidistant spacing of their rows and columns is permitted. In particular, matrices may be stored row-wise. The subprograms determine the location of the matrix element x_{ij} from the actual arguments `X11`, `X12`, and `X21`.

2. `tMRAN` sets z_{ij} to a random value of type `t` that is uniformly distributed in the interval `(A,B)`. For `CMRAN`, the real and imaginary parts of z_{ij} are distributed uniformly and independently in `(REAL(A),REAL(B))` and in `(AIMAG(A),AIMAG(B))`.

3. `tMUTL` copies the upper triangle of the square matrix (x_{jk}) of order `N` to the lower triangle of this matrix, thus creating a symmetric matrix.

4. The use of in-line `DO` loops will be more efficient than calling the equivalent matrix processing subprogram when the matrix dimensions are sufficiently small, due to the overhead of the subprogram call.

•

Author(s) : H. Lipps | **Library:** KERNLIB
Submitter : | **Submitted:** 18.12.1979
Language : Fortran or Assembler or COMPASS | **Revised:** 27.05.1987

Matrix Multiplication

These subprograms calculate the matrix product

$$\mathbf{Z} = \mathbf{X}\mathbf{Y} \text{ or } \mathbf{Z} = \mathbf{X}\overline{\mathbf{Y}},$$

where $\overline{Y}$ denotes the conjugate of the complex matrix **Y**, or one of the matrix expressions

$$\mathbf{Z} = \mathbf{X}\mathbf{Y} + \mathbf{Z}, \quad \mathbf{Z} = \mathbf{X}\mathbf{Y} - \mathbf{Z}, \quad \mathbf{Z} = -\mathbf{X}\mathbf{Y} + \mathbf{Z}, \quad \mathbf{Z} = -\mathbf{X}\mathbf{Y} - \mathbf{Z}.$$

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RMMLA`, `RMMLS`, `RMMLT`, `RMNMA`, `RMNMS`,
`DMMLA`, `DMMLS`, `DMMLT`, `DMNMA`, `DMNMS`,
`CMMLA`, `CMMLS`, `CMMLT`, `CMNMA`, `CMNMS`, `CMMLTC`
External References: `LOCF` (N100) (some Fortran versions only).

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`), `t` = `C` (type `COMPLEX`):

`CALL tMMLT (M,N,K,X11,X12,X21,Y11,Y12,Y21,Z11,Z12,Z21,W)`	$\mathbf{Z} = \mathbf{X}\mathbf{Y}$
`CALL tMMLA (M,N,K,X11,X12,X21,Y11,Y12,Y21,Z11,Z12,Z21)`	$\mathbf{Z} = \mathbf{X}\mathbf{Y} + \mathbf{Z}$
`CALL tMMLS (M,N,K,X11,X12,X21,Y11,Y12,Y21,Z11,Z12,Z21)`	$\mathbf{Z} = \mathbf{X}\mathbf{Y} - \mathbf{Z}$
`CALL tMNMA (M,N,K,X11,X12,X21,Y11,Y12,Y21,Z11,Z12,Z21)`	$\mathbf{Z} = -\mathbf{X}\mathbf{Y} + \mathbf{Z}$
`CALL tMNMS (M,N,K,X11,X12,X21,Y11,Y12,Y21,Z11,Z12,Z21)`	$\mathbf{Z} = -\mathbf{X}\mathbf{Y} - \mathbf{Z}$
`CALL CMMLTC(M,N,K,X11,X12,X21,Y11,Y12,Y21,Z11,Z12,Z21,W)`	$\mathbf{Z} = \mathbf{X}\overline{\mathbf{Y}}$

`M,N,K` (`INTEGER`) The mathematical dimensions of the matrices: **X** has `M` rows and `N` columns, **Y** has `N` rows and `K` columns, **Z** has `M` rows and `K` columns.

`X11,X12,X21` (Type according to `t`) Array elements. They must contain the elements x_{11}, x_{12}, x_{21} of the matrix **X**.

`Y11,Y12,Y21` (Type according to `t`) Array elements. They must contain the elements y_{11}, y_{12}, y_{21} of the matrix **Y**.

`Z11,Z12,Z21` (Type according to `t`) Array elements. On exit, they will contain the elements z_{11}, z_{12}, z_{21} of the matrix **Z**.

`W` (Type according to `t`) Working space array as specified below, required only if **Z** overlaps **X** or **Y**. Otherwise a dummy variable.

For `M` < `1` or `N` < `1` or `K` < `1`, all subroutines return control without action.

The matrices **X**, **Y** and **Z** need not to be stored according to the Fortran conventions: any equidistant spacing of their rows and columns is permitted. In particular, matrices may be stored row-wise. Each subroutine can work with the transpose of a matrix. To make this possible, each matrix is specified in the calling sequence by three arguments. For example, the called subroutine will operate on the matrix $\mathbf{A} = (a_{ij})$ if the actual arguments which replace `X11`, `X12`, `X21` in the calling sequence are a_{11}, a_{12}, a_{21}, and will operate on the transpose $\mathbf{A}'$ of **A** if the actual arguments are a_{11}, a_{21}, a_{12}.

The only cases in which the result matrix **Z** is permitted to overlap **X** or **Y** are the following:

`tMMLT`:	$\mathbf{X} = \mathbf{X}\mathbf{Y}$	or	$\mathbf{Y} = \mathbf{Y}'\mathbf{Y}$,	provided `W` is an array of at least `K` elements.
	$\mathbf{Y} = \mathbf{X}\mathbf{Y}$	or	$\mathbf{X} = \mathbf{X}\mathbf{X}'$,	provided `W` is an array of at least `M` elements.
`CMMLTC`:	$\mathbf{X} = \mathbf{X}\overline{\mathbf{Y}}$	or	$\mathbf{Y} = \mathbf{Y}'\overline{\mathbf{Y}}$,	provided `W` is an array of at least `K` elements.
	$\mathbf{Y} = \mathbf{X}\overline{\mathbf{Y}}$	or	$\mathbf{X} = \mathbf{X}\overline{\mathbf{X}}'$,	provided `W` is an array of at least `M` elements.

Accuracy:

On computers with IBM 370 architecture, all routines that accumulate the inner product of type `REAL` or `COMPLEX` use double-precision arithmetic internally; the final result is then rounded to single precision.

Notes:

The product of a matrix and its transpose (or Hermitian conjugate) is recognized by `tMMLT` (or `CMMLTC`) and the computation is shortened accordingly.

Examples:

Assume that the two-dimensional arrays `A`, `B`, `C`, `D`, `E`, the one-dimensional array `W`, and the dummy variable `V` are declared by

```
COMPLEX A(9,9),B(9,9),C(9,9),D(9,9),E(9,9),V,W(99)
```

and that a 4×5 matrix **A**, a 5×7 matrix **B**, and a 7×3 matrix **C** have been stored according to the Fortran conventions in arrays of corresponding name.

1. To compute $\mathbf{D} = \mathbf{A}\mathbf{B}$:

   ```
   CALL CMMLT (4,5,7,A,A(1,2),A(2,1),B,B(1,2),B(2,1),D,D(1,2),D(2,1),V).
   ```

   To pack the 4×7 product matrix **AB** row-wise into array `W`:

   ```
   CALL CMMLT (4,5,7,A,A(1,2),A(2,1),B,B(2,1),B(1,2),W,W(2),W(8),V).
   ```

   (Note that z_{11} goes into `W(1)`, z_{12} into `W(2)`, and z_{21} into `W(8)`).

   For the purpose of abbreviation we shall denote
   `A,A(1,2),A(2,1)` by `a`,   `A,A(2,1),A(1,2)` by `a'`,
   and similarly for arrays `B`, `C`, `D`, `E`. The first example above then becomes

   ```
   CALL CMMLT(4,5,7,a,b,d,V).
   ```

2. To compute $\mathbf{D} = \mathbf{B}'\mathbf{A}' = (\mathbf{A}\mathbf{B})'$:

   ```
   CALL CMMLT(7,5,4,b',a',d,V)  or  CMMLT(4,5,7,a,b,d',V).
   ```

3. To compute $\mathbf{D} = \mathbf{A}\mathbf{A}'$ and $\mathbf{E} = \mathbf{A}'\mathbf{A}$:

   ```
   CALL CMMLT(4,5,4,a,a',d,V)
   CALL CMMLT(5,4,5,a',a,e,V).
   ```

4. To replace **A** by **AB** or by $\mathbf{A}\mathbf{A}'$:

   ```
   CALL CMMLT(4,5,7,a,b,a,W)  or  CALL CMMLT(4,5,4,a,a',a,W).
   ```

   These two calls require a working vector `W` containing 7 or 4 complex elements, respectively.

5. To compute $\mathbf{D} = \mathbf{A}\overline{\mathbf{B}}$ and $\mathbf{E} = \overline{\mathbf{B}}\mathbf{C} = (\mathbf{C}'\overline{\mathbf{B}}')'$:

   ```
   CALL CMMLTC(4,5,7,a,b,d,V)
   CALL CMMLTC(3,7,5,c',b',e',V).
   ```

•

Author(s) : G.A. Erskine **Library:** KERNLIB
Submitter : **Submitted:** 18.12.1979
Language : Fortran **Revised:** 27.11.1984

Linear Equations, Matrix Inversion

Subroutine `tEQN` (where `t` = `R`, `D` or `C` as described below) solves the matrix equation

$$\mathbf{AX} = \mathbf{B}, \qquad\qquad (*)$$

which represents a system of N simultaneous linear equations with K right-hand sides:

$$\sum_{j=1}^{N} a_{ij} x_{jk} = b_{ik}, \qquad (i = 1, 2, \ldots, N,\ k = 1, 2, \ldots, K).$$

Subroutine `tINV` computes the inverse of a square matrix **A**. Subroutine `tEQINV` solves the system (*) and also computes the inverse of **A**, but is appreciably slower than `tEQN`.

If the determinant of **A** is also required, or if several systems of the form (*) are to be solved sequentially with the same coefficient matrix **A** but differing right-hand sides **B**, the subroutines in `RFACT` (F011) should be used.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `RINV`, `REQN`, `REQINV`, `DINV`, `DEQN`, `DEQINV`, `CINV`, `CEQN`, `CEQINV`
Internal Entry Names: `F010PR`
Files Refeenced: Printer
External References: `RFACT` (F011), `RFEQN` (F011), `RFINV` (F011),
`DFACT` (F011), `DFEQN` (F011), `DFINV` (F011),
`CFACT` (F011), `CFEQN` (F011), `CFINV` (F011),
`TMPRNT` (F011), `KERMTR` (N001), `ABEND` (Z035)

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`), `t` = `C` (type `COMPLEX`):

```
CALL tEQN  (N,A,IDIM,IR,IFAIL,K,B)
CALL tINV  (N,A,IDIM,IR,IFAIL)
CALL tEQINV(N,A,IDIM,IR,IFAIL,K,B)
```

`N` (`INTEGER`) Order of the square matrix **A**.

`A` (Type according to `t`) Two-dimensional array whose first dimension has the value `IDIM`.

`IDIM` (`INTEGER`) First dimension of array `A` (and of array `B` if `K` > `1`).

`IR` (`INTEGER`) Array of at least `N` elements, required as working space.

`IFAIL` (`INTEGER`) On exit, `IFAIL` will be set to -1 if **A** is found to be singular, and to `0` otherwise. (Singularity will often go undetected because of rounding errors during factorization even if the elements of **A** have integral values.)

`K` (`INTEGER`) Number of columns of the matrices **B** and **X**.

`B` (Type according to `t`) In general, a two-dimensional array whose first dimension has the value `IDIM`. `B` may be one-dimensional if `K` = `1`.

These subroutines must be called with matrix **A** in array `A` and matrix **B** in array `B`. Then, provided the matrix **A** is non-singular, `IFAIL` will be set to `0` and arrays `A` and `B` will be set as follows:

`tEQN`	The solution **X** replaces **B**. The matrix **A** is destroyed.
`tINV`	The inverse $\mathbf{A}^{-1}$ of **A** replaces **A**.
`tEQINV`	The solution **X** replaces **B**, and the inverse $\mathbf{A}^{-1}$ of **A** replaces **A**.

If the matrix **A** is singular, `IFAIL` will be set to -1. In this case the contents of `A` is unpredictable and the contents of `B` is unchanged.

Method:

Triangular factorization with row interchanges, implemented by in-line code if $\texttt{N} \leq 3$ and by calls to library program `RFACT` (F011) if $\texttt{N} > 3$. If $\texttt{N} < 1$ or $\texttt{IDIM} < \texttt{N}$ or $\texttt{K} < 1$, a message is printed and program execution is terminated by calling `ABEND` (Z035).

Examples:

Assume that the 10×10 matrix **A** and the 10×3 matrix **B** are stored according to the Fortran convention in arrays `A` and `B` respectively of a program containing declarations

```
DIMENSION IR(25)
DOUBLE PRECISION A(25,30),B(25,10)
```

To replace **B** by the 10×3 solution matrix **X** of the system of equations $\mathbf{AX} = \mathbf{B}$ and to replace **A** by $\mathbf{A}^{-1}$, with a jump to label `100` if **A** is singular:

```
CALL DEQINV (10,A,25,IR,IFAIL,3,B)
IF(IFAIL .NE. 0) GO TO 100
```

•

Author(s) : G.A. Erskine, H. Lipps **Library:** KERNLIB
Submitter : **Submitted:** 18.12.1979
Language : Fortran or Assembler or COMPASS **Revised:** 27.11.1984

Repeated Solution of Linear Equations, Matrix Inversion, Determinant

These subroutines provide a two-step procedure for solving sets of linear equations

$$\mathrm{AX} = \mathrm{B} \qquad (*)$$

which is faster than the library programs RINV (F010) when (*) must be solved repeatedly for the same matrix **A** with different sets of right-hand sides. The inverse matrix $\mathbf{A}^{-1}$ and the determinant det(**A**) may also be calculated.

Structure:

SUBROUTINE subprograms
User Entry Names: RFACT, RFEQN, RFINV, DFACT, DFEQN, DFINV, CFACT, CFEQN, CFINV
Internal Entry Names: TMPRNT
Files Referenced: Printer
External References: KERMTR (N001), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION), t = C (type COMPLEX):

```
CALL tFACT(N,A,IDIM,IR,IFAIL,DET,JFAIL)
CALL tFEQN(N,A,IDIM,IR,K,B)
CALL tFINV(N,A,IDIM,IR)
```

N (INTEGER) Order of the square matrix **A**.

A (Type according to t) Two-dimensional array whose first dimension has the value IDIM.

IDIM (INTEGER) First dimension of array A (and of array B if K $>$ 1).

IR (INTEGER) Array of at least N elements, required as working space.

IFAIL (INTEGER) On exit, IFAIL will be set to −1 if **A** is found to be singular, and to 0 otherwise. (Singularity will often go undetected because of rounding errors during factorization even if the elements of **A** have integral values.)

DET (Type according to t) On exit, DET will be set to the value det(**A**) unless JFAIL returns a non-zero value.

JFAIL (INTEGER) On exit, JFAIL will be set to zero if det(**A**) can be safely evaluated. Otherwise JFAIL is set as follows:
= −1 if det(**A**) is probably too small,
= +1 if det(**A**) is probably too large.

K (INTEGER) Number of columns of the matrices **B** and **X**.

B (Type according to t) In general, a two-dimensional array whose first dimension has the value IDIM. B may be one-dimensional if K = 1.

Subroutine tFACT must be called with matrix **A** in array A prior to any calls to tFEQN and tFINV. On return the situation is as follows:

1. Provided **A** is non-singular, IFAIL will be set to 0, and A and R will be set in preparation for calls to tFEQN and tFINV.

   If **A** is singular, IFAIL will be set to -1, in which case any subsequent call to tFEQN or tFINV will give unpredictable results.

2. Provided det(**A**) can be safely evaluated within the range of the computer, JFAIL will be set to 0 and and DET will be set to det(**A**). In particular, if **A** is singular, both JFAIL and DET will be set to zero.

   If the evaluation of det(**A**) would probably cause underflow, JFAIL will be set to -1 and DET will be set to zero.

   If the evaluation of det(**A**) would probably cause overflow, JFAIL will be set to $+1$ and DET will be incorrect.

   Execution continues, and subsequent calls to tFEQN and tFINV will give correct results.

Subroutine tFEQN may be called only after tFACT has been called, with the contents of A and R unchanged, and with matrix **B** in array B. On return, B will contain the solution **X**, with A and R unchanged. Therefore a single call to tFACT may be followed by several calls to tFEQN with differing **B**.

Subroutine tFINV may be called only after tFACT has been called, with the contents of A and R unchanged. On return, A will contain the inverse $\mathbf{A}^{-1}$ of **A**. Therefore, once tFINV has been called, it is no longer meaningful to call tFEQN with A as parameter.

Method:

Triangular factorization with row interchanges. The inverse matrix $\mathbf{A}^{-1}$ is the product, in reverse order, of the in-place inverses of the triangular factors. The array R holds information specifying the row interchanges.

Accuracy:

On computers with IBM 370 architecture, inner products are accumulated using double-precision arithmetic internally for arrays of type REAL and COMPLEX.

Error handling:

If $\mathtt{N} < 1$ or $\mathtt{IDIM} < \mathtt{N}$ or $\mathtt{K} < 1$, a message is printed and program execution is terminated by calling ABEND (Z035).

Examples:

Assume that the 10×10 matrix **A**, the 10×3 matrix **B**, and the 10-element vector **z** are stored according to the Fortran convention in arrays A, B and Z respectively of a program containing the declarations

```
DIMENSION IR(25)
COMPLEX A(25,30),B(25,10),Z(25),DET
```

Then, unless **A** is singular (which is to cause a jump to statement 100), the following statements will set $\mathtt{DET} = \det(\mathbf{A})$, replace **B** by $\mathbf{A}^{-1}\mathbf{B}$, replace **z** by $\mathbf{A}^{-1}\mathbf{z}$, and replace **A** by $\mathbf{A}^{-1}$:

```
CALL CFACT (10,A,25,IR,IFAIL,DET,JFAIL)
IF(IFAIL .NE. 0) GO TO 100
CALL CFEQN(10,A,25,IR,3,B)
CALL CFEQN(10,A,25,IR,1,Z)
CALL CFINV(10,A,25,IR)
```

•

Author(s) : H. Lipps **Library:** KERNLIB
Submitter : **Submitted:** 01.09.1983
Language : Fortran or Assembler or COMPASS **Revised:**

Symmetric Positive-Definite Linear Systems

Subroutine tSINV (where t = R or D as described below) computes the inverse of a symmetric positive-definite matrix **A**.

Subroutine tSEQN solves a set of linear equations

$$\mathrm{AX} = \mathrm{B} \qquad (*)$$

whose coefficient matrix **A** is symmetric and positive-definite. The determinant det(**A**) of **A** may be calculated by subroutine tSFACT described below.

If several systems of the form (*) are to be solved with the same **A** but differing **B**, a procedure which is appreciably faster than calling subroutine tSEQN repeatedly is to execute a single call to subroutine tSEQN (or subroutine tSFACT if the determinant is required), and then to call subroutine tSFEQN as many times as required. When the last system (*) has been solved, the inverse matrix A^{-1}, if required, may be computed by calling tSFINV.

Subroutine tSEQN and tSFACT both replace the matrix **A** by a lower triangular matrix **L** and an upper triangular matrix **U** such that $\mathrm{LU} = \mathrm{A}$. This **LU** decomposition is referred to below as **lu(A)**.

Given **lu(A)** and some matrix **B**, subroutine tSFEQN replaces **B** by the solution **X** of equation (*) without changing **lu(A)**. Subroutine tSFEQN may therefore be called repeatedly with differing **B**.

Given **lu(A)**, subroutine tSFINV replaces **lu(A)** by the inverse A^{-1} of **A**.

Structure:

SUBROUTINE subprograms
User Entry Names: RSFACT, RSEQN, RSFEQN, RSINV, RSFINV
DSFACT, DSEQN, DSFEQN, DSINV, DSFINV
Files Referenced: Printer
External References: TMPRNT (F011), KERMTR (N001), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION):

```
CALL tSINV (N,A,IDIM,IFAIL)
CALL tSEQN (N,A,IDIM,IFAIL,K,B)
CALL tSFACT(N,A,IDIM,IFAIL,DET,JFAIL)
CALL tSFEQN(N,A,IDIM,K,B)
CALL tSFINV(N,A,IDIM)
```

N (INTEGER) Order of the matrix **A**.

A (Type according to t) Two-dimensional array whose first dimension has the value IDIM.

IDIM (INTEGER) First dimension of array A (and of array B if K > 1).

IFAIL (INTEGER) On exit, IFAIL will be set to 0 if **A** is positive-definite, and to -1 otherwise.

DET (Type according to t) On exit, DET will be set to the value det(**A**) unless JFAIL returns a non-zero value.

JFAIL (INTEGER) On exit, JFAIL will be set to zero if det(**A**) can be safely evaluated. Otherwise JFAIL is set as follows:
= -2 if **A** is not positive-definite,
= -1 if det(**A**) is probably too small,
= $+1$ if det(**A**) is probably too large.

K (INTEGER) Number of columns of the matrices **B** and **X**.

B (Type according to t) In general, a two-dimensional array whose first dimension has the value IDIM. B may be one-dimensional if K = 1. tSEQN accepts a dummy argument B if K = 0.

The contents of arrays A and B on entry and exit are as follows:

tSINV On entry, **A** must be stored in A. On exit, A contains $\mathbf{A}^{-1}$ if IFAIL = 0, or else is undefined.

tSEQN On entry, **A** must be stored in A and **B** in B. On exit, A contains **lu(A)** and B contains **X** if IFAIL = 0, or else A is undefined and B is unchanged.

tSFACT On entry, **A** must be stored in A. On exit, A contains **lu(A)** if IFAIL = 0, or else is undefined. DET contains det(**A**) if JFAIL = 0, contains zero if JFAIL = −1, and is undefined otherwise.

tSFEQN On entry, **lu(A)** must be stored in A, and **B** in B. On exit, A is unchanged and B contains **X**.

tSFINV On entry, **lu(A)** must be stored in A. On exit, A contains $\mathbf{A}^{-1}$.

Method:

Modified Cholesky factorization (without square roots). See Ref. 1.

Accuracy:

On computers with IBM 370 architecture, inner products are accumulated using double precision arithmetic internally for arrays of type REAL.

Notes:

Only those elements a_{ij} of the original matrix **A** for which $i \geq j$ are required on entry to tSINV, tSEQN and tSFACT.

Error handling:

If N < 1 or IDIM < N or K < 0 (tSEQN) or K < 1 (tSFEQN), a message is printed and program execution is terminated by calling ABEND (Z035).

Examples:

Assume that the 10×10 matrix **A** and the 10×3 matrix **B** are stored according to the Fortran convention in arrays A and B respectively of a program containing the declarations

```
REAL A(25,30),B(25,10)
```

To replace **B** by the 10×3 solution matrix **X** of the system of equations $\mathbf{AX} = \mathbf{B}$, with a jump to label 100 if **A** is not positive definite:

```
CALL RSEQN(10,A,25,IFAIL,3,B)
IF(IFAIL .NE. 0) GO TO 100
```

References:

1. J.H. Wilkinson and C. Reinsch (eds.), Handbook for automatic computation, Vol.2: Linear algebra (Springer-Verlag, New York 1971), Chapter 2.

•

POLROT	CERN Program Library	F105

Author(s) : M. Regler **Library:** MATHLIB
Submitter : **Submitted:** 01.03.1968
Language : Fortran **Revised:** 27.11.1984

Rotate a Three-Dimensional Polar Coordinate System

`POLROT` calculates the values of θ' and ϕ' of the coordinate system $S'(\theta', \phi', r)$, obtained by rotation of the 3-dimensional polar coordinate system $S(\theta, \phi, r)$ about any axis $(0 \leq \theta \leq \pi, 0 \leq \phi \leq 2\pi)$.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `POLROT`

Usage:

```
CALL POLROT(THETA,PHI,THPRIM,PHPRIM,THAX,PHAX,ROTANG)
```

`THETA`	(`REAL`) Angle θ in the old system $S(\theta, \phi, r)$.
`PHI`	(`REAL`) Angle ϕ in the old system $S(\theta, \phi, r)$.
`THPRIM`	(`REAL`) Angle θ' in the new system $S'(\theta', \phi', r)$.
`PHPRIM`	(`REAL`) Angle ϕ' in the new system $S'(\theta', \phi', r)$.
`THAX,PHAX`	(`REAL`) Angles defining the axis of rotation in the old system $S(\theta, \phi, r)$.
`ROTANG`	(`REAL`) Angle in the old system through which the system is rotated.

The subroutine calculates from `THETA` and `PHI` the new values `THPRIM` and `PHPRIM` in a coordinate system obtained by rotating the old system through an angle `ROTANG` about an axis defined by `THAX` and `PHAX` in the old system.

Method:

`THETA` and `PHI` are converted to a unit vector in Cartesian coordinates; `THAX`, `PHAX` and `ROTANG` are converted to a tensor, which is used to obtain a vector in the new system of axes giving `THPRIM` and `PHPRIM`.

Notes:

If `THPRIM` is very small, `PHPRIM` is badly defined.

•

Author(s) : TC
Submitter : C. Letertre
Language : Fortran

Library: KERNLIB
Submitted: 01.08.1969
Revised: 07.03.1989

TC Matrix Manipulation Package

OBSOLETE

Please note that this routine has been obsoleted in CNL 194. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: `RVADD` (F002), `RMADD` (F003), `RMMLT` (F004)

The routines of `MXPACK` compute the product of two matrices or the product of their transposed matrices and may add or subtract to the resultant matrix a third one, add or subtract one matrix from another, or transfer a matrix, its negative, or a multiple of it, transpose a given matrix, build up a unit matrix, multiply a matrix by a diagonal (from left or from right) and may add the result to another matrix, add to square matrix the multiple of a diagonal matrix, compute the products $\mathbf{X} = \mathbf{ABA'}$ ($\mathbf{A'}$ denotes the transpose of $\mathbf{A}$) and $\mathbf{X} = \mathbf{A'BA}$. It is assumed that matrices are stored **row-wise without gaps**, contrary to the Fortran convention.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `MXMAD`, `MXMAD1`, `MXMAD2`, `MXMAD3`, `MXMPY`, `MXMPY1`, `MXMPY2`, `MXMPY3`, `MXMUB`, `MXMUB1`, `MXMUB2`, `MXMUB3`, `MXTRP`, `MXUTY`, `MXMLRT`, `MXMLTR`

Usage:

Matrix Multiplication

`CALL MXMPY(A,B,C,NI,NJ,NK)`	$(A_{ij})(B_{jk}) \to (C_{ik})$
`CALL MXMPY1(A,Q,C,NI,NJ,NK)`	$\mathbf{AQ'} \to \mathbf{C}$ ($\mathbf{Q}$ is `NK` $\times$ `NJ`)
`CALL MXMPY2(P,B,C,NI,NJ,NK)`	$\mathbf{P'B} \to \mathbf{C}$ ($\mathbf{P}$ is `NJ` $\times$ `NI`)
`CALL MXMPY3(P,Q,C,NI,NJ,NK)`	$\mathbf{P'Q'} \to \mathbf{C}$

If `NJ` $=$ `0`, $\mathbf{C}$ will be filled with zeros.

Matrix Multiplication and Addition

`CALL MXMAD(A,B,C,NI,NJ,NK)`	$(A_{ij})(B_{jk}) + (C_{ik}) \to (C_{ik})$
`CALL MXMAD1(A,Q,C,NI,NJ,NK)`	$\mathbf{AQ'} + \mathbf{C} \to \mathbf{C}$
`CALL MXMAD2(P,B,C,NI,NJ,NK)`	$\mathbf{P'B} + \mathbf{C} \to \mathbf{C}$
`CALL MXMAD3(P,Q,C,NI,NJ,NK)`	$\mathbf{P'Q'} + \mathbf{C} \to \mathbf{C}$

If `NJ` $=$ `0`, $\mathbf{C}$ will not be changed.

Matrix Multiplication and Subtraction

`CALL MXMUB(A,B,C,NI,NJ,NK)`	$(A_{ij})(B_{jk}) - (C_{ik}) \to (C_{ik})$
`CALL MXMUB1(A,Q,C,NI,NJ,NK)`	$\mathbf{AQ}' - \mathbf{C} \to \mathbf{C}$
`CALL MXMUB2(P,B,C,NI,NJ,NK)`	$\mathbf{P}'\mathbf{B} - \mathbf{C} \to \mathbf{C}$
`CALL MXMUB3(P,Q,C,NI,NJ,NK)`	$\mathbf{P}'\mathbf{Q}' - \mathbf{C} \to \mathbf{C}$

If `NJ` $=$ `0`, $\mathbf{C}$ will be replaced by $-\mathbf{C}$.

Matrix Transposition

`CALL MXTRP(A,B,NI,NJ)`	$(A_{ij}) \to (B_{ji})$

Unity Matrix

`CALL MXUTY(A,NI)`	$(A_{ii}) = 1;\ (A_{ij}) = 0,\ (i \neq j)$

Matrix Multiplication

`CALL MXMLRT(A,B,X,M,N)`	$\mathbf{A}[m \times n]\,\mathbf{B}[n \times n]\,\mathbf{A}'[n \times m] \to \mathbf{X}[m \times m]$
`CALL MXMLTR(A,B,X,N,M)`	$\mathbf{A}'[n \times m]\,\mathbf{B}[m \times m]\,\mathbf{A}[m \times n] \to \mathbf{X}[n \times n]$

Notes:

In the formulae above, (A_{ij}) *etc* denotes the ensemble of elements of the matrix $\mathbf{A}$ *etc* with the row index i and the column index j. The Fortran variables `NI`, `NJ` and `NK` specify the dimensions associated with the indices i, j and k. If `DIMENSION A(NJ,NI)` reserves space for the matrix $\mathbf{A}$, then the element A_{ij} is contained in `A(J,I)`.

•

Author(s) : W. Hart
Library: KERNLIB
Submitter :
Submitted: 01.01.1975
Language : Fortran
Revised: 12.12.1986

Manipulation of Triangular and Symmetric Matrices

At CERN, matrices are often stored row-wise (TC-convention); furthermore, symmetric matrices are stored packed as the lower left triangular part only, i.e., the Ith diagonal element is found in position $I(I+1)/2$. The `TR`-package performs many of the frequently required operations associated with such matrices without resorting to expanding into the unpacked square form. In all the following routines an $M \times M$ *symmetric* matrix is taken to be stored in the packed form with $M(M+1)/2$ elements.

Some of these operations produce and require the manipulation of *lower triangular* matrices which have all elements zero above the leading diagonal. These are also stored in the packed form with all the zeros dropped; therefore, care has to be taken in the interpretation of a packed matrix as to whether it represents a symmetric or lower triangular array. To facilitate this distinction in the Write-up, the following nomenclature has been adopted:

`A,B,C`	unpacked rectangular matrices (row-wise storage)
`Q,R,S,T`	packed symmetric matrices
`V,W`	packed lower triangular matrices

On 32-bit machines the calculations are performed internally in double-precision mode.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `TRCHUL`, `TRCHLU`, `TRSMUL`, `TRSMLU`, `TRINV`, `TRSINV`, `TRLA`, `TRLTA`, `TRAL`, `TRALT`, `TRSA`, `TRAS`, `TRSAT`, `TRATS`, `TRAAT`, `TRATA`, `TRASAT`, `TRATSA`, `TRQSQ`, `TRPCK`, `TRUPCK`

Usage:

Choleski Decomposition

`CALL TRCHUL(S,W,M)` $\quad \mathbf{S} = \mathbf{W}'\mathbf{W}$

`CALL TRCHLU(S,V,M)` $\quad \mathbf{S} = \mathbf{V}\mathbf{V}'$

`S` is an `M` $\times$ `M` *positive semi-definite* symmetric matrix (e.g., error or weight matrix) and the routines calculate the complementary lower triangular Choleski factors. It is allowed to overwrite `S` by `W` or `V`.

Symmetric Multiplication of Lower Triangular Matrices

`CALL TRSMUL(W,S,M)` $\quad \mathbf{W}'\mathbf{W} \to \mathbf{S}$

`CALL TRSMLU(W,R,M)` $\quad \mathbf{W}\mathbf{W}' \to \mathbf{R}$

`W` is an `M` $\times$ `M` lower triangular matrix and `S`, `R` the two symmetric products of the multiplication of `W` by its transpose. It is allowed to overwrite `W` by either `S` or `R`.

Lower Triangular Matrix Inversion

```
CALL TRINV(W,V,M)
```
$\mathbf{W}^{-1} \rightarrow \mathbf{V}$

`W` is an `M` $\times$ `M` lower triangular matrix which is inverted into `V` (the inverse of a lower triangular matrix is lower triangular). `W` *may have rows and columns of zeros* as produced by the Choleski decomposition of a weight matrix with unmeasured variables. It is allowed to overwrite `W` by `V`.

Symmetric Matrix Inversion

```
CALL TRSINV(S,R,M)
```
$\mathbf{S}^{-1} \rightarrow \mathbf{R}$

`S` is an `M` $\times$ `M` *positive semi-definite* symmetric matrix which is inverted into `R` (also stored packed). It is permissible to overwrite `S` by `R`.

Triangular – Rectangular Multiplication

`CALL TRLA (W,A,B,M,N)`	$\mathbf{WA} \rightarrow \mathbf{B}$
`CALL TRLTA(W,A,B,M,N)`	$\mathbf{W'A} \rightarrow \mathbf{B}$
`CALL TRAL (A,V,B,M,N)`	$\mathbf{AV} \rightarrow \mathbf{B}$
`CALL TRALT(A,V,B,M,N)`	$\mathbf{AV'} \rightarrow \mathbf{B}$

`A` and `B` are `M` $\times$ `N` rectangular matrices, `W` is an `M` $\times$ `M` lower triangular matrix, and `V` is an `N` $\times$ `N` lower triangular matrix. In each call it is allowed to overwrite `A` by `B`.

Symmetric - Rectangular Multiplication

`CALL TRSA (S,A,C,M,N)`	$\mathbf{SA} \rightarrow \mathbf{C}$
`CALL TRAS (A,R,C,M,N)`	$\mathbf{AR} \rightarrow \mathbf{C}$
`CALL TRSAT(S,B,C,M,N)`	$\mathbf{SB'} \rightarrow \mathbf{C}$
`CALL TRATS(B,R,C,M,N)`	$\mathbf{B'R} \rightarrow \mathbf{C}$

`A` and `C` are `M` $\times$ `N` rectangular matrices, `B` is an `N` $\times$ `M` matrix, `S` is an `M` $\times$ `M` symmetrix matrix, and `R` is an `N` $\times$ `N` symmetric matrix. It is *not* allowed to overwrite `A` or `B` by the product matrix `C`.

Symmetric Multiplication of Rectangular Matrices

`CALL TRAAT(A,S,M,N)`	$\mathbf{AA'} \rightarrow \mathbf{S}$
`CALL TRATA(B,R,M,N)`	$\mathbf{B'B} \rightarrow \mathbf{R}$

`A` is an `M` $\times$ `N` matrix, `B` is an `N` $\times$ `M` matrix, `S` is an `M` $\times$ `M` symmetric matrix, and `R` is an `M` $\times$ `M` symmetric matrix. No overwriting is allowed.

Transformation of Symmetric Matrix

`CALL TRASAT(A,S,R,M,N)`	$\mathbf{ASA'} \rightarrow \mathbf{R}$
`CALL TRATSA(B,S,R,M,N)`	$\mathbf{B'SB} \rightarrow \mathbf{R}$
`CALL TRQSQ (Q,T,R,M)`	$\mathbf{QTQ} \rightarrow \mathbf{R}$

`A` is an `M` $\times$ `N` matrix, `B` is an `N` $\times$ `M` matrix, `S` is an `N` $\times$ `N` symmetric matrix, and `R`, `Q`, `T` are `M` $\times$ `M` symmetric matrices. No overwriting is allowed.

Packing and Unpacking a Symmetric Matrix

`CALL TRPCK (A,S,M)`	$\mathbf{A} \rightarrow \mathbf{S}$
`CALL TRUPCK(S,A,M)`	$\mathbf{S} \rightarrow \mathbf{A}$

`A` is an `M` $\times$ `M` unpacked symmetric matrix (all $\mathtt{M}^2$ elements) and `S` is the same matrix stored packed. Overwriting is allowed for both `TRPCK` and `TRUPCK`.

•

Author(s) : CERN TC Division **Library:** KERNLIB
Submitter : C. Letertre **Submitted:** 01.09.1969
Language : Fortran **Revised:** 27.11.1984

Scalar Product of Two Space-Time Vectors

Function subprogram `DOTI` computes the scalar product $\mathbf{a}.\mathbf{b}$ of two space-time vectors (a_1, a_2, a_3, ia_4), (b_1, b_2, b_3, ib_4), where $i = \sqrt{-1}$, i.e.

$$\mathbf{a}.\mathbf{b} \;=\; a_1b_1 + a_2b_2 + a_3b_3 - a_4b_4.$$

Structure:

`FUNCTION` subprogram
User Entry Names: `DOTI`

Usage:

In any arithmetic expression,

```
DOTI(A,B)
```

has the value $\mathbf{a}.\mathbf{b}$.

`A,B` (`REAL`) One-dimensional arrays of length 4, containing a_j, b_j, $(j = 1, 2, 3, 4)$, respectively.

•

Author(s) : CERN TC Division **Library:** KERNLIB
Submitter : C. Letertre **Submitted:** 01.09.1969
Language : Fortran **Revised:**

Vector Product of Two 3-Vectors

Subroutine subprogram CROSS computes the vector (or cross) product

$$\mathbf{c} = \mathbf{a} \times \mathbf{b}$$

of two 3-vectors $\mathbf{a}$, $\mathbf{b}$.

Structure:

SUBROUTINE subprogram
User Entry Names: CROSS
COMMON Block Names and Lengths: /SLATE/ 40

Usage:

```
CALL CROSS(A,B,C)
```

A,B (REAL) One-dimensional arrays of length 3, containing the components (a_1, a_2, a_3), (b_1, b_2, b_3), respectively.

C (REAL) On exit, C contains the components (c_1, c_2, c_3) of $\mathbf{a} \times \mathbf{b}$, i.e.

$$c_1 = a_2 b_3 - a_3 b_2$$

$$c_2 = a_3 b_1 - a_1 b_3$$

$$c_3 = a_1 b_2 - a_2 b_1.$$

C may overlap either A or B.

•

Author(s) : CERN TC Division **Library:** KERNLIB
Submitter : C. Letertre **Submitted:** 01.09.1969
Language : Fortran **Revised:**

Rotating a 3-Vector

Subroutine subprogram `ROT` rotates a 3-vector (a_1, a_2, a_3) by a given angle θ around the $z-$axis.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `ROT`
`COMMON` Block Names and Lengths: `/SLATE/ 40`

Usage:

```
CALL ROT(A,TH,B)
```

`A` (`REAL`) One-dimensional array of length 3, containing (a_1, a_2, a_3).

`TH` (`REAL`) Angle θ given in radians.

`B` (`REAL`) One-dimensional array of length 3. On exit, `B` contains the components (b_1, b_2, b_3) of the rotated vector, i.e.

$$b_1 = a_1 \cos\theta - a_2 \sin\theta$$

$$b_2 = a_1 \sin\theta + a_2 \cos\theta$$

$$b_3 = a_3.$$

`B` may overlap `A`.

•

Author(s) : M. Aderholz, P.M. Nicholson
Submitter : M. Aderholz
Language : Fortran or Assembler

Library: KERNLIB
Submitted: 01.06.1973
Revised: 16.09.1991

Vector Algebra

Performs various vector manipulations, such as addition of two vectors, multiplication of a vector by a scalar, scalar product, pre- and post-multiplication of a vector by a matrix.

Structure:

SUBROUTINE, and FUNCTION subprograms
User Entry Names: VADD, VSUB, VMUL, VBIAS, VSCALE, VLINCO, VUNIT, VMATR, VMATL, VCOPYN, VFIX, VFLOAT, VFILL, VZERO, VBLANK, VEXCUM, VDIST, VDIST2, VDOT, VDOTN, VDOTN2, VMOD, VASUM, VSUM, VMAXA, VMAX, VMINA, VMIN, LVMAXA, LVMAX, LVMINA, LVMIN, LVSMI, LVSMX, LVSDMI, LVSDMX, LVSIMI, LVSIMX

Notes:

VLINE is the original and obsolete name for the linear combination routine VLINCO; it was changed because it clashed with an entry point in some system library.

Usage:

The arguments in the calling sequences below are defined as follows:

A,B,X	(REAL) One-dimensional arrays of length N.
DA	(DOUBLE PRECISION) One-dimensional array of length N.
IA,IX	(INTEGER) One-dimensional arrays of length N.
C,V	(REAL) One-dimensional arrays of length M.
EX	(REAL) One-dimensional array of length 3.
G	(REAL) Two-dimensional array of dimension (M,N).
ALPHA	(REAL) Variable.
F1,F2	(REAL) Variables.
Y	(REAL) Variable.
N,M	(INTEGER) Variables.

Matrix G is assumed to be stored *row-wise*, contrary to the Fortran convention, i.e. element G_{ij} is found in word G(J,I) of the memory allocated with DIMENSION G(M,N).
Any summation $\sum$ is taken over the index I from 1 to N or over the index J from 1 to M.

Subroutines

CALL VADD(A,B,X,N)	$\mathtt{X(I)} = \mathtt{A(I)} + \mathtt{B(I)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VSUB(A,B,X,N)	$\mathtt{X(I)} = \mathtt{A(I)} - \mathtt{B(I)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VMUL(A,B,X,N)	$\mathtt{X(I)} = \mathtt{A(I)} * \mathtt{B(I)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VBIAS(A,ALPHA,X,N)	$\mathtt{X(I)} = \mathtt{A(I)} + \mathtt{ALPHA}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VSCALE(A,ALPHA,X,N)	$\mathtt{X(I)} = \mathtt{A(I)} * \mathtt{ALPHA}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VLINCO(A,F1,B,F2,X,N)	$\mathtt{X(I)} = \mathtt{A(I)} * \mathtt{F1} + \mathtt{B(I)} * \mathtt{F2}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VUNIT(A,X,N)	$x = a/\lvert a\rvert$	
	$\mathtt{X(I)} = \mathtt{A(I)}/\mathtt{VMOD(A,N)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VMATR(A,G,V,N,M)	$v = aG$	
	$\mathtt{V(J)} = \sum \mathtt{A(I)} * \mathtt{G(J,I)}$	$(\mathtt{J} = 1, 2, \ldots, \mathtt{M})$
CALL VMATL(G,C,X,N,M)	$x = Gc$	
	$\mathtt{X(I)} = \sum \mathtt{G(J,I)} * \mathtt{C(J)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VCOPYN(A,X,N)	$\mathtt{X(I)} = -\mathtt{A(I)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VFIX(A,IX,N)	$\mathtt{IX(I)} = \mathtt{A(I)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VFLOAT(IA,X,N)	$\mathtt{X(I)} = \mathtt{IA(I)}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VFILL(X,N,ALPHA)	$\mathtt{X(I)} = \mathtt{ALPHA}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VZERO(IX,N)	$\mathtt{IX(I)} = \mathtt{0}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VBLANK(IX,N)	$\mathtt{IX(I)} = \mathtt{blank}$	$(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$
CALL VEXCUM(A,EX,N)	$\mathtt{EX(1)} = \min(\mathtt{EX(1)}, \mathtt{A(1)}, \ldots, \mathtt{A(N)})$	
	$\mathtt{EX(2)} = \max(\mathtt{EX(2)}, \mathtt{A(1)}, \ldots, \mathtt{A(N)})$	
	$\mathtt{EX(3)} = \mathtt{EX(3)} + \sum \mathtt{A(I)}$	

REAL functions

VDIST2(A,B,N)	$(a-b)^2 = \sum(\mathtt{A(I)} - \mathtt{B(I)})^2$
VDIST(A,B,N)	$\lvert a-b\rvert = \sqrt{(a-b)^2}$
VDOT(A,B,N)	$ab = \sum \mathtt{A(I)} * \mathtt{B(I)}$
VDOTN2(A,B,N)	$(ab)^2/(a^2b^2)$
VDOTN(A,B,N)	$ab/\lvert a\rvert\lvert b\rvert$
VMOD (A,N)	$\lvert a\rvert = \sqrt{a^2}$
VASUM(A,N)	$\sum \lvert\mathtt{A(I)}\rvert$
VSUM (A,N)	$\sum \mathtt{A(I)}$
VMAXA(A,N)	$\max(\lvert\mathtt{A(1)}\rvert, \lvert\mathtt{A(2)}\rvert, \ldots, \lvert\mathtt{A(N)}\rvert)$
VMAX (A,N)	$\max(\mathtt{A(1)}, \mathtt{A(2)}, \ldots, \mathtt{A(N)})$
VMINA(A,N)	$\min(\lvert\mathtt{A(1)}\rvert, \lvert\mathtt{A(2)}\rvert, \ldots, \lvert\mathtt{A(N)}\rvert)$
VMIN (A,N)	$\min(\mathtt{A(1)}, \mathtt{A(2)}, \ldots, \mathtt{A(N)})$

INTEGER functions

`LVMAXA(A,N)`	Location of $\max \lvert \mathtt{A(I)} \rvert$
`LVMAX (A,N)`	Location of $\max \mathtt{A(I)}$
`LVMINA(A,N)`	Location of $\min \lvert \mathtt{A(I)} \rvert$
`LVMIN (A,N)`	Location of $\min \mathtt{A(I)}$
`LVSMI(A,N,INC)`	Location of $\min \mathtt{A(k)}$
`LVSMX(A,N,INC)`	Location of $\max \mathtt{A(k)}$
`LVSDMI(DA,N,INC)`	Location of $\min \mathtt{DA(k)}$
`LVSDMX(DA,N,INC)`	Location of $\max \mathtt{DA(k)}$
`LVSIMI(IA,N,INC)`	Location of $\min \mathtt{IA(k)}$
`LVSIMX(IA,N,INC)`	Location of $\max \mathtt{IA(k)}$

where $\mathtt{k} = 1, 1 + \mathtt{INC}, 1 + 2 * \mathtt{INC}, \ldots, 1 + (\mathtt{N} - 1) * \mathtt{INC}$

•

Author(s) : F. Antonelli
Submitter : F. Carminati
Language : Fortran (IBM: Assembler)

Library: MATHLIB
Submitted: 29.05.1989
Revised:

Search Operations on Sparse Vectors

Performs logical search and data movement operations on sparse vectors. On Cray systems these routines are part of the default libraries (scilib). An optimized Assembler version is provided for IBM 3090 with Vector Facilities. Fortran code is used on the other systems.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `IILZ`, `ILSUM`, `SCATTER`, `GATHER`, `WHENEQ`, `WHENNE`, `WHENFLT`, `WHENFGT`, `WHENFLE`, `WHENFGE`, `WHENILT`, `WHENIGT`, `WHENILE`, `WHENIGE`

Usage:

The arguments in the calling sequences below are defined as follows:

`A,B`	(`REAL`) One-dimensional arrays.
`IA,INDX`	(`INTEGER`) One-dimensional arrays.
`LA`	(`LOGICAL`) One-dimensional array.
`NW,INC`	(`INTEGER`) Variables or expressions.
`TARG`	(`REAL`) Variable or expression.
`ITARG,NFOUND`	(`INTEGER`) Variables.

In any arithmetic expression,

```
IILZ(NW,A,INC)
```

represents the `INTEGER` number of leading zero elements in
$\mathtt{LA(1)}, \mathtt{LA(INC+1)}, \mathtt{LA(2*INC+1)}, \ldots, \mathtt{LA((NW-1)*INC+1)}$;

```
ILSUM(NW,LA,INC)
```

represents the `INTEGER` number of `.TRUE.` elements in
$\mathtt{LA(1)}, \mathtt{LA(INC+1)}, \mathtt{LA(2*INC+1)}, \ldots, \mathtt{LA((NW-1)*INC+1)}$.

```
CALL SCATTER(NW,A,INDX,B)
CALL GATHER(NW,A,B,INDX)
```

set $\mathtt{A(INDX(I))} = \mathtt{B(I)}$, $(\mathtt{I} = 1, 2, \ldots, \mathtt{NW})$ and $\mathtt{A(I)} = \mathtt{B(INDX(I))}$, $(\mathtt{I} = 1, 2, \ldots, \mathtt{NW})$, respectively.

```
CALL WHENFxx(NW,A,INC,TARG,INDX,NFOUND)
```

searches $\mathtt{A(1)}, \mathtt{A(INC+1)}, \mathtt{A(2*INC+1)}, \ldots, \mathtt{A((NW-1)*INC+1)}$ for elements which satisfy the relation `A(.).xx.TARG` where $\mathtt{xx} = \mathtt{LT}, \mathtt{LE}, \mathtt{GT}, \mathtt{GE}$. On exit, $\mathtt{INDX(1)}, \ldots, \mathtt{INDX(NFOUND)}$ will contain the indices of the `NFOUND` elements which satisfy the relation specified.

```
CALL WHENIxx(NW,IA,INC,ITARG,INDX,NFOUND)
```

performes the same task as `WHENFxx` but for `INTEGER` draw and target.

```
CALL WHENEQ(NW,a,INC,targ,INDX,NFOUND)
CALL WHENNE(NW,a,INC,targ,INDX,NFOUND)
```

performs the same task as `WHENFxx` or `WHENIxx`, but for `xx` = `EQ`, `NE`, and `REAL` draw `a` and `REAL` target `targ`, or `INTEGER` draw `a` and `INTEGER` target `targ`, respectively.

•

BVSL	CERN Program Library	F123

Author(s) : F. Antonelli
Submitter : F. Carminati
Language : Fortran, IBM Assembler

Library: MATHLIB
Submitted: 27.11.1989
Revised: 16.08.1994

Bit Vector Manipulation Package

This package contains high performance procedures to operate with sparse arrays using Bit Vectors instead of ordinary Index Vectors to address the elements of an arrays. The routines are, at present, available only on IBM 3090 VF machines.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names:

`YLOSB, IYLOSB, YLOXB, IYLOXB,`
`GTHRB, SCTTB, ANDB, XORB, NOTB, NANDB, NORB, ORB, BINVEC, ZEROB,`
`ONEB, CNTOB, CNTZB, RANGB, INTGB, RJCTB, SXPYB, VXPYB, SXYB, XPWZB,`
`DOTB, SCALB, VSETB, COPYB`

Usage:

The arguments in the calling sequences below are defined as follows:

`NW`	(`INTEGER`) Number of elements to process. The index `i` below runs from `1` to `NW`.
`Y,X,V,W`	(`REAL`) Arrays of length `NW` at least.
`IX,IY`	(`INTEGER`) Arrays of length `NW` at least.
`S,T`	(`REAL`) Variables or expressions.
`IS,IT`	(`INTEGER`) Variables or expressions.
`BV,BV1,BV2`	Arrays of length $(\mathtt{NW}-1)/32+1$ at least, used to contain the bit vectors.
`IFOUND`	(`INTEGER`) Number of elements which satisfy the condition, or set-bit count, for `BV`.

The expression `X(BV)` indicates all these elements of the vector `X` for which the corresponding bit is set in the bit array `BV`. `BV(i)` indicates the `i`-th bit of the array `BV`, counted across words boundaries. The expression $\mathtt{BV(i)}=1$ means that the `i`-th bit of the array `BV` is set.

Vector to scalar comparison:

Two `SUBROUTINE` subprograms are provided for `REAL` and `INTEGER` comparison. The subprogram `YLOSB` is for vectors with `REAL` elements and the subprogram `IYLOSB` for vectors with `INTEGER` elements.

`CALL YLOSB(NW,Y,S,BV,IFOUND,'EQ')`	$\mathtt{BV(i)}=1$ if $\mathtt{Y(i)}=\mathtt{S}$
`CALL YLOSB(NW,Y,S,BV,IFOUND,'NE')`	$\mathtt{BV(i)}=1$ if $\mathtt{Y(i)}\neq\mathtt{S}$
`CALL YLOSB(NW,Y,S,BV,IFOUND,'GT')`	$\mathtt{BV(i)}=1$ if $\mathtt{Y(i)}>\mathtt{S}$
`CALL YLOSB(NW,Y,S,BV,IFOUND,'LT')`	$\mathtt{BV(i)}=1$ if $\mathtt{Y(i)}<\mathtt{S}$
`CALL YLOSB(NW,Y,S,BV,IFOUND,'GE')`	$\mathtt{BV(i)}=1$ if $\mathtt{Y(i)}\geq\mathtt{S}$
`CALL YLOSB(NW,Y,S,BV,IFOUND,'LE')`	$\mathtt{BV(i)}=1$ if $\mathtt{Y(i)}\leq\mathtt{S}$
`CALL IYLOSB(NW,Y,S,BV,IFOUND,'EQ')`	$\mathtt{BV(i)}=1$ if $\mathtt{IY(i)}=\mathtt{IS}$
`CALL IYLOSB(NW,Y,S,BV,IFOUND,'NE')`	$\mathtt{BV(i)}=1$ if $\mathtt{IY(i)}\neq\mathtt{IS}$
`CALL IYLOSB(NW,IY,IS,BV,IFOUND,'GT')`	$\mathtt{BV(i)}=1$ if $\mathtt{IY(i)}>\mathtt{IS}$
`CALL IYLOSB(NW,IY,IS,BV,IFOUND,'LT')`	$\mathtt{BV(i)}=1$ if $\mathtt{IY(i)}<\mathtt{IS}$
`CALL IYLOSB(NW,IY,IS,BV,IFOUND,'GE')`	$\mathtt{BV(i)}=1$ if $\mathtt{IY(i)}\geq\mathtt{IS}$
`CALL IYLOSB(NW,IY,IS,BV,IFOUND,'LE')`	$\mathtt{BV(i)}=1$ if $\mathtt{IY(i)}\leq\mathtt{IS}$

Vector to vector comparison:

Two `SUBROUTINE` subprograms are provided for `REAL` and `INTEGER` comparison. The subprogram `YLOXB` is for vectors with `REAL` elements and the subprogram `IYLOXB` for vectors with `INTEGER` elements.

`CALL YLOXB(NW,Y,X,BV,IFOUND,'EQ')`	$\mathtt{BV(i)} = 1$ if $\mathtt{Y(i)} = \mathtt{X(i)}$
`CALL YLOXB(NW,Y,X,BV,IFOUND,'NE')`	$\mathtt{BV(i)} = 1$ if $\mathtt{Y(i)} \neq \mathtt{X(i)}$
`CALL YLOXB(NW,Y,X,BV,IFOUND,'GT')`	$\mathtt{BV(i)} = 1$ if $\mathtt{Y(i)} > \mathtt{X(i)}$
`CALL YLOXB(NW,Y,X,BV,IFOUND,'LT')`	$\mathtt{BV(i)} = 1$ if $\mathtt{Y(i)} < \mathtt{X(i)}$
`CALL YLOXB(NW,Y,X,BV,IFOUND,'GE')`	$\mathtt{BV(i)} = 1$ if $\mathtt{Y(i)} \geq \mathtt{X(i)}$
`CALL YLOXB(NW,Y,X,BV,IFOUND,'LE')`	$\mathtt{BV(i)} = 1$ if $\mathtt{Y(i)} \leq \mathtt{X(i)}$
`CALL IYLOXB(NW,Y,X,BV,IFOUND,'EQ')`	$\mathtt{BV(i)} = 1$ if $\mathtt{IY(i)} = \mathtt{IX(i)}$
`CALL IYLOXB(NW,Y,X,BV,IFOUND,'NE')`	$\mathtt{BV(i)} = 1$ if $\mathtt{IY(i)} \neq \mathtt{IX(i)}$
`CALL IYLOXB(NW,IY,IX,BV,IFOUND,'GT')`	$\mathtt{BV(i)} = 1$ if $\mathtt{IY(i)} > \mathtt{IX(i)}$
`CALL IYLOXB(NW,IY,IX,BV,IFOUND,'LT')`	$\mathtt{BV(i)} = 1$ if $\mathtt{IY(i)} < \mathtt{IX(i)}$
`CALL IYLOXB(NW,IY,IX,BV,IFOUND,'GE')`	$\mathtt{BV(i)} = 1$ if $\mathtt{IY(i)} \geq \mathtt{IX(i)}$
`CALL IYLOXB(NW,IY,IX,BV,IFOUND,'LE')`	$\mathtt{BV(i)} = 1$ if $\mathtt{IY(i)} \leq \mathtt{IX(i)}$

Scatter/gather operations:

```
CALL GTHRB(NW,X,BV,Y)       Y=X(BV)
CALL SCTTB(NW,Y,BV,X)       Y(BV)=X
```

Elements are gathered or scattered from vector `X` into vector `Y` according to the bit mask contained in `BV`. Only words for which the corresponding bit is set are moved.

Logical operations:

`CALL ANDB(NW,BV1,BV2,BV,IFOUND)`	$\mathtt{BV(i)} = 1$ if $\mathtt{BV1(i)} = 1 \wedge \mathtt{BV2(i)} = 1$
`CALL ORB(NW,BV1,BV2,BV,IFOUND)`	$\mathtt{BV(i)} = 1$ if $\mathtt{BV1(i)} = 1 \vee \mathtt{BV2(i)} = 1$
`CALL XORB(NW,BV1,BV2,BV,IFOUND)`	$\mathtt{BV(i)} = 1$ if $\begin{array}{l}(\mathtt{BV1(i)} = 1 \vee \mathtt{BV2(i)} = 1) \wedge \\ \neg(\mathtt{BV1(i)} = 1 \wedge \mathtt{BV2(i)} = 1)\end{array}$
`CALL NANDB(NW,BV1,BV2,BV,IFOUND)`	$\mathtt{BV(i)} = 1$ if $\mathtt{BV1(i)} = 0 \vee \mathtt{BV2(i)} = 0$
`CALL NORB(NW,BV1,BV2,BV,IFOUND)`	$\mathtt{BV(i)} = 1$ if $\begin{array}{l}(\mathtt{BV1(i)} = 1 \wedge \mathtt{BV2(i)} = 1) \vee \\ (\mathtt{BV1(i)} = 0 \wedge \mathtt{BV2(i)} = 0)\end{array}$
`CALL NOTB(NW,BV1,BV,IFOUND)`	$\mathtt{BV(i)} = 1$ if $\mathtt{BV(i)} = 1 - \mathtt{BV1(i)}$

Miscellaneous operations:

```
CALL BINVEC(NW,BV,IVEC)
```

is equivalent to

```
DO J = 1,NW
   IF bit J of BV is set THEN
      IVEC(IFOUND)=J
   ENDIF
ENDDO
```

`CALL ZEROB(NW,BV)`	$\texttt{BV(i)} = 0$
`CALL ONEB (NW,BV)`	$\texttt{BV(i)} = 1$
`CALL CNTOB(NW,BV,IFOUND)`	`IFOUND` = Number of set bits
`CALL CNTZB(NW,BV,IFOUND)`	`IFOUND` = Number of clear bits
`CALL RANGB(NW,Y,S,T,BV,IFOUND)`	$\texttt{BV(i)} = 1$ if $\texttt{S} \leq \texttt{Y(i)} \leq \texttt{T}$
`CALL INTGB(NW,Y,V,W,BV,IFOUND)`	$\texttt{BV(i)} = 1$ if $\texttt{V(i)} \leq \texttt{Y(i)} \leq \texttt{W(i)}$

```
CALL RJCTB(RAN,X,FREJ,Y,BV,NW,NWOUT,ISWTCH)
```

`RAN`	Array of random numbers uniformly distributed between zero and the maximum of the rejection function.
`X`	Array of points where the rejection function is computed.
`FREJ`	Array of values of the rejection function.
`Y`	Array of accepted values of `X`.
`BV`	Bit vectors of length $(\texttt{NW} - 1)/32 + 1$ at least.
`NW`	Initial number of values to extract.
`NWOUT`	Current number of values left to extract.
`ISWTCH`	Switch to be set to 1 for the first call.

Linear algebra operations:

Let `H` be an `NW` × `NC` matrix. The `FUNCTION` subrogram `DOTB` is of type `REAL`.

`CALL SXPYB(NW,BV,Y,X,S)`	$\texttt{Y(BV)} = \texttt{Y(BV)} + \texttt{S} * \texttt{X(BV)}$
`CALL VXPYB(NW,BV,X,Y,V)`	$\texttt{Y(BV)} = \texttt{Y(BV)} + \texttt{V(BV)} * \texttt{X(BV)}$
`CALL SXYB(NW,BV,X,Y,S)`	$\texttt{Y(BV)} = \texttt{Y(BV)} * \texttt{V(BV)} * \texttt{S}$
`CALL XYPWZB(NW,BV,S,X,Y,T,W,Z)`	$\texttt{Y(BV)} = \texttt{S} * \texttt{X(BV)} * \texttt{Y(BV)} + \texttt{T} * \texttt{W(BV)} * \texttt{Z(BV)}$
`RES = DOTB(NW,BV,X,Y)`	$\texttt{DOTB} = \sum \texttt{X(BV)} * \texttt{Y(BV)}$
`CALL SCALB(NW,BV,Y,S)`	$\texttt{Y(BV)} = \texttt{Y(BV)} * \texttt{S}$
`CALL VSETB(NW,BV,Y,S)`	$\texttt{Y(BV)} = \texttt{S}$
`CALL COPYB(NW,BV,Y,X)`	$\texttt{Y(BV)} = \texttt{X(BV)}$

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 15.09.1978
Language : Fortran | **Revised:**

Direct or Tensor Matrix Product

Subroutine subprogram MXDIPR computes the direct (sometimes called tensor, or Kronecker) product $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ of two matrices **A** and **B**. Let $\mathbf{A} = (a_{ik})$, $(i = 1, 2, \ldots, I; k = 1, 2, \ldots K)$; $\mathbf{B} = (b_{jl})$, $(j = 1, 2, \ldots, J; l = 1, 2, \ldots, L)$; then $\mathbf{C} = (c_{ij;kl})$ with $c_{ij;kl} = a_{ik}b_{jl}$. **C** has $I \times J$ rows and $K \times L$ columns. If, in particular, **A** and **B** are square matrices, **C** is also square.

Structure:

SUBROUTINE subprogram
User Entry Names: MXDIPR

Usage:

```
CALL MXDIPR(A,B,C,IAD,JBD,IJD,IA,KA,JB,LB)
```

A,B	(REAL) Matrices A and B.
C	(REAL) On exit, C contains the direct product $\mathbf{A} \times \mathbf{B}$.
IAD	(INTEGER) First dimension of A.
JBD	(INTEGER) First dimension of B.
IJD	(INTEGER) First dimension of C.
IA,KA	(INTEGER) Number of rows, columns of **A**.
JB,LB	(INTEGER) Number of rows, columns of **B**.

Restrictions:

A, B, C must not overlap.

Error handling:

If IA or KA or JB or LB are equal to zero, the subprogram acts as do-nothing.

Examples:

```
DIMENSION A(2,2),B(2,2),C(4,4)
...
CALL MXDIPR(A,B,C,2,2,4,2,2,2,2)
```

assuming

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

would set

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix} = \begin{pmatrix} c_{11;11} & c_{11;12} & c_{11;21} & c_{11;22} \\ c_{12;11} & c_{12;12} & c_{12;21} & c_{12;22} \\ c_{21;11} & c_{21;12} & c_{21;21} & c_{21;22} \\ c_{22;11} & c_{22;12} & c_{22;21} & c_{22;22} \end{pmatrix}.$$

References:

1. E.P. Wigner, Group Theory, (Academic Press, New York 1959) 17
2. W.I. Smirnow, Lehrgang der höheren Mathematik, Vol. III.1, (Deutscher Verlag der Wissenschaften, Berlin 1954) 221

•

Author(s) : G.A. Erskine
Submitter :
Language : Fortran

Library: KERNLIB
Submitted: 01.09.1983
Revised: 27.11.1984

Banded Linear Equations

Subroutine subprograms RBEQN and DBEQN solve a system of N simultaneous linear equations with K right-hand sides, the coefficient matrix being a band matrix with bandwidth $2M+1$:

$$\sum_{j=1}^{N} a_{ij}x_{jk} = b_{ik}, \qquad (i=1,2,\ldots,N,\ k=1,2,\ldots,K); \qquad (a_{ij}=0 \text{ for } |i-j|>M).$$

Only those coefficients a_{ij} for which $|i-j| \le M$ need be supplied on entry (see **Usage**).

Structure:

SUBROUTINE subprograms
User Entry Names: RBEQN, DBEQN
Files Referenced: Printer
External References: KERMTR (N001), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tBEQN(N,M,ABAND,IDIM,IFAIL,K,B)
```

N (INTEGER) Number of equations.

M (INTEGER) Band parameter M.

ABAND (type according to t) Two-dimensional array whose first dimension has the value IDIM.

IDIM (INTEGER) First dimension of array ABAND (and of array B if K > 1).

IFAIL (INTEGER) On exit, IFAIL will be set to -1 if the coefficient matrix is singular, and to 0 otherwise.

K (INTEGER) Number of right-hand sides in array B.

B (type according to t) In general, a two-dimensional array whose first dimension has the value IDIM. B may be one-dimensional if K = 1.

On entry, ABAND must contain the packed form of the coefficient matrix as described below, and array B must contain the matrix of right-hand sides b_{ik}. Then, provided the coefficient matrix is non-singular, IFAIL will be set to 0 and the solution x_{ik} will replace b_{ik} in B. The contents of ABAND are destroyed. If the coefficient matrix is singular, IFAIL will be set to -1. In this case the contents of ABAND and B are unpredictable.

The storage convention for ABAND is that it must contain, on entry, those coefficients a_{ij} for which $|i-j| \le$ M, stored "left-justified" as an array of N rows and at most 2M + 1 columns. For example, if N = 4 and M = 1, the coefficient matrix

$$\begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix} \quad \text{is stored as} \quad \begin{pmatrix} a_{11} & a_{12} & X \\ a_{21} & a_{22} & a_{23} \\ a_{32} & a_{33} & a_{34} \\ a_{43} & a_{44} & X \end{pmatrix}$$

where X denotes elements whose value need not to be set.

If ALPHA(I,J) is a function subprogram or statement function which computes a_{ij}, the following Fortran statements will set ABAND correctly:

```
      DO 2 I =1,N
      L = 1
      DO 1 J = MAX(I-M,1),MIN(I+M,N)
      ABAND(I,L) = ALPHA(I,J)
      L = L+1
    1 CONTINUE
    2 CONTINUE
```

Method:

Gaussian elimination with row interchanges. The storage organization is as described in the reference.

Error handling:

If the integer arguments do not satisfy the conditions $1 \leq M + 1 \leq N \leq IDIM$, $K \leq 0$, a message is printed and program execution is terminated by calling ABEND (Z035).

References:

1. J.H. Wilkinson and C. Reinsch (eds.), Handbook for automatic computation, Vol.2: Linear algebra (Springer-Verlag, New York 1971) 54.

•

RLHOIN	CERN Program Library	F500

Author(s) : K.S. Kölbig, F. Schwarz **Library:** MATHLIB
Submitter : **Submitted:** 01.07.1979
Language : Fortran **Revised:** 01.12.1994

Linear Homogeneous Inequalities

Subroutine subprograms `RLHOIN` and `DLHOIN` find the basis $\mathbf{v}_j$, $(j = 1, 2, \ldots, J)$, of the convex polyhedral cone defining the solution of a system of homogeneous linear inequalities $\mathbf{Ax} \geq 0$. $\mathbf{A} = a_{mn}$ is a given $M \times N$ matrix, $M \geq N$, and rank$(\mathbf{A}) = N$. $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a column vector. Any solution **x** of $\mathbf{Ax} \geq 0$ can be expressed as

$$\mathbf{x} = \sum_{j=1}^{J} \lambda_j \mathbf{v}_j.$$

where all $\lambda_j \geq 0$. The number J of vectors $\mathbf{v}_j$ depends on the matrix **A** in an unknown way, except when $M = N$, where $J = N$.

On CDC and Cray computers, the double-precision version `DLHOIN` is not available.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `RLHOIN`, `DLHOIN`
Obsolete User Entry Names: `LIHOIN` $\equiv$ `RLHOIN`
Files Referenced: `Unit 6`
External References: `RVCPY` (F002), `RVMPY` (F002), `RVSCL` (F002),
`DVCPY` (F002), `DVMPY` (F002), `DVSCL` (F002),
`RMCPY` (F003), `RMSET` (F003), `DMCPY` (F003), `DMSET` (F003),
`RINV` (F010), `DINV` (F010), `MTLMTR` (N002), `ABEND` (Z035)

Usage:

For `t` = `R` (type `REAL`), `t` = `D` (type `DOUBLE PRECISION`),

```
CALL tLHOIN(A,MA,M,N,MAXV,V,NV,JVEC,EPS,IOUT,W,IW)
```

`A` (type according to `t`) Two-dimensional array, dimensioned (`MA`, $\geq$ `N`), whose rows contain the coefficients of the inequalities, arranged in such a way that the upper left `N` $\times$ `N` corner has a non-vanishing determinant. Usually it is advisable to normalise the rows of `A` to unity before calling this subprogram.

`MA` (`INTEGER`) First dimension parameter of `A`.

`M` (`INTEGER`) Number M of inequalities.

`N` (`INTEGER`) Number N of variables.

`MAXV` (`INTEGER`) Maximum number of basis vectors which may occur at any intermediate step, to be chosen sufficiently large and in any case $\geq$ `N`.

`V` (type according to `t`) Two-dimensional array, dimensioned (`NV`, $\geq$ `MAXV`), whose columns contain, on return, the basis vectors $\mathbf{v}_j$ of the solution cone.

`NV` (`INTEGER`) First dimension parameter of `V`($\geq$ `N`).

`JVEC` (`INTEGER`) Number J of basis vectors of the final cone.

`EPS` (type according to `t`) A small parameter which discriminates small quantities against zero, chosen to take into account the accuracy of the machine used.

IOUT (INTEGER)
$= 0$: Gives no intermediate printout,
$= 1$: Gives, for each iteration, the basis vectors of the respective cone, the matrix of scalar products and the index of the inequality taken into account in the next step.

W (type according to t) Two-dimensional array, dimensioned $(\mathtt{MAXV}, \geq \mathtt{M}+1)$, used as working space.

IW (INTEGER) Two-dimensional array, dimensioned $(\mathtt{MA}, 5)$ whose columns serve as book-keepers for certain properties of the system during the iteration procedure.

Method:

The Motzkin-Burger procedure is used to obtain the solution iteratively. Ref. 1 should be consulted before using this subprogram.

Restrictions:

The routine may fail if the matrix **A** is ”ill-conditioned” in a certain sense.

Notes:

A given system of linear homogenous inequalities may have no solution.

Error handling:

Error F500.1: MAXV too small.
Error F500.2: Upper left $\mathtt{N} \times \mathtt{N}$ corner of A is singular.
Error F500.3: Inequality k is inconsistent.
In all cases, a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. K.S. Kölbig and F. Schwarz, A program for solving systems of homogeneous linear inequalities. Computer Phys. Comm. **17** (1979) 375–382.

•

Author(s) : G. Folger, K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 21.08.1971
Revised: 15.01.1994

Upper Tail Probability of Chi-Squared Distribution

Function subprogram `PROB` computes the probability that a random variable having a χ^2-distribution with $N \geq 1$ degrees of freedom assumes a value which is larger than a given value $X \geq 0$, i.e.

$$Q(X|N) \;=\; \frac{1}{\sqrt{2^N}\,\Gamma(\frac{1}{2}N)} \int_X^{\infty} e^{-\frac{1}{2}t}\, t^{\frac{1}{2}N-1}\, dt.$$

Structure:

`FUNCTION` subprogram
User Entry Names: `PROB`
External References: `ERFC` (C300), `DERFC` (C300), `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`PROB(X,N)` has the value Q(`X`,`N`).

`PROB` and `X` are of type `REAL` and `N` is of type `INTEGER`.

Method:

See Ref. 1, formulae Nr. 26.4.4, 26.4.5 and, for $N > 300$, No. 26.4.14.

Accuracy:

For `N` $\leq$ `300`, `PROB` has an accuracy of about six digits. For `N` $>$ `300`, the accuracy decreases for `X` $>$ `N` with increasing `X`.

Error handling:

Error `G100.1`: `N` $<$ `1`.
Error `G100.2`: `X` $<$ `0`.
In both cases, the function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

References:

1. M. Abramowitz and I.A. Stegun (eds.), Handbook of mathematical functions with formulas, graphs, and mathematcal tables, 9th printing with corrections, (Dover, New York 1972).

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran
Library: MATHLIB
Submitted: 15.10.1976
Revised: 15.03.1993

Inverse of Chi-Square Distribution

Function subprogram `CHISIN` calculates $\chi^2(P,N)$ for a given probability $P(\chi^2)$ and a given degree of freedom N, where

$$P(\chi^2|N) = \frac{1}{\sqrt{2^N}\Gamma(\frac{1}{2}N)} \int_0^{\chi^2(P,N)} e^{-\frac{1}{2}t} t^{\frac{1}{2}N-1}\, dt$$

and $N \geq 1$ and $0 \leq P(\chi^2) < 1$.

Structure:

`FUNCTION` subprogram
User Entry Name: `CHISIN`
Files Referenced: `Unit 6`
External References: `GAUSIN` (G105), `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`CHISIN(P,N)` has the value χ^2(`P`,`N`),

where `CHISIN` and `P` are of type `REAL`, and `N` is of type `INTEGER`.

Method:

The method is described in Ref. 1. Note that there the complementary integral is taken.

Accuracy:

Approximately three to six digits are correct. The case $N = 3$ is the least accurate.

Error handling:

Error `G101.1`: `P` $<$ `0` or `P` $\geq$ `1`.
Error `G101.2`: `N` $<$ `1`.
In both cases, the function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Source:

This subprogram is based on an Algol60 procedure published in Ref. 1.

References:

1. R.B. Goldstein, Algorithm 451, Chi-Square Quantiles, Collected Algorithms from CACM (1972)

•

Author(s) : F. James, K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.10.1976
Language : Fortran **Revised:** 15.03.1993

Kolmogorov Distribution

Function subprogram PROBKL calculates the Kolmogorov distribution function

$$P(X) \;=\; -2\sum_{j=1}^{\infty}(-1)^j\exp(-2j^2X^2)$$

for real arguments X.

Structure:

FUNCTION subprogram
User Entry Name: PROBKL

Usage:

In any arithmetic expression,

PROBKL(X) has the value $P(\mathtt{X})$,

where PROBKL and X are of type REAL.

Method:

Direct evaluation or using functional relations.

Accuracy:

Approximately seven digits are correct. Results smaller than 10^{-40} (corresponding to $X > 6.8116$) are set to zero. Note that the above formula has a statistical meaning only for "large" $N(> 10)$.

Notes:

1. For an experimental distribution with N events and a maximum deviation ΔN from a hypothetical distribution, $P(\mathtt{X})$ with $\mathtt{X} = \Delta N\sqrt{N}$ gives the confidence level for the null hypothesis.
2. To compare two experimental distributions with M and N events, respectively, one may use $\mathtt{X} = \sqrt{MN/(M+N)}\Delta N$.

•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 15.02.1991
Language : Fortran **Revised:**

Kolmogorov Test

Subroutine subprogram `TKOLMO` tests whether two one-dimensional sets of points are compatible with coming from the same parent distribution, using the Kolmogorov test. That is, it is used to compare two experimental distributions of unbinned data.

Structure:

`SUBROUTINE` subprogram
User Entry Name: `TKOLMO`
External routine referenced: `PROBKL` (G102)

Usage:

```
CALL TKOLMO(A,NA,B,NB,PROB)
```

`A,B` (`REAL`) One-dimensional arrays of length `NA`, `NB`, respectively. The elements of `A` and `B` must be given in ascending order. (This can be accomplished, for example, by using `FLPSOR` (M103)).

`NA,NB` (`INTEGER`) The number of points in `A` and `B`, respectively.

`PROB` (`REAL`) A calculated confidence level which gives a statistical test for compatibility of `A` and `B`.

Values of `PROB` close to zero are taken as indicating a small probability of compatibility. For two point sets drawn randomly from the same parent distribution, the value of `PROB` should be uniformly distributed between zero and one.

Method:

The Kolmogorov test is used. The test statistic is the maximum deviation between the two integrated distribution functions, multiplied by the normalizing factor $\sqrt{MN/(M+N)}$, where M and N are the numbers of points in the two samples.

Accuracy:

Approximately seven digits are correct.

Notes:

Probabilities smaller than 10^{-40} are set to zero. However, the method has a statistical meaning only for "large" M and $N(>10)$.

References:

1. W.T. Eadie, D. Drijard, F.E. James, M. Roos and B. Sadoulet, Statistical Methods in Experimental Physics, (North-Holland, Amsterdam 1971) 269-271.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.02.1994
Language : Fortran **Revised:**

Student's t-Distribution and Its Inverse

Function subprogram `STUDIS` calculates the value of the Student t-distribution function

$$F(t,n) \;=\; \frac{\Gamma(\frac{1}{2}(n+1))}{\sqrt{\pi n}\,\Gamma(\frac{1}{2}n)} \int_{-\infty}^{t} \left(1+\frac{x^2}{n}\right)^{-\frac{1}{2}(n+1)} dx$$

for a given degrees of freedom $n \geq 1$.
Function subprogram `STUDIN` calculates the inverse $t(F,n)$.

Structure:

`FUNCTION` subprogram
User Entry Names: `STUDIS`, `STUDIN`
Files Referenced: Printer
External References: `GAUSIN` (G105), `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`STUDIS(T,N)` or `STUDIN(F,N)` has the value $F(\mathtt{T},\mathtt{N})$ or $t(\mathtt{F},\mathtt{N})$,

respectively. `STUDIS`, `STUDIN`, `F` and `T` are of type `REAL`, `N` is of type `INTEGER`.

Error handling:

Error `G104.1`: `N` $\leq$ `0`.
Error `G104.2`: `F` $<$ `0` or `F` $>$ `1`.
In both cases, a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Accuracy:

About six decimal places are usually correct. Accuracy is lost for `STUDIS` when `T` $<<$ `0` and `N` $>$ `4`.

Notes:

The subprograms are based on algorithms given in the references.

References:

1. B.E. Cooper, Algorithm AS3 - Applied Statistics **17** (1968) 189.
2. G.W. Hill, Algorithm 396, Student's t-quantiles, Collected algorithms from CACM (1970).

•

GAUSIN	CERN Program Library	G105

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 01.12.1988
Language : Fortran **Revised:** 15.03.1993

Inverse of Normal Frequency Function

Function subprograms GAUSIN and DGAUSN calculate the inverse $X(P)$ of the normal frequency function (Gaussian distribution)

$$P(X) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{X(P)} e^{-\frac{1}{2}t^2} dt$$

for real arguments P, where $0 < P < 1$.

Structure:

FUNCTION subprogram
User Entry Name: GAUSIN, DGAUSN
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

GAUSIN(P) has the value X(P),

where GAUSIN and P are of type REAL.

Method:

The method is described in Ref. 1.

Accuracy:

Accuracy:

GAUSIN (except on CDC and Cray computers) has an accuracy of about six digits. For most values of the argument P, DGAUSN (and GAUSIN on CDC and Cray computers) has an accuracy of approximately one significant digit less than the machine precision.

Error handling:

Error G105.1: P $\leq$ 0 or P $\geq$ 1.
The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

Source:

This subprogram is based on an Algol60 procedure published in Ref. 1.

References:

1. G.W. Hill and A.W. Davis, Algorithm 442, Normal Deviate, Collected Algorithms from CACM (1973)

•

Author(s) : K.S. Kölbig
Submitter :
Language : Fortran

Library: MATHLIB
Submitted: 01.05.1990
Revised: 15.03.1993

Gamma Distribution

Function subprogram GAMDIS calculates the gamma distribution function (incomplete gamma function)

$$P(x,a) \;=\; \frac{1}{\Gamma(a)}\int_0^x e^{-t}\,t^{a-1}\,dt$$

for real arguments $x \geq 0$ and $a > 0$.

Structure:

FUNCTION subprogram
User Entry Name: GAMDIS
Files Referenced: Unit 6
External References: GAMMA (C302), ALGAMA (C304), MTLMTR (N002), ABEND (Z035)

Usage:

In any arithmetic expression,

GAMDIS(X,A) has the value $P(\mathtt{X},\mathtt{A})$,

where GAMDIS, X and A are of type REAL.

Method:

The method is described in Ref. 1.

Accuracy:

Approximately six digits are correct.

Error handling:

Error G106.1: $\mathtt{X} < 0$ or $\mathtt{A} \leq 0$.
Error G106.2: Difficulties of convergence (unlikely).
The function value is set equal to zero, and a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

Notes:

1. For greater accuracy, or for the case $a \leq 0$, use GAPNC (C334). Note, however, that in this case the arguments X and A must be interchanged.
2. Note that, for integer $\mathtt{N} \geq 1$, $\mathtt{GAMDIS(X,N/2.)} = 1 - \mathtt{PROB}(2*\mathtt{X},\mathtt{N})$, where PROB (G100) is the upper tail probability of the chi-squared distribution function. PROB (G100) is faster than GAMDIS (G106) in this case.

Source:

This subprogram is based on a Fortran program for the incomplete gamma functions published in Ref. 2.

References:

1. W. Gautschi, A computational procedure for incomplete gamma functions, ACM Trans. Math. Software **5** (1979) 466–481.
2. W. Gautschi,Algorithm 542, Incomplete gamma functions, Collected Algorithms from CACM (1979).

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 30.08.1985
Language : Fortran **Revised:** 15.03.1993

Landau Distribution

The `LANDAU` function subprogram package contains six independent subprograms for the calculation of the following functions related to the Landau distribution:

$$\begin{array}{lll}
\text{The density} & \phi(\lambda) & = \dfrac{1}{2\pi i}\displaystyle\int_{c-i\infty}^{c+i\infty} \exp(\lambda s + s\ln s)ds, \\
\text{the distribution} & \Phi(\lambda) & = \displaystyle\int_{-\infty}^{\lambda} \phi(\lambda)d\lambda, \\
\text{the derivative} & \phi'(\lambda) & = \dfrac{d\phi(\lambda)}{d\lambda}, \\
\text{the first moment} & \Phi_1(x) & = \dfrac{1}{\Phi(x)}\displaystyle\int_{-\infty}^{x} \lambda\phi(\lambda)d\lambda, \\
\text{the second moment} & \Phi_2(x) & = \dfrac{1}{\Phi(x)}\displaystyle\int_{-\infty}^{x} \lambda^2\phi(\lambda)d\lambda, \\
\text{the inverse of } \Phi(x) & \Psi(x) & = \Phi^{-1}(x).
\end{array}$$

The function $\Psi(x)$ can be used to generate Landau random numbers (see **Usage**).

Structure:

`FUNCTION` subprograms
User Entry Names: `DENLAN`, `DISLAN`, `DIFLAN`, `XM1LAN`, `XM2LAN`, `RANLAN`
Obsolete User Entry Names: `DSTLAN` $\equiv$ `DISLAN`

Usage:

In any arithmetic expression,

`DENLAN(X)`	has the value	$\phi(\mathtt{X})$,
`DISLAN(X)`	has the value	$\Phi(\mathtt{X})$,
`DIFLAN(X)`	has the value	$\phi'(\mathtt{X})$,
`XM1LAN(X)`	has the value	$\Phi_1(\mathtt{X})$,
`XM2LAN(X)`	has the value	$\Phi_2(\mathtt{X})$,
`RANLAN(X)`	has the value	$\Psi(\mathtt{X})$,

where `DENLAN`, `DISLAN`, `DIFLAN`, `XM1LAN`, `XM2LAN`, `RANLAN` and `X` are of type `REAL`.
To generate a set of Landau random numbers, `RANLAN` should be referenced repeatedly, using as argument a random number from a uniform distribution over the interval (0,1).

Method:

Approximation by rational functions. For reason of speed, `RANLAN` proceeds mainly by table look-up and quadratic interpolation.

Accuracy:

At least six significant digits (five for `RANLAN`) are correct.

Restrictions:

1. Underflow may occur for DENLAN, DISLAN and DIFLAN if X is negative and (moderately) large.
2. No test is made whether X for RANLAN lies outside the interval (0,1), and hence no error message is printed.

Notes:

This program package is a version of the *CPC Program Library* package LANDAU (Ref. 1).

References:

1. K.S. Kölbig and B. Schorr, A program package for the Landau distribution, Computer Phys. Comm. **31** (1984) 97–111.

•

Author(s) : A. Rotondi, P. Montagna, K.S. Kölbig
Submitter : K.S. Kölbig
Library: MATHLIB
Submitted: 10.12.1993
Language : Fortran
Revised:

Approximate Vavilov Distribution and its Inverse

The `VAVLOV` package contains subprograms for fast approximate calculation of functions related to the Vavilov distribution.

For $\kappa > 0$ and $0 \le \beta^2 \le 1$, the Vavilov density function is *mathematically* defined by

$$\phi_V(\lambda;\kappa,\beta^2) \;=\; \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} e^{\lambda s}\, f(s;\kappa,\beta^2)\, ds,$$

where c is an arbitrary real constant and

$$f(s;\kappa,\beta^2) \;=\; C(\kappa,\beta^2)\,\exp\left\{ s\ln\kappa + (s+\kappa\beta^2)\left[\ln\left(\frac{s}{\kappa}\right) + E_1\left(\frac{s}{\kappa}\right)\right] - \kappa\,\exp\left(-\frac{s}{\kappa}\right)\right\}.$$

$E_1(x) = \int_0^x t^{-1}\,(1-e^{-t})\,dt$ is the exponential integral, $C(\kappa,\beta^2) = \exp\{\kappa(1+\beta^2\gamma)\}$, and $\gamma = 0.57721\ldots$ is Euler's constant.

The Vavilov distribution function is defined by

$$\Phi_V(\lambda;\kappa,\beta^2) \;=\; \int_{-\infty}^{\lambda} \phi_V(\lambda;\kappa,\beta^2)\, d\lambda$$

and its inverse by $\Psi_V(x;\kappa,\beta^2) = \Phi_V^{-1}(x;\kappa,\beta^2)$.

The function $\Psi_V(x;\kappa,\beta^2)$ can be used to generate Vavilov random numbers (see **Usage**).

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `VAVSET`, `VAVDEN`, `VAVDIS`, `VAVRND`, `VAVRAN`
External References: `LOCATR` (E106), `DENLAN` (G110), `DISLAN` (G110)
`COMMON` Block Names and Lenghts: `/G115C1/ 226`

Usage:

```
CALL VAVSET(RKAPPA,BETA2,MODE)
```

sets auxiliary quantities used in `VAVDEN`, `VAVDIS` and `VAVRND`; this call has to precede a reference to any of these entries.

`RKAPPA`	The variable κ (the straggling parameter); ($0.01 \le \kappa \le 12$).
`BETA2`	The variable β^2 (the square of velocity in unit c); ($0 \le \beta^2 \le 1$).
`MODE`	$= 1$; $=$ `0` in the particular case that `VAVDEN` only is referenced after the call to `VAVSET`.

In any arithmetic expression,

`VAVDEN(X)`	has an approximate value of	ϕ_V(`X`; `RKAPPA`, `BETA2`),
`VAVDIS(X)`	has an approximate value of	Φ_V(`X`; `RKAPPA`, `BETA2`),
`VAVRND(X)`	has an approximate value of	Ψ_V(`X`; `RKAPPA`, `BETA2`),

RKAPPA and BETA2 are defined by the last call to VAVSET prior to a reference to VAVDEN, VAVDIS, or VAVRND.

To generate a *set* of Vavilov random numbers with identical κ and β^2, VAVSET should be called once and then VAVRND be referenced repeatedly, using as argument X a random number from a uniform distribution over the interval (0,1).

In any arithmetic expression,

VAVRAN(RKAPPA,BETA2,X) has an approximate value of $\Psi_V(\mathtt{X};\mathtt{RKAPPA},\mathtt{BETA2})$.

To generate *one* Vavilov random number for given values of κ and β^2, VAVRAN should be used, using as argument X a random number from a uniform distribution over the interval (0,1).

VAVDEN, VAVDIS, VAVRND, VAVRAN and X, RKAPPA, BETA2 are of type REAL, and MODE is of type INTEGER.

Method:

The method is discribed in Ref. 1.

Accuracy:

The accuracy depends on the parameters. Although often rather poor from a mathematical point of view, it is usually sufficient for the intended application in physics (see **Notes**).

Restrictions:

No test is made whether the parameters κ and β^2 are in the specified ranges.

Notes:

1. Representing the Vavilov functions by approximations which are both fast and accurate is a difficult task. In view of the requirements in physics, speed is much more important than accuracy. This is taken into account for the present routines.
2. For a more accurate, but much slower, calculation of the Vavilov density and distribution functions, use VVILOV (G116).
3. For $\kappa \leq 0.01$, the Vavilov distribution can be replaced by the Landau distribution (LANDAU (G110)), taking into account that $\lambda_V = (\lambda_L - \ln\kappa)/\kappa$.
4. For $\kappa \geq 10$, the Vavilov distribution can be replaced by the Gaussian distribution with mean $\mu = \gamma - 1 - \beta^2 - \ln\kappa$ and variance $\sigma^2 = (2-\beta^2)/(2\kappa)$.

References:

1. A. Rotondi and P. Montagna, Fast calculation of Vavilov distribution, Nucl. Instr. and Meth. **B47** (1990) 215–224.

•

Author(s) : B. Schorr, K.S. Kölbig
Submitter : K.S. Kölbig
Library: MATHLIB
Submitted: 10.12.1993
Language : Fortran
Revised:

Vavilov Density and Distribution Functions

The `VVILOV` package contains subprograms for the calculation of the Vavilov density and distribution functions. Though generally more accurate, these routines are considerably slower than those in `VAVLOV` (G115).
For $\kappa > 0$ and $0 \leq \beta^2 \leq 1$, the Vavilov density function is *mathematically* defined by

$$\phi_V(\lambda;\kappa,\beta^2) \;=\; \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} e^{\lambda s}\, f(s;\kappa,\beta^2)\,ds,$$

where c is an arbitrary real constant and

$$f(s;\kappa,\beta^2) \;=\; C(\kappa,\beta^2)\,\exp\left\{s\ln\kappa + (s+\kappa\beta^2)\,\left[\ln\left(\frac{s}{\kappa}\right) + E_1\left(\frac{s}{\kappa}\right)\right] - \kappa\,\exp\left(-\frac{s}{\kappa}\right)\right\}.$$

$E_1(x) = \int_0^x t^{-1}\,(1-e^{-t})\,dt$ is the exponential integral, $C(\kappa,\beta^2) = \exp\{\kappa(1+\beta^2\gamma)\}$, and $\gamma = 0.57721\ldots$ is Euler's constant.
The Vavilov distribution function is defined by

$$\Phi_V(\lambda;\kappa,\beta^2) \;=\; \int_{-\infty}^{\lambda} \phi_V(\lambda;\kappa,\beta^2)\,d\lambda.$$

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `VVISET`, `VVIDEN`, `VVIDIS`
Internal Entry Names: `G116F1`, `G116F2`
External References: `RZERO` (C205), `RSININ` (C336), `RCOSIN` (C336), `REXPIN` (C337)
`COMMON` Block Names and Lenghts: `/G116C1/ 322`

Usage:

```
CALL VVISET(RKAPPA,BETA2,MODE,XL,XU)
```

sets auxiliary quantities used in `VVIDEN` and `VVIDIS`; this call has to precede a reference to either of these entries.

`RKAPPA`	The variable κ (the straggling parameter); $(0.01 \leq \kappa \leq 12)$.
`BETA2`	The variable β^2 (the square of velocity in unit c); $(0 \leq \beta^2 \leq 1)$.
`MODE`	$=$ `0` if `VVIDEN` is referenced after the call to `VVISET`; $=$ `1` if `VVIDIS` is referenced after the call to `VVISET`.
`XL,XU`	On exit, `XL` and `XU` contain a lower and upper limit as defined below.

In any arithmetic expression,

`VVIDEN(X)`	has the value	ϕ_V(`X`; `RKAPPA`, `BETA2`),
`VVIDIS(X)`	has the value	Φ_V(`X`; `RKAPPA`, `BETA2`),

By definition

$$\texttt{VVIDEN(X)} = 0 \quad \text{if} \quad \texttt{X} < \texttt{XL}; \qquad \texttt{VVIDIS(X)} = 0 \quad \text{if} \quad \texttt{X} < \texttt{XL};$$
$$\texttt{VVIDEN(X)} = 0 \quad \text{if} \quad \texttt{X} > \texttt{XU}; \qquad \texttt{VVIDIS(X)} = 1 \quad \text{if} \quad \texttt{X} > \texttt{XU}.$$

`RKAPPA`, `BETA2`, `XL` and `XU` are defined by the last call to `VVISET` (with `MODE` = `0`) prior to a reference to `VVIDEN`, or (with `MODE` = `1`) prior to a reference to `VVIDIS`.

`VVIDEN`, `VVIDIS` and `X`, `RKAPPA`, `BETA2`, `XL`, `XU` are of type `REAL`, and `MODE` is of type `INTEGER`.

Method:

The method, based on Fourier expansions, is described in Ref. 1.

Accuracy:

About five significant digits are usually accurate.

Error handling:

Error `G116.1`: $\kappa <$ `0.01` or $\kappa >$ `10`.
Error `G116.2`: $\beta^2 >$ `1`.
These errors can occur when calling `VVISET`. In both cases, a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Notes:

1. Representing the Vavilov functions by approximations which are both fast and accurate is a difficult task. These routines, though rather accurate, are slow. If speed is of higher importance than accuracy, and for calculating Vavilov random numbers, use `VAVLOV` (G115).
2. For $\kappa \le 0.01$, the Vavilov distribution can be replaced by the Landau distribution (`LANDAU` (G110)), taking into account that $\lambda_V = (\lambda_L - \ln\kappa)/\kappa$.
3. For $\kappa \ge 10$, the Vavilov distribution can be replaced by the Gaussian distribution with mean $\mu = \gamma - 1 - \beta^2 - \ln\kappa$ and variance $\sigma^2 = (2-\beta^2)/(2\kappa)$.

References:

1. B. Schorr, Programs for the Landau and the Vavilov distributions and the corresponding random numbers, Computer Phys. Comm. **7** (1974) 215–224.

•

Author(s) : CDC | **Library:** KERNLIB or Fortran intrinsic
Submitter : H. Lipps (not CDC or Cray) | **Submitted:** 02.06.1980
Language : Fortran or Assembler | **Revised:** 24.06.1985

Random Number Generator

OBSOLETE

Please note that this routine has been obsoleted in CNL 215. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement:
`RANMAR` (V113) or `RANECU` (V114) or `RANLUX` (V115)

Function subprograms `RANF` and `DRANF` return pseudo-random values uniformly distributed in the interval (0,1), the end points excluded. The multiplicative congruential method is used.

Subroutine subprogram `RANGET` makes the current seed value of `RANF` and `DRANF` available to the user, and subroutine `RANSET` restores a seed value for further use by `RANF` and `DRANF`.

On CDC computers, the subprograms other than `DRANF` are part of Control Data's Fortran execution-time library.

The non-CDC versions of `RANF` and `DRANF` use the same multiplier (`2875 A2E7 B175`), the same initial seed value (`2BC6 8CFE 166D`), and the same modulus (`2**48`). They thus generate, within the limitations of machine accuracy, the same random sequence as the CDC versions.

`DRANF` is identical to `RANF` except that it returns a function value of type `DOUBLE PRECISION`.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `RANF`, `DRANF`, `RANGET`, `RANSET`

Usage:

In any arithmetic expression,

`RANF()` or `DRANF()`

is set to a value greater than zero and less than one. `RANF` is of type `REAL`, `DRANF` is of type `DOUBLE PRECISION`.

```
CALL RANGET(SEED)
CALL RANSET(SEED)
```

`SEED` (`REAL` for CDC, `DOUBLE PRECISION` otherwise). On exit from `RANGET`,`SEED` will be set to a value that determines the current position in the sequence of random numbers. This value may be used later as an actual argument in a call to `RANSET` in order to restart the random sequence at this point.

References:

1. Fortran Version 5 Reference Manual (Control Data Corporation, 1981).

•

Author(s) : M. Gyr **Library:** MATHLIB
Submitter : K.S. Kölbig **Submitted:** 15.02.1994
Language : Fortran **Revised:**

Linear Optimization Using the Simplex Algorithm

Subroutine subprograms RSMPLX and DSMPLX calculate the quantities $x_1, x_2, \ldots, x_m$ for which the linear form, or objective function,

$$z = z_0 - \sum_{i=1}^{m} b_i x_i$$

assumes a *maximum* value subject to the n_1 inequality constraints

$$\sum_{i=1}^{m} a_{ik} x_i \le c_k \qquad (k = 1, 2, \ldots, n_1)$$

and the $n - n_1$ equality constraints

$$\sum_{i=1}^{m} a_{ik} x_i = c_k \qquad (k = n_1 + 1, n_1 + 2, \ldots, n).$$

A number $m_1 \le m$ of the variables x_i, $(i = 1, 2, \ldots, m_1)$ can be restricted to non-negative values ($x_i \ge 0$). The remaining $m - m_1$ variables x_i, $(i = m_1 + 1, \ldots, m)$ are then unrestricted ($-\infty < x_i < \infty$). In the case $m_1 = 0$, all variables x_i are unrestricted. These subprograms can also be used for the so-called degenerate case.

On computers other than CDC or Cray, only the double-precision version DSMPLX is available. On CDC and Cray computers, only the single precision version RSMPLX is available.

Structure:

SUBROUTINE subprograms
User Entry Names: RSMPLX, DSMPLX
Internal Entry Names: H101S1, H101S2
Files Referenced: Unit 6
External References: MTLMTR (N002), ABEND (Z035)

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tSMPLX(A,B,C,Z0,IDA,M,M1,N,N1,LW,IDW,W,X,Z,ITYPE)
```

A (type according to t) Two-dimensional array of dimension (IDA, $\ge$ N). Contains, on entry, the coefficients $a_{i,k}$, $(i = 1, 2, \ldots, m;\ k = 1, 2, \ldots, n)$. Destroyed during execution.

B (type according to t) One-dimensional array of dimension $\ge$ M. Contains, on entry, the coefficients b_i, $(i = 1, 2, \ldots, m)$. Destroyed during execution.

C (type according to t) One-dimensional array of dimension $\ge$ N. Contains, on entry, the coefficients c_k, $(k = 1, 2, \ldots, n)$. Destroyed during execution.

Z0 (type according to t) Contains, on entry, the initial value of the objective function.

IDA (INTEGER) Declared first dimension of A in the calling program (IDA $\ge$ M).

M (INTEGER) The total number m of variables x_i (M $\ge$ 0).

M1 (INTEGER) The number m_1 of restricted variables $x_i \geq 0$ ($0 \leq$ M1 $\leq$ M).

N (INTEGER) The total number n of constraints (N ≥ 0).

N1 (INTEGER) The number n_1 of inequality constraints ($0 \leq$ N1 $\leq$ N).

LW (INTEGER) Two-dimensional array of dimension (IDW, ≥ 5). Used as working space.

IDW (INTEGER) Declared first dimension of LW in the calling program (IDW $\geq$ M + 2 * N).

W (type according to t) One-dimensional array of dimension $\geq$ M + N. Used as working space.

X (type according to t) One-dimensional array of dimension $\geq$ M + N. If ITYPE = 1 or ITYPE = 2, its first m elements X(1), ... ,X(M) contain, on exit, the or a solution $x_1, \ldots, x_m$, respectively. The next n elements X(M+1), ... ,X(M+N) contain the values of the so-called slack variables $x_{m+1}, \ldots, x_{m+n}$. If ITYPE = 3 or ITYPE = 4, the elements X(1), ... ,X(M + N) are undefined.

Z (type according to t) If ITYPE = 1 or ITYPE = 2, Z contains, on exit, the result z of the objective function. Undefined for ITYPE = 3 and ITYPE = 4.

ITYPE (INTEGER) Defines, on exit, the type of the result:

= 1 : There is exactly one finite solution.

= 2 : There is more than one solution.

= 3 : There is no finite solution.

= 4 : There is no feasable initial solution.

Method:

The method is described in Ref. 1 and Ref. 2.

Error handling:

Error H101.1: M $\leq$ 0 or N $\leq$ 0.
Error H101.2: M1 $<$ 0 or M1 $>$ M or N1 $<$ 0 or N1 $>$ N.
In both cases, a message is written on Unit 6, unless subroutine MTLSET (N002) has been called.

References:

1. H.P. Künzi, H.G. Tzschach and C.A. Zehnder, Numerical methods of mathematical optimization, (Academic Press, New York 1968)
2. E. Stiefel, Einführung in die Numerische Mathematik, (B.G. Teubner, Stuttgart 1965)

•

Author(s) : F. Bourgeois
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 15.02.1994
Revised:

Assignment Problem

Subroutine subprogram ASSNDX solves the so-called *Assignment problem*: Given an $n \times m$ matrix A of real numbers $a(i,j)$, find either

1. a set $\{k(1), k(2), \ldots, k(n)\} \in \{1, 2, \ldots, m, 0, \ldots, 0\}$, where $0, \ldots, 0$ indicates $\max(n-m, 0)$ zeros, and where for non-zero elements $k(p) \neq k(q)$ for $p \neq q$, which minimizes

$$S = \sum_{i=1}^{n} a(i, k(i))$$

assuming that $a(i, 0) = 0$, or

2. a set $\{k(1), k(2), \ldots, k(m)\} \in \{1, 2, \ldots, n, 0, \ldots, 0\}$, where $0, \ldots, 0$ indicates $\max(m-n, 0)$ zeros, and where for non-zero elements $k(p) \neq k(q)$ for $p \neq q$, which minimizes

$$S = \sum_{j=1}^{m} a(k(j), j)$$

assuming that $a(0, j) = 0$.

Structure:

SUBROUTINE subprogram
User Entry Names: ASSNDX
Files Referenced: Unit 6

Usage:

```
CALL ASSNDX(MODE,A,N,M,IDA,K,SMIN,IW,IDW)
```

MODE	(INTEGER) Must be set either 1 (for case (1)), or 2 (for case (2)).
A	(REAL) Two-dimensional array of dimension (IDA, $\geq$ M). Must contain, on entry, the matrix A. Destroyed during execution.
N	(INTEGER) Number n of rows of A.
M	(INTEGER) Number m of columns of A.
IDA	(INTEGER) Declared first dimension of A in the calling program (IDA $\geq$ N).
K	(INTEGER) One-dimensional array of length $\geq \max$(N, M). Contains, on exit, the assigned set of integers $\{k(1), \ldots, k(n)\}$ or $\{k(1), \ldots, k(m)\}$, respectively.
SMIN	(REAL) The calculated minimum value of S.
IW	(INTEGER) Two-dimensional array of dimension (IDW, $\geq$ 6). Used as working space.
IDW	(INTEGER) Declared first dimension of IW in the calling program (IDW $\geq \max$(N, M)).

Method:

The subprogram is based on the Algol procedure given in Ref. 3.

Error handling:

Error `H301.1`: $N < 1$ or $M < 1$.
A message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

Examples:

The following example illustrates a possible use of the subprogram. A workshop has to carry out N jobs, each of which can be performed on any of $M (> N)$ available machines. The cost of performing job I on machine J is $A(I, J)$. It is required to assign jobs to machines in such a way as to minimize the total cost. The solution is obtained by calling the subprogram with $\mathtt{MODE} = 1$ and then assigning job I to machine $\mathtt{K(I)}$, $(\mathtt{I} = 1, 2, \ldots, \mathtt{N})$.

References:

1. J. Munkres, Algorithms for the assignment and transportation problems, J. SIAM 5 (1957) 32–38.
2. R. Silver, An algorithm for the assignment problem, Comm. ACM **3** (1960) 605–606.
3. R. Silver, Algorithm 27 ASSIGNMENT, Collected Algorithms from CACM (1960).

•

Author(s) : H. Grote, I. McLaren
Library: PACKLIB
Submitter :
Submitted: 01.12.1981
Language : Fortran, Assembler
Revised: 01.02.1982

EP Standard Format Input/Output Package

The EP format off-line package is intended to be used for all data (at least on tape) in an experiment, in such a way that from the raw data tape to the DST, the tape (or file) format is identical. This makes the transport of data between computers easier, and simplifies the task of passing the files or tapes at different stages of the production chain through any other part of the production chain. `EPIO` is designed to make almost all features of the very flexible EP format available to the user.

Structure:

`SUBROUTINE` package
User Entry Names: `EPINIT`, `EPREAD`, `EPOUTS`, `EPOUTL`, `EPCLOS`, `EPRWND`, `EPDROP`, `EPEND`, `EPUREF`, `EPGETW`, `EPGETA`, `EPGETC`, `EPSETW`, `EPSETA`, `EPSETC`, `EPADDH`, `EPUPDH`, `EPSTAT`
Files Referenced: User defined
External References: `UZERO` (V300), `UCOPY` (V301), `IOPACK` (Z300) (IBM only)
`COMMON` Block Names and Lengths: `/EPCOMM/ 136`

Usage:

See **Long Write-up**.
•

Author(s) : R. Brun, P. Zanarini **Library:** PACKLIB
Submitter : **Submitted:** 10.02.1988
Language : Fortran **Revised:** 17.12.1991

KUIP - Kit for a User Interface Package

The KUIP package is part of PAW (Q121) (Physics Analysis Workstation), but can be used independently. KUIP is an interface program for any application based on interactive input of commands. From the application it is seen as a slave which supplies the next command with its associated parameters. It takes care of program input in various (e.g., graphics or menu) forms and performs preliminary checking on command syntax and parameters.

Structure:

SUBROUTINE subprograms

Usage:

See **Long Write-up**.
•

Author(s) : See below
Submitter : J.C. Lassalle
Language : Fortran

Library: PACKLIB
Submitted: 30.01.1980
Revised: 17.12.1991

Format-Free Input Processing

Authors: R. Brun, R. Hagelberg, M. Hansroul, I. Ivanchenko, J.C. Lassalle, G. Misuri, J. Vorbrueggen

OBSOLETE

Please note that this routine has been obsoleted in CNL 219. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: `KUIP` (I202)

`FFREAD` provides the user with a facility for free-format data input, providing a suitable tool to transmit and/or modify variables at run-time without recompilation.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `FFREAD`, `FFINIT`, `FFSET`, `FFKEY`, `FFGO`, `FFGET`
Internal Entry Names: `FFCARD`, `FFFIND`, `FFGOR`, `FFSKIP`, `FFUPCA`
Files Referenced: Input, Output (both default or user defined)
External References: `UCOPY` (V301), `UCTOH` (M409), `UHTOC` (M409), `FFUSER` (()optionally user-supplied)

Usage:

See **Long Write-up**.
•

Author(s) : J. Zoll
Library: KERNLIB
Submitter :
Submitted: 19.09.1991
Language : Fortran
Revised:

Print Large Characters

`VIZPRI` prints one line of large characters to make banner pages. A large line occupies 12 text lines; each large character is 12 columns wide with 2 blank columns to separate.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `VIZPRI`
Files Referenced: Parameter

Usage:

```
CALL VIZPRI(LUN,CHTEXT)
```

with:

`LUN`	Fortran logical unit number for printing, if zero: use 'standard output'.
`CHTEXT`	(`CHARACTER`) text to be printed.

Examples:

```
CALL VIZPRI(0,'e=mc2')
```

gives:

```
eeeeeeeeeeee                mm        mm   cccccccccc    2222222222
eeeeeeeeeeee                mmm      mmm  cccccccccccc  222222222222
ee                          mmmm    mmmm  cc        cc  22        22
ee             ==========   mm mm  mm mm  cc                      22
ee             ==========   mm  mmmm  mm  cc                      22
eeeeeeee                    mm   mm   mm  cc                     22
eeeeeeee                    mm        mm  cc                   22
ee             ==========   mm        mm  cc                 22
ee             ==========   mm        mm  cc               22
ee                          mm        mm  cc        cc   22
eeeeeeeeeeee                mm        mm  cccccccccccc  222222222222
eeeeeeeeeeee                mm        mm   cccccccccc   222222222222
```

•

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 19.09.1991
Language : Fortran **Revised:**

Print Banner Text

XBANNER can be used to create either a banner page or to print simple banner text. For a banner page printing may be repeated to make a recto-verso page; for simple text printing is done only once without page eject. The current date and time is always printed.

Structure:

Complete program, executable module normally on /cern/pro/bin
User Entry Names: XBANNER
External References: VIZPRI (J200), DATIME (Z007)
Files Referenced: User controlled

Usage:

The command line

```
xbanner  where arg1 arg2 arg3 ...
```

prints the text strings 'arg$_i$' as large characters, normally on one line each, onto the file selected by 'where'.

'where' specifies the output file, pre–fixed by zero, one, or two control characters. If no file name is given, standard output is assumed, in which case exactly one control character, 1 or 0, must be given.
The pre-fix control characters select the following actions:

- 2 create a recto-verso banner page;
- 1 create a single banner page; page-eject is Fortran style with '1' in column 1.
- 0 print banner text only, default.
- + append to existing file.
- = overwrite file if existing.

If a file-name is given without '+' or '=' a new file (cycle) is created on the VAX, and on Unix machines '=' is assumed.

The parameters 'arg$_i$' specify the text to be printed, each 'arg$_i$' giving rise to one or more lines: Normally a parameter gives just one line. But if its first character is not alphabetic *and* equal to its last character each such character, except the first, indicates a line break.

Typing xbanner without parameters causes a display of the help information.

Examples:

```
xbanner 1         KERN UPDATE /// 1.18 APOLLO
xbanner 1=y.lis '/KERN/UPDATE/oct 89//1.18/APOLLO/'
xbanner 1+y.lis   KERN UPDATE "oct 89" // 1.18 APOLLO
```

all create a single banner page of 6 large lines; the first example prints to standard output, the other two onto file `y.lis`, either overwriting or appending. In these examples `//` causes one blank line and `///` gives 2 blank lines. Note that a blank within a parameter has to be protected so as not to break it into 2 parameters.

The next example adds one large line to `y.lis`:

```
xbanner  +y.lis /fzcopy
```

giving:

```
                        19/09/91  16.06            19/09/91  16.06

              ffffffffffff  zzzzzzzzzzzz   cccccccccc   oooooooooooo  ppppppppp
         //   ffffffffffff  zzzzzzzzzzzz  cccccccccccc  oooooooooooo  ppppppppp
        //    ff                     zz   cc         cc oo         oo pp
       //     ff                    zz    cc            oo         oo pp
      //      ff                   zz     cc            oo         oo pp
     //       ffffffff            zz      cc            oo         oo ppppppppp
    //        ffffffff           zz       cc            oo         oo ppppppppp
   //         ff                zz        cc            oo         oo pp
  //          ff               zz         cc            oo         oo pp
 //           ff              zz          cc         cc oo         oo pp
//            ff             zzzzzzzzzzz  cccccccccccc  oooooooooooo pp
              ff            zzzzzzzzzzzz   cccccccccc   oooooooooooo pp
```

•

Author(s) : F. James **Library:** KERNLIB
Submitter : **Submitted:** 01.10.1974
Language : Fortran **Revised:**

Reasonable Intervals for Histogram Binning

BINSIZ determines reasonable lower and upper limits and bin width for a histogram, given the lower and upper limits of the data and the desired maximum number of bins. The output bin width is always an integral power of $10 \times 1, 2, 2.5$ or 5, and the output lower and upper limits are the nearest multiples of the bin width containing the specified range. Another option allows the bin width to be imposed and determines only the new limits.

Structure:

SUBROUTINE subprogram
User Entry Names: BINSIZ

Usage:

```
CALL BINSIZ(AL,AH,NA,BL,BH,NB,BWID)
```

AL	(REAL) Lower limit of data to be histogrammed.
AH	(REAL) Upper limit of data to be histogrammed.
NA	(INTEGER) Maximum number of bins desired.
BL	(REAL) Lower limit determined by BINSIZ ($\mathtt{BL} \leq \mathtt{AL}$).
BH	(REAL) Upper limit determined by BINSIZ ($\mathtt{BH} \geq \mathtt{AH}$).
NB	(INTEGER) Number of bins determined by BINSIZ ($\mathtt{NA}/2 < \mathtt{NB} \leq \mathtt{NA}$).
BWID	(REAL) Bin width $(\mathtt{BH} - \mathtt{BL})/\mathtt{NB}$.

If $\mathtt{NA} = 0$ or $\mathtt{NA} = -1$, BINSIZ always makes exactly one bin.

If $\mathtt{NA} = 1$, BINSIZ takes BWID as *input* and determines only BL, BH, and NB. This is especially useful when it is desired to have the same bin width for several histograms (or for the two axes of a scatter-plot).

If $\mathtt{AL} > \mathtt{AH}$, BINSIZ takes AL to be the upper limit and AH to be the lower limit, so that in fact AL and AH may appear in any order. They are not changed by BINSIZ. If $\mathtt{AL} = \mathtt{AH}$, BINSIZ takes the lower limit as AL, and the upper limit is set to $\mathtt{AL} + 1$.

•

Author(s) : V. Berezhnoi, R. Brun, S. Nikitin, Y. Petrovykh, V. Sikolenko **Library:** PACKLIB

Submitter : R. Brun **Submitted:** 10.02.1988

Language : Fortran **Revised:**

COMIS - Compilation and Interpretation System

The `COMIS` package is part of `PAW` (Q121) (Physics Analysis Workstation), but can be used independently. It is a Fortran interpreter with which the user can interactively define, edit and execute any Fortran routines without recompiling and relinking. A small user interface system is part of `COMIS` and an interface with the local editor is also provided.

Structure:

`SUBROUTINE` subprograms

Usage:

See **Long Write-up**.

•

Author(s) : J. Zoll
Library: none
Submitter :
Submitted: 31.01.1972
Language : Fortran
Revised: 15.01.1977

Source Code Maintenance

PATCHY and the associated auxiliary programs serve in development, maintenance, and inter-computer transport of source programs. Suitably structured source files containing several versions of a given program permit code selection and code modification (down to single-statement-level) by simple control cards to YPATCHY. Compacting and structuring of card files for efficiency (YTOBIN), maintenance of compacted files at the deck level (YEDIT), creation of machine-independent, transportable files (YTOCETA) and listing of compacted files (YLIST) and others are simple auxiliary operations in this environment.

Structure:

Complete programs; executable modules exist on all machines at CERN where the CERN Program Library is installed, normally in the directory /cern/pro/bin.

User Entry Names: YPATCHY, YEDIT, YTOBIN, YTOBCD, YLIST, YTOCETA, YFRCETA, YCOMPAR, YSEARCH, YSHIFT

Usage:

See **Long Write-up** (PATCHY Reference Manual).

•

Author(s) : H. von Eicken **Library:** KERNLIB
Submitter : **Submitted:** 14.08.1985
Language : CDC: Compass, IBM: Fortran **Revised:**

Sort One-Dimensional Array

SORTZV will sort a one-dimensional array containing Hollerith or numerical integer or real information. The user may specify his own collating sequence for characters; otherwise that of the display code will be used. The array to be sorted is not changed. The output of SORTZV is an integer array containing the ordered indices indicating the order of the original array (see **Examples**).

Structure:

SUBROUTINE subprogram
User Entry Names: SORTZV

Usage:

CDC:

```
CALL SORTZV(A,INDEX,N,MODE,NWAY,NSORT,M,CARSET)
```

Others:

```
CALL SORTZV(A,INDEX,N,MODE,NWAY,NSORT)
```

A One-dimensional array of elements to be sorted.

INDEX One-dimensional array of indices. After execution it contains the indices denoting the desired order of A. On input it may contain (depending on NSORT) indices denoting which elements of A are to be sorted (see **Examples**).

N Number of words to be sorted.

MODE Type of sort required:
< 0 : Integer,
$= 0$: Hollerith,
> 0 : Real.

NWAY Order of sort:
$= 0$: Ascending order,
$\neq 0$: Descending order.

NSORT Elements to be sorted:
$= 0$: Sort the first N elements of A,
$\neq 0$: Sort N words of A as indicated by array INDEX.

M Character set to be used: (CDC only)
$= 0$: Use display code (only applicable to Hollerith sort),
$= K$: Use collating sequence specified in CARSET ($K \leq 64$).

CARSET Defines the collating sequence for a Hollerith sort. This array must be at least 64 elements in length. On entering SORTZV the K characters for which the user wishes to specify the order, must be in the first K words of CARSET (one character/word, left-adjusted and blank-filled). Any characters found during the sort which have not been defined in CARSET will be added to CARSET.

Restrictions:

The input order of equal elements is not necessarily retained. The parameters M and CARSET are only used in the CDC version.

Examples:

1. Assume the array `I` contains `0,1,-1,4,-2,0,4,5,7,8`. Then the statement

   ```
   CALL SORTZV(I,INDEX,5,-1,0,0)
   ```

   (`M` and `CARSET` omitted) sets the array `INDEX` to `5,3,1,2,4`.
2. With the same array `I` and the array `INDEX` containing `1,3,5,6,7,8`,

   ```
   CALL SORTZV(I,INDEX,6,-1,0,1)
   ```

   sets the array `INDEX` to `5,3,1,6,7,8`.

For more details, see **Long Write-up**.

Source:

Based on an Algol procedure described in Ref. 1.

References:

1. R.S. Scowen, Algorithm 271 QUICKERSORT, Collected Algorithms from CACM (1965).

•

Author(s) : H. von Eicken **Library:** KERNLIB
Submitter : **Submitted:** 15.09.1978
Language : Fortran **Revised:**

Sort One-Dimensional Array into Itself

The FLPSOR package contains two entry points for sorting a one-dimensional array, containing either floating point number or integers, into itself. The sort is done in ascending order.

Structure:

SUBROUTINE subprogram
User Entry Names: FLPSOR, INTSOR

Usage:

```
CALL FLPSOR(A,N)
```

sorts the first N elements of the REAL array A in ascending order into itself.

```
CALL INTSOR(IA,N)
```

sorts the first N elements of the INTEGER array IA in ascending order into itself.
For more details, see **Long Write-up** for SORTZV (M101).

Source:

Based on an Algol procedure described in Ref. 1.

References:

1. R.S. Scowen, Algorithm 271 QUICKERSORT, Collected Algorithms from CACM (1965).

•

Author(s) : H. Renshall
Library: KERNLIB
Submitter :
Submitted: 27.11.1984
Language : Fortran
Revised:

Sort One-Dimensional Character Array into Itself

SORCHA does a slow linear sort of a type CHARACTER array into itself in either ascending or descending order. The sort is done on any user specified substring of the elements in a CHARACTER array.

Structure:

SUBROUTINE subprogram
User Entry Names: SORCHA

Usage:

```
CALL SORCHA(A,ICH1,ICH2,NPOINT,ITYPE)
```

A (CHARACTER) One-dimensional array of dimension NPOINT to be sorted into itself. The maximum length of the elements in A is 256 characters.

ICH1 (INTEGER) Variable or constant giving the first character position in each element of A of the substring upon which the array shall be sorted. ICH1 should be 1 if the whole length of the elements of A is to be used.

ICH2 (INTEGER) Variable or constant giving the last character position analogously to ICH1 above. ICH2 should be equal to the length of the elements of A if the sort should be on the entire length of the elements of A.

NPOINT (INTEGER) Variable or constant. The first NPOINT elements of A will be sorted.

ITYPE (INTEGER) Variable or constant controlling the type of the sort. It is possible to sort in ascending or descending order; in addition it is possible to use either the Fortran collation sequence ordering via the LLE and LGE functions, or the machine internal relational sequence ordering via the LE and GE relations (see **Notes**).
= 1 : Ascending sort, i.e. A(1) will be lower than A(2), using collation sequence.
= 2 : Descending sort, i.e. A(2) will be lower than A(1), using collation sequence.
= 3 : Ascending sort, i.e. A(1) will be lower than A(2), using relational sequence.
= 4 : Descending sort, i.e. A(2) will be lower than A(1), using relational sequence.

Notes:

On the machines and compilers tested (CDC with FTN5, VAX VMS with Fortran, ND500 with FORT-5, IBM with VS-Fortran and Siemens compilers) the collating sequence orders are the same and give blank less than numbers and numbers less than letters (this matches the ASCII internal representations).

On IBM with both compilers the relational sorts give blank less than letters and letters less than numbers (the EBCDIC sequence).

On CDC, VAX and ND500 collation and relational orders are the same.

On all machines the relational sort is faster than the collation sequence sort.

•

Author(s) : F. Carminati   **Library:** KERNLIB
Submitter :   **Submitted:** 09.02.1989
Language : Fortran   **Revised:**

Sort Rows of a Matrix

SORTR re-arranges the row order of a matrix in such a way that the elements of a selected column are either in increasing or decreasing order as described. When these elements are equal, the rows are kept in their original order.

Structure:

SUBROUTINE subprogram
User Entry Names: SORTR, SORTI, SORTD
External References: VECMAN (F121), USWOP (V301) (not on all machines)

Usage:

For t = I (type INTEGER), t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL SORTt(MX,NC,NR,NCS)
```

performs an ordering operation on the matrix MX of type t, dimensioned (NC,NR), using the NCS-th element of each row as ordering criterion.

The matrix MX is stored by rows, the first element of a row following immediately after the last element of the preceding row.

Obviously, $1 \leq |NCS| \leq NC$ is a condition. If this is not met or if $NR \leq 1$, SORTX will do nothing.

If $NCS > 0$, the subroutine re-orders the rows of MX in such a way that the NCS-th element of each row is greater than or equal to the NCS-th element of the preceding row. If $NCS < 0$, the rows of MX are re-ordered in such a way that the NCS-th element of each row is smaller than or equal to the NCS-th element of the preceding row.

•

Author(s) : T. Lindelöf
Submitter : F. Carminati
Language : Fortran

Library: MATHLIB
Submitted: 15.09.1978
Revised: 09.02.1989

Sort Rows of a Matrix

SORTRQ rearranges the row order of a matrix in such a way that the elements of a selected column are either in increasing or decreasing order, as desired. Row orders are not necessarily preserved in case these elements are equal. Otherwise, SORTRQ does the same job as SORTR (M107), but SORTRQ is sometimes faster.

Structure:

SUBROUTINE subprogram
User Entry Names: SORTIQ, SORTRQ, SORTDQ
External References: USWOP (V301) (not on all machines)

Usage:

For t = I (type INTEGER), t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL SORTtQ(MX,NC,NR,NCS)
```

performs an ordering operation on the matrix MX of type t, dimensioned (NC,NR), using the NCS-th elements of each row as ordering criterion.

The matrix MX is stored by rows, the first element of a row following immediatly after the last element of the preceding row.

Obviously, $1 \leq |\mathrm{NCS}| \leq \mathrm{NC}$ is a condition. If this is not met, or if $\mathrm{NR} \leq 1$, SORTtQ will do nothing.

If $\mathrm{NCS} > 0$, SORTRQ reorders the rows of MX in such a way that the NCS-th element of each row is $\geq$ the NCS-th element of the preceding row. If $\mathrm{NCS} < 0$, the rows of MX are reordered in the strict reverse order to that for $\mathrm{NCS} > 0$.

Source:

Based on an Algol procedure described in Ref. 1.

References:

1. R.S. Scowen, Algorithm 271 QUICKERSORT, Collected Algorithms from CACM (1965).

•

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : C. Letertre **Submitted:** 01.09.1969
Language : Fortran **Revised:** 15.09.1978

Find Power-of-Ten Scale for Printing

PSCALE gives the power of ten by which it is necessary to multiply a REAL number A for the purpose of obtaining a new REAL number B having a fixed number of digits on the left of the decimal point.

Structure:

FUNCTION subprogram
User Entry Names: PSCALE

Usage:

```
FACT=PSCALE(N,NMAX,A,IDIG)
```

returns the largest N and its power FACT = 10.0**N, such that FACT*A has at most IDIG digits to the left of the decimal point. N is limited to $\leq$ NMAX, however.

Examples:

Suppose we have an array B(100), which we want to print with a FORMAT(10F10.3). Using VMAXA (F121) we find the smallest number BMAX, such that BMAX $\geq$ |B(I)| for all I. Then

```
FACT=PSCALE(N,9,BMAX,4)
```

allows us to print the vector FACT*B(I) with the above FORMAT. The following sample values of BMAX give values for FACT as indicated below:

```
      BMAX                      FACT
1234567800.                     10.0**(-6)
   1234567.8                    10.0**(-3)
      1234.5678                 1
         1.2345678              10.0**3
         0.0012345678           10.0**6
      1234.5678*10.0**(-9)      10.0**9
      1234.5678*10.0**(-12)     10.0**9
         0.0                    10.0**9
```

All FACT*BMAX but the two last ones, will be printed as 1234.567.
•

Author(s) : J. Zoll, M. Jonker, M. Roethlisberger **Library:** KERNLIB
Submitter : **Submitted:** 30.11.1986
Language : Fortran or Assembler **Revised:** 01.04.1994

Conversion To and From IEEE Number Format

These routines convert vectors of single- or double- precision numbers between the internal and the standard IEEE representations.

Structure:

SUBROUTINE subprograms
User Entry Names: IE3FOS, IE3FOD, IE3TOS, IE3TOD

Usage:

IEEE **for/to internal, single precision:**

```
CALL IE3FOS(VSINGL,VIEEES,NV,JCODE)
CALL IE3TOS(VIEEES,VSINGL,NV,JCODE)
```

VSINGL Vector of NV words with floating point numbers in internal representation.

VIEEES Vector of NV words with the same floating point number in IEEE representation.

NV Size of the vectors.

JCODE Error code returned, normally zero,otherwise VSINGL(JCODE) is the last number which had conversion problems, such as overflow and not-a-number.

IEEE **for/to internal, double precision:**

```
CALL IE3FOD(VDOUBL,VIEEED,NV,JCODE)
CALL IE3TOD(VIEEED,VDOUBL,NV,JCODE)
```

VDOUBL Vector of 2*NV words with NV double-precision floating point numbers in internal representation.

VIEED Vector of 2*NV words with the same floating point numbers in IEEE representation.

NV Size of the vectors.

JCODE Error code returned, normally zero, otherwise VDOUBL(JCODE) is the last number which had conversion problems, assuming the declaration DOUBLE PRECISION VDOUBL(NV).

Notes:

The IEEE format provides for representing exceptions, both for single and for double precision:

a) Not-a-number:	single	7F8nnnnn,
	double	7FFnnnnn....
b) Positive infinity:	single	7F800000,
	double	7FF00000....
c) Negative infinity:	single	FF800000,
	double	FFF00000....

Depending on the machine, mapping is done either naturally or artificially:

CDC	Indefinite maps to not-a-number, overflow to infinity.	
CRAY	Overflow maps to infinity, not-a-number gives overflow.	
IBM	Positive infinity	maps to internal `7FFFFFF0`,
	not-a-number	maps to internal `7FFFFFFF`.
NORD	Positive infinity	maps to internal `177...70`,
	not-a-number	maps to internal `177...77`.
VAX	Positive infinity	maps to internal `00007F81`,
	not-a-number	maps to internal `00008001`.

Underflow gives exact zero in all cases.

On the VAX: if a file has been imported from a big-endian machine, byte-inversion (see `VXINV` (M434)) has to be done before calling `IE3TOx`; similarly byte-inversion has to be done after calling `IE3FOx` and before exporting the file.

On machines where the internal representation is `IEEE` (Apollo, Sun, Silicon Graphics, etc) these routines are simple copy operations.

•

Author(s) : H. Renshall **Library:** KERNLIB
Submitter : M. Metcalf **Submitted:** 27.11.1984
Language : Fortran **Revised:** 12.03.1985

Portable Conversion Between Type CHARACTER and Type INTEGER

`CHTOI` converts between a `CHARACTER*1` value in a 95–character set and `INTEGER` values in the range 32–126 via a look-up table.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `CHTOI`, `ITOCH`

Usage:

```
CALL CHTOI(CHAR,INTGR,*label)
```

`CHAR` (`CHARACTER*1`) Variable or constant (may be a substring of a longer string) containing on input the character for which the integer equivalent is required.

`INTGR` (`INTEGER`) Variable which will contain on output the integer equivalent from a look-up table of the input character argument. A zero will be returned if the character was not found in the table.

`label` (`INTEGER`) Label of an executable statement within the calling program to which control will be transferred should the input character not be found in the table.

```
CALL ITOCH(INTGR,CHAR,*label)
```

`CHAR` (`CHARACTER*1`) variable which will contain on output the character equivalent from a look-up table of the input integer argument. A question mark will be returned if the integer is outside the range `32` – `126` inclusive.

`INTGR` (`INTEGER`) variable or constant containing on input an integer in the range `32` – `126` for which the character equivalent is required.

`label` (`INTEGER`) Label of an executable statement in the calling program to which control will be transferred should the input integer be outside the range `32` – `126`.

Method:

A look-up table containing 95 entries is mapped consecutively into integers `32` – `126`. The table is as follows:

```
 32- 47:   ! " # $ % & ' ( ) * + , - . / (32 is a blank)
 48- 57: 0 ... 9
 58- 64: : ; < = > ? @
 65- 90: A ... Z
 91- 96: [ \ ] ^ _ `
 97-122: a ... z
123-126: { | } ~
```

Restrictions:

This routine is typed in Fortran on a system which includes all the above characters. Systems with fewer characters available usually make some local translation when they read the source for example on CDC NOS/BE the lower case letters are translated to upper case. Exact reproducibility of other than the subset of characters is not guaranteed.

Notes:

These integer values are the same as for the 8-bit `ASCII` set.

•

UBUNCH	CERN Program Library	M409

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 01.03.1968
Language : Fortran or Assembler **Revised:** 09.09.1991

Concentrate and Disperse Character Strings

PARTIALLY OBSOLETE

Please note that this routine has been partially obsoleted in CNL 219. Users are advised not to use the entries refering to Hollerith any longer and to replace them in older programs. No maintenance for this part will take place and it will eventually disappear.

Suggested replacement: `CHPACK` (M432)

The concept *string of n Hollerith characters* is machine independent, but its usual representation in `Am` format (with `m` = character capacity of a machine word: `A10`, `A8`, `A6`, `A4`) is not.

Supposing any computer to have a character capacity of at least `A4`, string representations in `A4`, `A3`, `A2` or `A1` are transportable. Representations `A1` and `A4` are particularly interesting.

Fortran 77 defines a new data type `CHARACTER` though most compilers also support Hollerith strings (without a clear definition of the differences). A set of routines has been added to this package in its Fortran 77 implementation to convert between `CHARACTER` strings and Hollerith strings.

The routines `UBLOW`, `UBUNCH` and `UTRANS` work on Hollerith only **and so should be considered obsolete**, while `UCTOH`, `UCTOH1` and `UHTOC` and `UH1TOC` copy between `CHARACTER` and Hollerith. Unpredictable results will be obtained if wrong data types are passed to these routines. On most machines text strings passed in quotes are implicitly of type `CHARACTER` while a string preceeded by `nH` is not.

The routines of this package perform transformations between different representations.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `UBUNCH`, `UBLOW`, `UTRANS`, `UCTOH`, `UCTOH1`, `UHTOC`, `UH1TOC`
`COMMON` Block Names and Lengths: `/SLATE/ NI,NJ,DUMMY(38)`

Usage:

```
CALL UBLOW(IVM,IV1,NCH)
```

disperses the string of `NCH` Hollerith characters from `IVM` into `IV1`.

`IVM` Input vector, continuous string of `NCH` Hollerith characters in `Am` form (i.e. `A10`, `A8` or `A4` depending on the machine).

`IV1` Output vector, `NCH` words in `A1` form, i.e. a single Hollerith character per word with blank-fill.

`NCH` Number of Hollerith characters to be copied.

```
CALL UBUNCH(IV1,IVM,NCH)
```

concentrates the string of `NCH` Hollerith characters from `IV1` into `IVM`.

`IV1` Input vector, `NCH` words in `A1` form (one Hollerith character per word).

`IVM` Output vector, continuous string of `NCH` Hollerith characters in `Am` form (i.e. `A10`, `A8` or `A4` depending on the machine), with blank-fill of trailing characters in the last word, if any.

`NCH` Number of Hollerith characters to be copied.

```
CALL UTRANS(IVI,IVJ,NCH,I,J)
```

copies the string of `NCH` Hollerith characters from `IVI` into `IVJ`.

`IVI` Input vector of `NCH` Hollerith characters with `I` characters per machine word in Ai form. The variable `NI` in `/SLATE/` is set to the number of machine words used from `IVI`.

`IVJ` Output vector of `NCH` Hollerith characters with `J` characters per machine word in Aj form, with blank-fill. The variable `NJ` in `/SLATE/` is set to the number of machine words built in `IVJ`.

`NCH` Number of Hollerith characters to be copied.

`I,J` Number of Hollerith characters per word in `IVI` and `IVJ`. If either `I` or `J` is greater than the maximum possible number of characters storable in a machine word then this maximum is used instead.

```
CALL UCTOH(MCH,IVJ,J,NCH)
```

copies the `CHARACTER`-type string in `MCH` into Hollerith characters in `IVJ` in `Aj` form.

`MCH` Input vector of `NCH` characters, either of type `CHARACTER` or of type `INTEGER` holding Hollerith in `Am` form.

`IVJ` Output vector of `NCH` Hollerith characters with `J` characters per machine word in Aj form, with blank-fill.

`J` Number of Hollerith characters to put in each word of `IVJ`. If `J` is larger than the maximum possible number of Hollerith characters per word this maximum will be used instead.

`NCH` Number of characters to copy.

```
CALL UCTOH1(MCH,IV1,NCH)
```

disperses the `CHARACTER`–type string in `MCH` into Hollerith characters in `IV1` in `A1` form.

`MCH` Input vector of `NCH` characters, either of type `CHARACTER` or of type `INTEGER` holding Hollerith in `Am` form.

`IV1` Output vector, `NCH` words in `A1` form, i.e. a single Hollerith character per word with blank-fill.

`NCH` Total number of characters to copy.

```
CALL UHTOC(IVI,I,CHV,NCH)
```

copies the Hollerith characters in `IVI` into the `CHARACTER` variable `CHV`.

`IVI` Input vector of `NCH` Hollerith characters with `I` characters stored per word in Ai form.

`I` Number of Hollerith characters to take from each word of `IVI`. If `I` is larger than the maximum possible number of Hollerith characters per word this maximum will be used instead.

`CHV` Output variable of type `CHARACTER` to receive `NCH` characters.

`NCH` Number of characters to copy. If the `CHARACTER` variable `CHV` is of length greater than `NCH` trailing characters will not be changed.

```
CALL UH1TOC(IV1,CHV,NCH)
```

concentrates a Hollerith string in `A1` form into the `CHARACTER` variable `CHV`.

`IV1` Input vector of `NCH` words containing one Hollerith character each in A1 form.

CHV	Output variable of type CHARACTER to receive NCH characters.

NCH	Total number of characters to copy. If the variable CHV is of length greater than NCH trailing characters will not be changed.

Error handling:

NCH $\leq$ 0 acts as do-nothing.

Examples:

(b = blank).

1) `CALL UBLOW(11HABCDEFGHIJK,V,11)`

fills V: $\mathtt{V}(1) = \mathtt{1HA}, \ldots, \mathtt{V}(11) = \mathtt{1HK}$, with blank padding of each word.

2) `CALL UBUNCH(V,X,11)`

gives the inverse transformation, thus on the CDC 7600 ($\mathtt{m} = 10$):

```
X(1) = 10ABCDEFGHIJ, X(2) = 10HKbbbbbbbb
```

3) `CALL UTRANS(X,Y,11,99,4)`

copies the continuous X string to A4 representation in Y:

```
Y(1) = 4HABCD, Y(2) = 4HEFGH, Y(3) = 4HIJKb
```

with blank padding if $\mathtt{m} > 4$.

4) `CALL UTRANS(Y,X,11,4,99)`

gives the inverse of example 3) .

5) `CALL UTRANS(V,X,11,1,99)`

gives the same result as example 2), but is much slower.

6)
```
DIMENSION V(4)
CHARACTER*14 C/'THIS IS A TEST'/
CALL UCTOH(C,V,4,14)
```

copies the CHARACTER string in C into V such that

```
V(1) = 4HTHIS, V(2) = 4HbISb, V(3) = 4HAbTE, V(4) = 4HSTbb
```

7)
```
DIMENSION V(4)
CHARACTER*14 C
DATA V /14HTHIS IS A TEST/   or   DATA V /4HTHIS,4H IS ,4HA TE,2HST/
CALL UHTOC(V,4,C,14)
```

copies the Hollerith strings in V into C such that C='THIS IS A TEST'.

8)
```
DIMENSION V(4)
CHARACTER*4 C/'TEST'/
CALL UCTOH1(C,V,4)
```

copies the CHARACTER–string in C into V such that

```
V(1) = 4HTbbb, V(2) = 4HEbbb, V(3) = 4HSbbb, V(4) = 4HTbbb
```

9)
```
DIMENSION V(4)
CHARACTER*4 C
DATA V/1HT,1HE,1HS,1HT/
CALL UH1TOC(V,C,4)
```

copies the Hollerith characters in V into the CHARACTER string C such that C='TEST'.

•

Author(s) : C. Letertre, J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 28.01.1971
Language : Fortran or Assembler **Revised:** 12.06.1987

Package for Handling Bits and Bytes

This package manipulates individual bits and bytes in a word.

A *bit* in a word is specified by giving its position `J` $(= 1, 2, \ldots, 32[, \ldots, 64])$ in the word, bit `1` being the least significant bit.

A *byte* in a word is a group of `NBITS` consecutive bits. The byte is specified by giving `NBITS` and the bit position `J` of the least significant bit of the byte.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms

User Entry Names: `JBIT, SBIT0, SBIT1, SBIT, MSBIT0, MSBIT1, MSBIT, JBYT, SBYT, MSBYT, CBYT, MCBYT, JBYTET, JBYTOR, SBYTOR, MBYTOR, MBYTET, JRSBYT`

Usage:

`IX = JBIT(IW,J)`	returns `IX` = `0` or `1`, the value of bit `J` in `IW`.
`CALL SBIT0(IX,J)`	sets `0` into bit `J` of `IX`.
`CALL SBIT1(IX,J)`	sets `1` into bit `J` of `IX`.
`CALL SBIT(IA,IX,J)`	copies bit `1` of `IA` into bit `J` of `IX`.
`IX = MSBIT0(IW,J)`	returns `IW` in `IX` with bit `J` set to `0`.
`IX = MSBIT1(IW,J)`	returns `IW` in `IX` with bit `J` set to `1`.
`IX = MSBIT(IA,IW,J)`	returns `IW` in `IX` with bit `J` set to the value of bit `1` in `IA`.
`IX = JBYT(IW,J,NBITS)`	returns in `IX` right-justified the byte at `J` in `IW`.
`CALL SBYT(IA,IX,J,NBITS)`	copies the byte at `1` of `IA` into the byte at `J` of `IX`.
`IX = MSBYT(IA,IW,J,NBITS)`	returns `IW` in `IX` with the byte at `J` replaced by the byte at `1` of `IA`.
`CALL CBYT(IA,JA,IX,J,NBITS)`	copies the byte at `JA` of `IA` into the byte at `J` of `IX`.
`IX = MCBYT(IA,JA,IW,J,NBITS)`	returns `IW` in `IX` with the byte at `J` replaced by the byte at `JA` of `IA`.
`IX = JBYTET(IA,IW,J,NBITS)`	returns in `IX` the logical `AND` of `IA` and the byte at `J` of `IW` right-justified.
`IX = JBYTOR(IA,IW,J,NBITS)`	returns in `IX` the logical `OR` of `IA` and the byte at `J` of `IW` right-justified.
`CALL SBYTOR(IA,IX,J,NBITS)`	replaces the byte at `J` in `IX` by the logical `OR` of this byte and the byte at `1` of `IA`.
`IX = MBYTOR(IA,IW,J,NBITS`	returns `IW` in `IX` with the byte at `J` replaced by the logical `OR` of this byte and the byte at `1` of `IA`.
`IX = MBYTET(IA,IW,J,NBITS)`	returns `IW` in `IX` with the byte at `J` replaced by the logical `AND` of this byte and the byte at `1` of `IA`.
`IY = JRSBYT(IA,IX,J,NBITS)`	read and reset byte; equivalent to `IY = JBYT(IX,J,NBITS)` `CALL SBYT(IA,IX,J,NBITS)`.

Notes:

The subroutines

```
SBIT0   SBIT1   SBIT    SBYT    CBYT   SBYTOR
```

are duplicated by the functions

```
MSBIT0  MSBIT1  MSBIT   MSBYT   MCBYT   MBYTOR
```

to allow implementation by statement functions. Such implementations can be picked up from the ZEBRA CDE Pam-file for different machines as sequence definitions

```
Q$JBIT    :   JBIT,   JBYT
Q$SBIT    :   MSBIT0, MSBIT1, MSBIT
Q$SBYT    :   MSBYT
Q$CBYT    :   MCBYT
Q$JBYTET  :   JBYTET, JBYTOR, MBYTET, MBYTOR
```

•

Author(s) : J. Zoll, C. Letertre
Submitter :
Language : Fortran or Assembler
Library: KERNLIB
Submitted: 28.01.1971
Revised: 16.09.1991

Handling Packed Vectors of Bytes

`PACBYT` allows handling of packed vectors of bytes. Any such vector consists of a string of bytes, all of `NBITS` bits, `INWORD` of them packed into one computer word, stored from right to left.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `PKBYT`, `UPKBYT`, `JBYTPK`, `SBYTPK`
External References: `JBYT` (M421), `SBYT` (M421) (Fortran version)

Usage:

The 2–word vector `MPACK` specifies the packing parameters:

```
MPACK(1) = NBITS
MPACK(2) = INWORD
```

$\mathtt{MPACK(1)} = \mathtt{0}$ is accepted as specifying both $\mathtt{NBITS} = \mathtt{1}$ and `INWORD` equal to the number of bits per word on the given computer.

```
CALL PKBYT(IB,MX,JX,N,MPACK)
```

packs the `N`–word vector `IB` of small integers into the bytes `JX,JX+1, ... ,JX+N-1` of the byte-vector `MX`.

```
CALL UPKBYT(MA,JA,IY,N,MPACK)
```

unpacks the `N` bytes `JA,JA+1, ... ,JA+N-1` of the packed byte-vector `MA` into the vector `IY` of small integers.

```
IX = JBYTPK(MA,JA,MPACK)
```

fetches the `JA`-th byte from the packed byte-vector `MA`.

```
CALL SBYTPK(IT,MX,JX,MPACK)
```

sets the first byte from `IT` into the `JX`'th byte of the packed byte vector `MX`.

Notes:

1. These routines, and the manner of packing byte-vectors, is compatible with the routines `JBYT` and `SBYT` (M421), except that there the *location* of a byte in the word is specified, whereas here the *ordinal number* of a byte in the vector has to be given. The conversion is as follows:

   The byte with ordinal number `J` is found in word $\mathtt{JW} = (\mathtt{J} - \mathtt{1})/\mathtt{INWORD} + \mathtt{1}$,
   on byte $\mathtt{JB} = \mathtt{J} - (\mathtt{JW} - \mathtt{1}) * \mathtt{INWORD}$ starting at bit $\mathtt{L} = (\mathtt{JB} - \mathtt{1}) * \mathtt{NBITS} + \mathtt{1}$.

2. Bits and bytes are numbered from right to left within one and the same computer word; across a word boundary there is a jump from the most significant part of the current word to the least significant part of the next word.

•

Author(s) : J. Zoll, P. Rastl **Library:** KERNLIB
Submitter : **Submitted:** 28.01.1971
Language : Fortran or Assembler **Revised:** 16.09.1991

Increment a Byte of a Packed Vector

INCBYT allows incrementing a specified byte of a packed byte vector (cf. PACBYT (M422)).

Structure:

FUNCTION subprogram
User Entry Names: INCBYT

Usage:

```
LOST = INCBYT(INC,MX,JX,MPACK)
```

The 3-word vector MPACK specifies the packing parameters (much like for PACBYT (M422), but NBITS = 0 is not allowed):

MPACK(1) $\equiv$ NBITS, number of bits per byte.

MPACK(2) $\equiv$ INWORD, number of bytes per word.

MPACK(3) $\equiv$ MAXCAP, the maximum capacity of any byte, $\leq$ 2**NBITS-1.

INCBYT adds the increment INC into the JX'th byte of the packed byte-vector MX and returns any byte overflow, i.e. the part of INC which cannot be added into the byte, because it now contains MPACK(3).

•

Author(s) : CDC: J. Blake, G. Beltz, IBM: A. Berglund
Submitter :
Language : Fortran or Assembler

Library: KERNLIB
Submitted: 12.06.1972
Revised: 01.02.1982

Unpack Full Words into Bytes

BLOW converts a source array containing a record consisting of a continuous string of NBYTES bytes of NBITS bits per byte into a target array of NBYTES full words, right-adjusted with zero-fill. BLOW is the inverse of BUNCH (M436).

Structure:

SUBROUTINE subprogram
User Entry Names: BLOW
External References: UPKCH (M427)

Usage:

```
CALL BLOW(SOURCE,TARGET,NBYTES,NBITS)
```

SOURCE	Source array containing the string of NBYTES bytes.
TARGET	Target array, which must be at least NBYTES full words long.
NBYTES	Number of bytes in the source record (0 < NBYTES).
NBITS	Number of bits per byte (0 < NBITS ≤ nbpw, where nbpw = 60 on CDC and = 32 on IBM).

Restrictions:

The two arrays SOURCE and TARGET must not overlap in any way.

Error handling:

BLOW ignores calls with erroneous parameter values.

Examples:

CDC:

```
CALL BLOW(SOURCE,TARGET,200,18)
```

The array SOURCE contains a record of 200 18-bit bytes, stored contiguously in 60 60-bit words, i.e., a string of 3600 bits. After the completion of the call to BLOW, the array TARGET will contain 200 60-bit words, each containing one 18-bit byte, right-justified with zero-fill.

•

Author(s) : J. Zoll | **Library:** KERNLIB
Submitter : | **Submitted:** 01.06.1973
Language : Fortran or Assembler | **Revised:** 16.09.1991

Pack/Unpack Continuous Byte-strings

PKCHAR allows packing of integers into *continuous byte-strings* on zoned memory across word boundaries. The term *continuous byte-string* is used here to designate n-bit bytes, stored from left to right, as opposed to the objects handled by PKBYT (M422), which are stored right to left within each word. The inverse unpacking is performed by UPKCH. Leading and trailing bits of each zone can be ignored.

Structure:

SUBROUTINE subprograms
User Entry Names: PKCHAR, UPKCH
External References: JBYT (M421), SBYT (M421), CBYT (M421)
COMMON Block Names and Lengths: /SLATE/ NWU,DUMMY(39)

Usage:

```
CALL PKCHAR(INT,MPK,N,IPAR)
CALL UPKCH (MPK,INT,N,IPAR)
```

PKCHAR packs the N–word vector INT of integers into the continuous byte-string supported by the vector MPK according to the packing specifications given in IPAR.

UPKCH is the exact inverse of PKCHAR.

The packing parameters are given in the 5-element vector IPAR:

IPAR(1)	Number of bits per byte, must be ≥ 1.
IPAR(2)	Number of bytes to be used in each zone (starting with the left-most); if IPAR$(2) = 0$, the maximum possible number is used.
IPAR(3)	Number of bits per zone. If IPAR$(3) = 0$, a zone equals 1 word.
IPAR(4)	Number of leading bits of each zone to be ignored.
IPAR(5)	Each new word handled by PKCHAR is preloaded with IPAR(5).

MPK is seen as a continuous string of bits, starting with the most significant bit of MPK(1), ignoring word boundaries. This string is divided into a number of consecutive and contiguous zones, each of IPAR(3) bits; the first zone starts with the most significant bit of MPK(1). Each zone contains IPAR(4) leading bits, a number of bytes (each of IPAR(1) bits), and trailing bits, if any.

On return from either routine, NWU in COMMON block /SLATE/ indicates the number of words in MPK actually used. PKCHAR sets to IPAR(5) each word of MPK before filling it, but it does not clear any trailing unused words which logically belong to the last zone.

Examples:

1. To convert, on the CDC 7600, 6-bit Hollerith text to 7-bit ASCII–code, to be held in 36-bit words on the PDP10, with 5 characters per word.

```
DATA IPACK6 /6,0,0,0,0/
DATA IPACK7 /7,0,36,0,0/
CALL UPKCH(HOLL,INT,N,IPACK6)
```

   unpacks the Hollerith string HOLL into INT, where INT(I) is a small integer giving the display-code value of the I-th character. After conversion to ASCII, one may pack:

```
CALL PKCHAR(INT,MPK,N,IPACK7)
```

   giving the vector MPK ready to be written out. If for some reason one required the first and the last (5th) character in each 36-bit PDP10 word to be zero, one could use:

```
DATA IPACK7 /7,3,36,7,0/
```

2. To unpack 8-character bytes read with the CDC 7600 from 9-track tapes:

```
DATA IPACK /8,0,120,0,0/
CALL UPKCH(A,INT,N,IPACK)
```

3. To unpack on the CRAY 32-bit integers, read one each into one 64-bit machine word, into 16-bit integers, one each in one machine word, right-justified with zero-fill:

```
DATA IPACK /16,2,0,32,0/
CALL UPKCH(I32,I16,N,IPACK)
```

   The same operation on the Apollo, which has 32-bit words, could be done with

```
DATA IPACK /16,0,0,0,0/
```

4. The Fortran implementaion of BLOW (M426) executes:

```
IPACK(1) = NBITS
IPACK(2) = 0
IPACK(3) = NBYTES*NBITS + 127
IPACK(4) = 0
IPACK(5) = 0
CALL UPKCH(SOURCE,TARGET,NBYTES,IPACK)
```

•

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 01.06.1973
Language : Fortran or Assembler **Revised:** 15.09.1978

Search for Byte-Content

LOCBYT searches through a vector in steps of 1 or more words looking for the first word which has a certain bit configuration in a certain part of the word.

Structure:

FUNCTION subprogram
User Entry Names: LOCBYT

Usage:

```
J = LOCBYT(IT,VECT,N,INC,L,NBITS)
```

searches through the N element vector VECT, but only looking every INC words for the first word which contains IT in the byte (L,NBITS), and returns its address in J which may be 1, INC+1, 2*INC+1, 3*INC+1, etc.

IT must contain the desired byte value right-justified with zero-fill.

J = 0 is returned if such a word is not found, or if N = 0.

The byte (L,NBITS) is a byte of NBITS bits, occupying the bits L,L+1, ... ,L+NBITS-1. The bits are numbered as with the routines of BITBYT (M421) / PACBYT (M422): L = 1, 2, 3, . . .; bit 1 is the least significant bit of the word.

•

Author(s) : M. Metcalf
Library: KERNLIB
Submitter :
Submitted: 01.06.1973
Language : Assembler
Revised: 15.09.1978

Number of One-Bits in a Word

NUMBIT counts the one-bits in a word.

Structure:

FUNCTION subprogram
User Entry Names: NUMBIT

Usage:

In an arithmetic expression,

```
NUMBIT(X)
```

has the INTEGER value giving the number of one-bits in X.

Examples:

```
J=NUMBIT(5)
```

sets J to 2 as the binary representation of 5 has 2 one-bits.

•

Author(s) : M. Metcalf
Submitter :
Language : Fortran

Library: KERNLIB
Submitted: 15.01.1986
Revised: 16.05.1986

Convert Between Character String and Packed ASCII Form

`IFROMC` and `CFROMI` provide a simple, portable facility for storing character strings of 1–4 characters packed into integers.

Structure:

`FUNCTION` subprograms
User Entry Names: `IFROMC`, `FROMI`
External References: `CHTOI` (M400), `ITOCH` (M400)

Usage:

```
I=IFROMC('string')
```

stores in `I` a packed `ASCII` representation of the 4 leftmost characters of `'string'`. If there are fewer than 4 characters, blanks are stored in the empty positions.

```
CHARACTER*4 STRING
...
STRING=CFROMI(I)
```

stores in `STRING` the four characters stored packed in `I` in their `ASCII` representation.

References:

1. CERN Computer Newsletter **179** (April–May 1985) 11–14.

•

CHPACK	CERN Program Library	M432

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 02.06.1989
Language : Fortran **Revised:** 01.04.1994

Utility Routines for Character String Parsing and Construction

The routines of this package analyse and manipulate Fortran `CHARACTER` strings.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `CKRACK`, `CCOPYL`, `CCOPYR`, `CCOPIV`, `CCOSUB`, `CENVIR`, `CFILL`, `CLEFT`, `CRIGHT`, `CSQMBL`, `CSQMCH`, `CLTOU`, `CUTOL`, `CSETDI`, `CSETHI`, `CSETOI`, `CSETVI`, `CSETVM`, `CTRANS`, `ICDECI`, `ICHEXI`, `ICOCTI`, `ICEQU`, `ICFIND`, `ICFILA`, `ICFMUL`, `ICFNBL`, `ICLOC`, `ICLOCL`, `ICLOCU`, `ICLUNS`, `ICNEXT`, `ICNTH`, `ICNTHL`, `ICNTHU`, `ICINQ`, `ICINQL`, `ICINQU`, `ICNUM`, `ICNUMA`, `ICNUMU`, `ICTYPE`, `LNBLNK`, `NCDECI`, `NCHEXI`, `NCOCTI`
`COMMON` Block Names and Lengths: `/SLATE/ 40`

Summary:

`CALL CKRACK`	Read integer or real number from character
`CALL CCOPYL`	Copy string, left shift allowed if overlap
`CALL CCOPYR`	Copy string, right shift allowed if overlap
`CALL CCOPIV`	Copy string, with characters front-to-back
`CALL CCOSUB`	Copy string, replacing a token by text
`CALL CENVIR`	Copy string, replacing environment variables
`CALL CFILL`	Fill
`CALL CLEFT`	Left justify
`CALL CRIGHT`	Right justify
`CALL CSQMBL`	Squeeze multiple blanks
`CALL CSQMCH`	Squeeze multiple character
`CALL CLTOU`	Convert low to up
`CALL CUTOL`	Convert up to low
`CALL CSETDI`	Set decimal integer to character
`CALL CSETHI`	Set hexadecimal integer to character
`CALL CSETOI`	Set octal integer to character
`CALL CSETVI`	Set a vector of integers to character
`CALL CSETVM`	Set a series of generated integers to character
`CALL CTRANS`	Character translation
`IX = ICDECI`	Read decimal integer from character
`IX = ICHEXI`	Read hexadecimal integer from character
`IX = ICOCTI`	Read octal integer from character
`IX = ICEQU`	Compare two strings for equality
`JX = ICFIND`	Find first occurrence, single
`JX = ICFILA`	Find last occurrence, single
`JX = ICFMUL`	Find first occurrence, multiple
`JX = ICFNBL`	Find first non-blank

continue:	`JX = ICLOC`	Locate case sensitive
	`JX = ICLOCL`	Locate case insensitive, up to low
	`JX = ICLOCU`	Locate case insensitive, low to up
	`JX = ICLUNS`	Locate unseen characters
	`JX = ICNEXT`	Delimit next word
	`JX = ICNTH`	Identify choice case sensitive
	`JX = ICNTHL`	Identify choice case insensitive, up to low
	`JX = ICNTHU`	Identify choice case insensitive, low to up
	`JX = ICINQ`	Inquire presence in a list, case sensitive
	`JX = ICINQL`	Inquire presence in a list, case insensitive, up to low
	`JX = ICINQU`	Inquire presence in a list, case insensitive, low to up
	`JX = ICNUM`	Verify numeric
	`JX = ICNUMA`	Verify alpha-numeric
	`JX = ICNUMU`	Verify alpha-numeric or underscore
	`JX = ICTYPE`	Identify type
	`NX = LNBLNK`	Find last non-blank character
	`IX = NCDECI`	Read decimal integer from character
	`IX = NCHEXI`	Read hexadecimal integer from character
	`IX = NCOCTI`	Read octal integer from character

Usage:

General Remarks:

For what follows, let the `CHARACTER` variable `LINE` contain a string of n characters assuming the following declaration:

```
CHARACTER LINE*(n),COL(n)*1
EQUIVALENCE(LINE,COL)
```

thus `COL(j)` is the `j`-th character `LINE(j:j)`. A sub-string of `LINE` is specified by `JL` and `JR`, where `COL(JL)` is the first or left-most, and `COL(JR)` is the last or right-most character.

```
COMMON /SLATE/ ND,NE,NF,NG,NUM(2),DUMMY(34)
```

returns certain search parameters, which are set by some of the routines.

The types of all variables and functions follow from the Fortran default typing convention, except that `LINE`, `COL`, and variables starting with the letters `CH` are of type `CHARACTER`.

Convention

Typing rules for data to be interpreted by `CKRACK`:

`Binary:`	String of zeros or ones prefixed by `#B` or `#b`.
`Octal:`	String of octal digits prefixed by `#O` or `#0` or `#o`.
`Hex:`	String of hexadecimal digits prefixed by `#X` or `#x`.
`Integer:`	String of decimal digits optionally prefixed by `+` or `-`.
`Real:`	`[+|-] [int] [.]  [fract] [E] [+|-] [exp]` `int, fract, exp` are strings of decimal digits, either the decimal dot or the letter `E` (or `e`) must be present.
`Double:`	`[+|-] [int] [.]  [fract] D [+|-] [exp]` the letter `D` (or `d`) must be present.

Read integer or real number from character:

```
CALL CKRACK(LINE,JL,JR,IFLD)
```

reads the number whose character representation starts with the first non-blank character at or after `COL(JL)` and ends at `COL(JR)` or at the first blank after the number (*normal termination*), or at the first character after the number which cannot be part of it (*special termination*).

`CKRACK` detects the type of the number (bit-pattern, integer, real single, real double) from its representation. The typing rules for data to be interpreted by `CKRACK` are given in the note on the previous page.

The number read is returned in `/SLATE/` in `NUM(1)` or `ANUM(1)` or `DNUM`, for which one will need:

```
REAL ANUM(2)
DOUBLE PRECISION DNUM
EQUIVALENCE (ANUM(1),NUM(1)),(DNUM,NUM(1))
```

The flag in the last parameter is normally given as zero; $\texttt{IFLD} > 0$ demands that single-precision real numbers be handled and returned as double precision numbers; $\texttt{IFLD} < 0$ demands that double-precision numbers be returned in single precision.

Apart from `NUM`, the following parameters are returned in `/SLATE/`:

`ND`	Number of numeric digits seen.
`COL(NE)`	Terminating character in the field; $\texttt{NE} = \texttt{JR} + 1$ if terminated by end-of-field.
`NF`	Type of the number read: < 0 : error code for bad data; $= 0$: the whole field is blank; $= 1$: bit pattern (binary, octal, or hexadecimal); $= 2$: integer $= 3$: single precision real; $= 4$: double precision real.
`NG`	$= 0$ for normal termination; special termination otherwise.

Copy string, left shift allowed if overlap:

```
CALL CCOPYL (CHFROM,CHTO,NCH)
```

copies `NCH` characters from `CHFROM(1:NCH)` to `CHTO(1:NCH)`; the characters are copied in order, thus the end of the target may overlap the beginning of the source.

Copy string, right shift allowed if overlap:

```
CALL CCOPYR (CHFROM,CHTO,NCH)
```

copies `NCH` characters from `CHFROM(1:NCH)` to `CHTO(1:NCH)`; the characters are copied in reverse order, thus the beginning of the target may overlap the end of the source. These two routines are useful to copy strings from or into a very large array of type `CHARACTER*1`.

Copy string, with characters front-to-back:

```
CALL CCOPIV(CHFROM,CHTO,NCH)
```

copies `NCH` characters from `CHFROM(1:NCH)` to `CHTO(1:NCH)` inverting the order of the characters such that the last becomes the first, etc.

Copy string, replacing a token by text:

```
CALL CCOSUB(CHFROM,NFR,LINE,JL,JR,CHTOKEN,CHSUB)
```

copies `CHFROM(1:NFR)` to `LINE` starting at `COL(JL)` and not going beyond `COL(JR)`, substituting each occurrence of `CHTOKEN` by `CHSUB`.
The following parameters are returned in `/SLATE/`:

`ND`	Number of characters stored;
`COL(NE)`	is the first character after the last stored;
`NF`	Non-zero if `LINE(JL:JR)` too small to receive the complete copy;
`NG`	Zero if no substitution was done, i.e. `CHTOKEN` did not occur.

Copy string, replacing environment variables:

```
CALL CENVIR(CHFROM,NFR,LINE,JL,JR,IFLAG)
```

copies `CHFROM(1:NFR)` to `LINE` starting at `COL(JL)` and not going beyond `COL(JR)`, substituting each occurrence of `${name}` by the value of the environment variable `"name"` obtained by calling `GETENVF` (Z 265); on machines running UNIX the form `"$name"` is also recognized. The handling of undefined environment variables is defined by `IFLAG`: if zero the string `${name}` is skipped from the copy; if non-zero the string is copied through as is.
The following parameters are returned in `/SLATE/`:

`ND`	Number of characters stored;
`COL(NE)`	is the first character after the last stored;
`NF`	Bit 1 is set if undefined env/v have been encountered;
	Bit 2 is set if syntax error (missing closing bracket);
	Bit 3 is set if `LINE(JL:JR)` is too small to receive the copy;
`NG`	Zero if no substitution was done.

Fill:

```
CALL CFILL(CHI,LINE,JL,JR)
```

fills `LINE(JL:JR)` with as many copies of `CHI` as possible; if $\mathtt{JL} + 1 - \mathtt{JR}$ is not a multiple of `LEN(CHI)` as many characters of `CHI` as necessary to fill up to `JR` will be taken for the last copy.

Left justify:

```
CALL CLEFT(LINE,JL,JR)
```

left-justifies `LINE(JL:JR)` squeezing blanks to the right.

`ND`	Number of non-blank characters in the field.
`COL(NE)`	First blank character after left-justifying (or $\mathtt{NE} = \mathtt{JR} + 1$ if there are no blanks).

Right justify:

```
CALL CRIGHT(LINE,JL,JR)
```

right-justifies `LINE(JL:JR)` squeezing blanks to the left.

`ND`	Number of non-blank characters in the field.
`COL(NE)`	Last blank character after right-justifying (or $\mathtt{NE} = \mathtt{JL} - 1$ if there are no blanks).

Squeeze multiple blanks:

```
CALL CSQMBL(LINE,JL,JR)
```

left-justifies `LINE(JL:JR)` replacing any string of several consecutive blanks by a single blank.

`ND`	Number of characters retained (vacated trailing characters, if any, are blanked).
`COL(NE)`	First blank character after (or `NE` = `JR` + `1` if there are no multiple blanks).

Squeeze multiple characters:

```
CALL CSQMCH(CHIS,LINE,JL,JR)
```

left-justifies `LINE(JL:JR)` reducing any multiple occurrence of the character `CHIS` to this character just once. `CHIS` is of type `CHARACTER*1`.

`ND`	Number of characters retained (vacated trailing characters, if any, are blanked).
`COL(NE)`	First character after the squeezed string (or `NE` = `JR` + `1` if there are no multiple occurrences).

Convert low to up:

```
CALL CLTOU(LINE(JL:JR))
```

converts lower case letters in `LINE(JL:JR)` to upper case.

Convert up to low:

```
CALL CUTOL(LINE(JL:JR))
```

converts upper case letters in `LINE(JL:JR)` to lower case.

Set decimal integer to character:

```
CALL CSETDI(INT,LINE,JL,JR)
```

writes the integer `INT` into `LINE(JL:JR)` right-justified. If `INT` is too large, the most significant characters are lost. Unused positions are not cleared to blank, so that they may be pre-loaded with default characters, such as leading zeros. (One normally clears the whole of `LINE` initially with `LINE =' '`, one could clear the substring with `LINE(JL:JR)=' '` or preset it before calling `CSETDI`).

`ND`	Number of digits which have been set.
`COL(NE+1)`	Most significant digit set.
`COL(NF+1)`	Most significant character set. `NF` = `NE` if `INT` is positive, `NF` = `NE` − `1` if `INT` is negative and no overflow.
`NG`	= `0` normally, non-zero if field too small.

Set hexadecimal integer to character:

```
CALL CSETHI(INT,LINE,JL,JR)
```

acts like `CSETDI`, except that the hexadecimal rather than the decimal representation of `INT` is stored.

Set octal integer to character:

```
CALL CSETOI(INT,LINE,JL,JR)
```

acts like `CSETDI`, except that the octal rather than the decimal representation of `INT` is stored.

Set a vector of integers to character:

```
CALL CSETVI(NI,INTV,NBIAS,LINE,JL,JR,NCOL,IFLSQ)
```

sets the NI integers INTV(J) + NBIAS into LINE(JL:JR) in decimal representation, every NCOL columns, each right-justified within its field of NCOL − 1 columns; squeeze multiple blanks to single blanks in the resulting LINE(JL:JR) if IFLSQ non-zero. Like the other CSETxx routines, this routine does not clear unused positions to blank.

COL(NE)	Last character of the last integer stored.
NG	= 0 normally, = N > 0 if there is not enough room to store INTV(N).

Set a series of generated integers to character:

```
CALL CSETVM(NI,INC,IGO,LINE,JL,JR,NCOL,IFLSQ)
```

acts like CSETVI, but the NI integers are IGO + $n *$ INC, $n = 0, 1, \ldots,$ NI − 1.

Character translation:

```
CALL CTRANS(CHO,CHN,LINE,JL,JR)
```

replaces each occurrence in LINE(JL:JR) of the character CHO by the character CHN. CHO and CHN are of type CHARACTER*1.

Read decimal integer from character:

```
IX = ICDECI(LINE,JL,JR)
```

reads the decimal integer whose character representation starts at COL(JL) and stops on the first non-numeric character or at COL(JR+1), returning its value in IX. Leading blanks are ignored, a leading minus or plus sign is recognized. Note that a blank after the number, or after '+' or '-', is taken as terminator.

ND	Number of digits read ('-' or '+' do not count).
COL(NE)	Terminating character in the field; NE = JR + 1 if pure numeric or if the whole field is blank (in which case ND = 0).
NG	= 0 if the number is terminated by 'blank' or by end-of-field, non-zero otherwise.

Read hexadecimal integer from character:

```
IX = ICHEXI(LINE,JL,JR)
```

acts like ICDECI, but reads a hexadecimal rather than a decimal representation.

Read octal integer from character:

```
IX = ICOCTI(LINE,JL,JR)
```

acts like ICDECI, but reads an octal rather than a decimal representation.

Compare two strings for equality:

```
IX = ICEQU(CHA,CHB,N)
```

checks that CHA(1:N) and CHB(1:N) are identical and returns zero if so, otherwise the ordinal number of the first non-matching character is returned.

Note: this and many other routines of this package are handy when manipulating text stored in an area declared with CHARACTER TEXT(big)*1, which will explain some of the maybe unexpected calling sequences.

Find first occurrence, single:

```
JX = ICFIND(CHIS,LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first occurrence of the single character `CHIS` in `LINE(JL:JR)`.

`JX` $=$ `JR` $+$ `1` if `CHIS` is not contained in `LINE(JL:JR)`, or if `JL` $>$ `JR`.

`NG` $=$ `0` if not found, $=$ `JX` otherwise.

Find last occurrence, single:

```
JX = ICFILA(CHIS,LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the last occurrence of the single character `CHIS` in `LINE(JL:JR)`.

`JX` $=$ `JR` $+$ `1` if `CHIS` is not contained in `LINE(JL:JR)`, or if `JL` $>$ `JR`.

`NG` $=$ `0` if not found, $=$ `JX` otherwise.

Find first occurrence, multiple:

```
JX = ICFMUL(CHI,LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first occurrence in `LINE(JL:JR)` of any one of the characters `CHI(j:j)`, where `j` $= 1, 2, \ldots,$ `n` $=$ `LEN(CHI)`.

`JX` $=$ `JR` $+$ `1` if none of `CHI` is found in `LINE(JL:JR)`, or if `JL` $>$ `JR`.

`ND` $= j$, i.e. `COL(JX)` is `CHI(j:j)` if found.

`NG` $=$ `0` if not found, $=$ `JX` otherwise.

Find first non-blank:

```
JX = ICFNBL(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first non-blank character in `LINE(JL:JR)`.

`JX` $=$ `JR` $+$ `1` if `LINE(JL:JR)` is all blank, or if `JL` $>$ `JR`.

`NG` $=$ `0` if all blank, $=$ `JX` otherwise.

Locate, case sensitive:

```
JX = ICLOC(CHI,NI,LINE,JL,JR)
```

locates the first occurrence of the complete string `CHI(1:NI)` within `LINE(JL:JR)`, it returns in `JX` the position in `LINE` of the first character of the string found. `JX` $=$ `0` if `CHI` is not contained in `LINE(JL:JR)`.

Locate, case insensitive, up to low:

```
JX = ICLOCL(CHI,NI,LINE,JL,JR)
```

acts like `ICLOC`, but upper case characters from `LINE` are converted to lower case for the comparison.

Locate, case insensitive, low to up:

```
JX = ICLOCU(CHI,NI,LINE,JL,JR)
```

acts like `ICLOC`, but lower case characters from `LINE` are converted to upper case for the comparison.

Locate unseen characters:

```
JX = ICLUNS(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first 'unseen' character in `LINE(JL:JR)`, i.e. any character which will not show on the terminal, except 'blank'. `JX` $=$ `0` if `LINE(JL:JR)` does not contain unseen characters.

Delimit next word:

```
JX = ICNEXT(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first non-blank character in `LINE(JL:JR)` and in `NE` the position of the first blank character after `COL(JX)`, if any.

`JX` Position of the first character of the 'word'.

`NE` Position of the first 'blank' after the 'word' or `NE` = `JR` + `1`.

`ND` Number of characters in the 'word'.
`JX` = `NE` = `JR` + `1`, `ND` = `0` if `LINE(JL:JR)` is all blank.

Identify choice, case sensitive:

```
JX = ICNTH(CHACT,CHPOSS,NPOSS)
```

compares the character string `CHACT` against the strings stored in the character array `CHPOSS(NPOSS)`, and returns in `JX` the ordinal number of the first match found, or `JX` = `0` if no match. Neither the strings of `CHPOSS` nor of `CHACT` may contain embedded blanks: the first blank, if any, is the string terminator.
To allow matching a shortened key-word given in `CHACT` one may insert (à la VAX) a `'*'` in the text of `CHPOSS(J)` to mark the separation between the obligatory and further possible characters; a second `'*'` may be given to signal that `CHACT` may have any other characters beyond this point, this is implied if the string in `CHPOSS(J)` is not blank terminated.
For example:

```
    PARAMETER  (NPOSS=6)
    CHARACTER*8 CHPOSS(NPOSS)
    DATA CHPOSS /'del*ete ', 'add     ', 'adb*efor',
   +             'rep*lace', 'ch*ange ', 'c*ol*   '/
```

Calling the above with the following strings will give these results:

```
CHACT = 'add'       JX = 2   exact match
        'delete'         1   exact match
        'del'            1   short match
        'del  '          1
        'delphi'         0   wrong optional characters
        'deleted'        0   CHPOSS(1) is terminated
        'replaced'       4   CHPOSS(4) is not terminated
        'chan'           5   short match
        'channel'        0   wrong optional characters
        'c'              6   short match
        'columns'        6   abritrary trailing characters allowed
        'cols'           6
```

Identify choice, case insensitive, up to low:

```
JX = ICNTHL(CHACT,CHPOSS,NPOSS)
```

acts like `ICNTH` converting upper case characters from `CHACT` to lower case for the comparison, hence the `CHPOSS` array must be given in lower case.

Identify choice, case insensitive, low to up:

```
JX = ICNTHU(CHACT,CHPOSS,NPOSS)
```

acts like `ICNTH` converting lower case characters from `CHACT` to upper case for the comparison, hence the `CHPOSS` array must be given in upper case.

Inquire presence in a list, case sensitive:

```
JX = ICINQ(CHLOOK,CHHAVE,NHAVE)
```

like ICNTH this compares the character string CHLOOK against the strings stored in the character array CHHAVE(NHAVE), and returns in JX the ordinal number of the first match found, or JX = 0 if no match. Again, neither the strings of CHHAVE nor of CHLOOK may contain embedded blanks: the first blank, if any, is the string terminator.

As opposed to ICNTH, a '*' may be given in CHLOOK, but not in CHHAVE(J), to allow wild-card checking on the presence of a string in the list of CHHAVE(J). The '*' divides the string into the characters which must be present in the looked-for object of CHHAVE(J), and additional restricting characters which can be absent, but if present they must be right. Again a second '*' can be given in CHLOOK, but this is not useful, since any characters beyond the string terminator both in CHLOOK and in CHHAVE(J) are assumed to be allowed anyway, unlike as with ICNTH.

For example:

```
      PARAMETER  (NHAVE=7)
      CHARACTER*8 CHHAVE(NHAVE)
      DATA CHHAVE /'apo     ', 'apol    ', 'apollo  ', 'irs6000 ',
     +            'decra1  ', 'decra2  ', 'decra3  '/
```

Calling the above with the following strings will give these results:

```
CHLOOK = 'apo'        JX = 1
         'apo*'            1
         'ap*ollo'         1
         'ap*'             1
         'ap'              0
         'apol'            2
         'apoll'           0
         'apoll*'          3
         'ir*s60'          4
         'ir*s70'          0
         'dec*'            5
         'dec*ra'          5
         'dec*ra*'         5
         'dec*ra3'         7
```

In spite of the similarity, the operations of ICINQ and ICNTH serve really very different functions:

With ICNTH we have a key word CHACT which we try to identify; CHPOSS(N) is most likely a fixed table built into the program which gives the possible key words and allowed abbreviations à la VAX. The return value from ICNTH tells us which key word we have.

With ICINQ we inspect a table CHHAVE(N), which most likely has been built up at execution time, to see whether it contains an object according to the specifications given in CHLOOK. The interesting thing about the return value from ICINQ is mainly whether it is zero or not, the position of the found object in the table is of secondary importance.

Inquire presence in a list, case insensitive, up to low:

```
JX = ICINQL(CHLOOK,CHHAVE,NHAVE)
```

acts like ICINQ converting upper case characters from CHLOOK to lower case for the comparison, hence CHHAVE must be held in lower case.

Inquire presence in a list, case insensitive, low to up:

```
JX = ICINQU(CHLOOK,CHHAVE,NHAVE)
```

acts like `ICINQ` converting lower case characters from `CHLOOK` to upper case for the comparison, hence `CHHAVE` must be held in upper case.

Verify numeric:

```
JX = ICNUM(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first non-numeric character in `LINE(JL:JR)`; blanks are ignored. Note that `'+'`, `'-'` or `'.'` are not considered numeric.

`JX` = `JR` + `1` if `LINE(JL:JR)` is all numeric.

`ND` Number of digits seen in `LINE(JL:JX-1)`.

`NG` = `0` if all numeric, = `JX` otherwise.

Verify alpha-numeric:

```
JX = ICNUMA(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first non-alphanumeric character in `LINE(JL:JR)`; blanks are ignored. Note that `'+'`, `'-'` or `'.'` are not considered alpha-numeric.

`JX` = `JR` + `1` if `LINE(JL:JR)` is all alpha-numeric.

`ND` Number of alpha-numeric characters seen in `LINE(JL:JX-1)`.

`NE` Position of the first numeric character, = `0` if none.

`NF` Position of the first alphabetic character, = `0` if none.

`NG` = `0` if all alpha-numeric, = `JX` otherwise.

Verify alpha-numeric or underscore:

```
JX = ICNUMU(LINE,JL,JR)
```

acts like `ICNUMA`, but the character "underscore" is considered alphabetic.

Identify type:

```
JX = ICTYPE(CHIS)
```

returns in `JX` the type of the single character `CHIS`:

`JX` = `0` : *Unseen*, i.e. a character not showing on an ASCII terminal.
= `1` : Anything else.
= `2` : Numeric character.
= `3` : Lower case character.
= `4` : Upper case character.

Find last non-blank character:

```
NX = LNBLNK(CHV)
```

returns the non-blank length of the string in `CHV(1:LEN(CHV))`, i.e. the characters `NX+1` to `LEN(CHV)` are all blank. (Note that this is an intrinsic function of several compilers.) If there are many trailing blanks the routine `LENOCC` of `M507` is faster; depending on the machine the break-even point with `LENOCC` is around 25 trailing blanks.

Read decimal integer from character:

```
IX = NCDECI(CHTEXT)
```

acts like `ICDECI`, with `JL` = `1` and `JR` = `LEN(CHTEXT)`.

Read hexadecimal integer from character:

```
IX = NCHEXI(CHTEXT)
```

acts like `ICHEXI`, with `JL` = `1` and `JR` = `LEN(CHTEXT)`.

Read octal integer from character:

```
IX = NCOCTI(CHTEXT)
```

acts like `ICOCTI`, with `JL` = `1` and `JR` = `LEN(CHTEXT)`.

•

Author(s) : M. Goossens, A. Petrilli, M. Marquina **Library:** KERNLIB
Submitter : **Submitted:** 11.02.1986
Language : Fortran **Revised:** 28.09.94

Utility Package for Character Manipulation

M433 is a comprehensive package for the manipulation of type CHARACTER strings.

Structure:

SUBROUTINE and FUNCTION subprograms
User Entry Names: INDEXA, INDEXB, INDEXC, INDEXN, INDEXS, INDXAC, INDXBC, INDXNC, ISCAN, REPEAT, SPACES, STRIP, SUBWORD, VERIFY, WORD, WORDS, WORDSEP

Usage:

In what follows, the parameters STR, SSTR, SET, the functions REPEAT, SPACES and the variables CHD, CHOPT and H are of type CHARACTER. The function VERIFY is of type INTEGER.

```
I = INDEXA(STR)
```

sets I equal to the position of the first alphabetic character (upper or lower case) in STR. I = 0 if no such character is present.

```
I = INDEXB(STR,SSTR)
```

sets I equal to the position of the first occurrence of string SSTR in string STR scanning backwards. I = 0 if no such string is present.

```
I = INDEXC(STR,SSTR)
```

sets I equal to the leftmost position where string SSTR does not match a substring in STR. I = 0 if there is no such mismatch.

```
I = INDEXN(STR)
```

sets I equal to the position of the first numeric character in STR. I = 0 if no such character is present.

```
I = INDEXS(STR)
```

sets I to the position of the first special (i.e. non-alphanumeric) character in STR. I = 0 if no such character is present.

```
I = INDXAC(STR)
```

sets I equal to the position of the first non-alphabetic character (upper or lower case) in STR. I = 0 if no such character is present.

```
I = INDXBC(STR,SSTR)
```

sets `I` equal to the position of the first mismatch of string `SSTR` with respect to string `STR` scanning backwards. `I` = `0` if there is no such mismatch.

```
I = INDXNC(STR)
```

sets `I` equal to the position of the first non-numeric character in `STR`. `I` = `0` if no such character is present.

```
I = ISCAN(STR,SET)
```

sets `I` to the leftmost position where any of the characters in `SET` matches a character in `STR`. `I` = `0` if there is no such match.

```
H = REPEAT(STR,NTIMES)
```

sets `H` equal to `NTIMES` concatenated copies of the string `STR`.

```
H = SPACES(STR,NSPACE)
```

sets `H` equal to a character string equivalent to `STR` with leading blanks removed and each occurence of one or more blanks inside `STR` replaced by `NSPACE` blanks.

```
H = STRIP(STR,CHOPT,CHD)
```

sets `H` to a character string equivalent to `STR` with leading and trailing occurances of the character `CHD` removed. If `CHOPT` is equal to `'L'`, only leading characters will be removed. If `CHOPT` is equal to `'T'`, only trailing characters will be removed.

```
H = SUBWORD(STR,IW,NW)
```

sets `H` equal to the character string starting with word `IW` of `STR` and containing `NW` words.

```
I = VERIFY(STR,SET)
```

sets `I` to the leftmost position of any character in `STR` which is not part of `SET`.

```
H = WORD(STR,IW)
```

sets `H` equal to the word `IW` of `STR`.

```
I = WORDS(STR)
```

sets `I` to the number of words in `STR`.

```
CALL WORDSEP(STR)
```

sets the word separator for `SUBWORD`, `WORD` and `WORDS` to the first character of the string `STR`.

Examples:

Assume the following declarations:

```
CHARACTER STR*41,REP10*10,REP17*17
CHARACTER REPEAT*16,SPAC17*17,SPAC30*30,SPACES*20
INTEGER VERIFY
```

and a string `STR` defined as:

```
DATA STR /'A B C 1 2 3 A B C 1 2 3 A B C 1 2 3 A B C'/
```

The following results are obtained:

Statement/ Expression	Yields the value
`REP10 = REPEAT('ABC',5)`	`'ABCABCABCA'`
`REP17 = REPEAT('ABC',5)`	`'ABCABCABCABCABC  '`
`REP17 = REPEAT('ABC',6)`	`'ABCABCABCABCABCAB'`

```
INDEXB(STR,' ')                40
INDEXC(STR,' ')                 1
INDXBC(STR,' ')                41
INDEXA(STR)                     1
INDXAC(STR)                     2
INDEXN(STR)                     7
INDXNC(STR)                     1
INDEXS(STR)                     2
ISCAN(STR,' ')                  2
VERIFY(STR,' ')                 1

INDEXB(STR,'1 2 3')            31
INDEXC(STR,'1 2 3')             1
INDXBC(STR,'1 2 3')            37
ISCAN(STR,'123')                7
VERIFY(STR,'123')               1
INDEXB(STR,'A B C')            31
INDEXC(STR,'A B C')             2
INDXBC(STR,'A B C')            36
ISCAN(STR,'ABC')                1
VERIFY(STR,'ABC')               2

SPAC17=SPACES(STR,0)            'ABC123ABC123ABC12'
SPAC30=SPACES(STR,2)            'A  B  C  1  2  3  A  B  C  1  2'
```

•

VXINV	CERN Program Library	M434

Author(s) : F. Carminati, M. Jonker, J. Zoll **Library:** KERNLIB, VAX and DECSTATION only
Submitter : **Submitted:** 05.10.1987
Language : Fortran or Assembler **Revised:**

Fast VAX Byte Inversion

These routines do VAX byte inversions `1-2-3-4` to `4-3-2-1` in each word of an array, either in-place or copied.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `VXINVB`, `VXINVC`

Usage:

```
CALL VXINVB(IXV,N)
```

inverts four bytes in each of the `N words` at array `IXV`, in-place.

```
CALL VXINVC(IV,IXV,N)
```

copies the `N` words at array `IV` to array `IXV`, with the bytes inverted in each word.

On DEC machines bytes read from a disk file are loaded in memory in reverse order. One of the above routines, applied to the result of a binary read from a disk file, causes the bytes to be stored in each 32 bits word in the same order than in the disk file. This is useful when reading a binary file transferred through a network from a foreign system, in order to preserve the order of the bytes in each 32 bits word. Please note that several network utilities include the possibility to perform a bytes inversion in the network protocol. Note also that when reading or writing from a magnetic tape, the bytes may be swapped in pairs and not in groups of 4.

•

Author(s) : CDC: J. Blake, IBM: A.Berglund
Library: KERNLIB
Submitter :
Submitted: 20.10.1975
Language : Fortran or Assembler
Revised: 01.02.1982

Pack Bytes into Full Words

BUNCH converts a source array containing NBYTES bytes of NBITS bits per byte (where each byte is stored right-adjusted in a full word), into a target array in which the bytes follow each other contiguously without intermediate padding. The last word of the target array, if incomplete, is however padded with binary zero. BUNCH is the inverse of BLOW (M426).

Structure:

SUBROUTINE subprogram
User Entry Names: BUNCH
External References: PKCHAR (M427)

Usage:

```
CALL BUNCH(SOURCE,TARGET,NBYTES,NBITS)
```

SOURCE Source array containing NBYTES bytes, each right-adjusted in a full word.

TARGET Target array, which must be at least NBYTES * NBITS/nbpw (rounded up to an integral value) words long, where $\texttt{nbpw} = 60$ on CDC and $\texttt{nbpw} = 32$ on IBM.

NBYTES Number of bytes in the source array ($\texttt{NBYTES} > 0$).

NBITS Number of bits per byte ($0 < \texttt{NBITS} \leq \texttt{nbpw}$).

Restrictions:

The arrays SOURCE and TARGET must not overlap in any way.

Error handling:

BUNCH ignores calls with erroneous parameter values.

Examples:

IBM:

```
CALL BUNCH(SOURCE,TARGET,200,16)
```

The array SOURCE contains 200 words, each containing an 16-bit byte, right-adjusted. After returning from BUNCH, the array TARGET will contain 100 32-bit words, in which the 200 16-bit bytes are stored contiguously.

•

Author(s) : R. Matthews
Submitter : H. Grote
Language : Assembler

Library: KERNLIB
Submitted: 01.07.1979
Revised:

Set or Retrieve a Bit in a String

GETBIT/SETBIT find or set the value of a single bit in a bit-string which may extend across word boundaries.

Structure:

SUBROUTINE subprogram
User Entry Names: GETBIT, SETBIT

Usage:

```
CALL GETBIT(I,M,L)
CALL SETBIT(I,M,L)
```

I Position of the selected bit, starting on the left with 1.

M A word or an array, considered as a continuous string of bits.

L Integer whose right-most bit will contain the value found by GETBIT or the value to be set by SETBIT in the I-th position of the bit-string starting at the left-most bit of the first word of M.

•

BTMOVE	CERN Program Library	M438

Author(s) : H. Grote **Library:** KERNLIB
Submitter : **Submitted:** 01.12.1980
Language : CDC: Fortran and Compass, IBM: Assembler **Revised:**

Move Bit String

BTMOVE moves a contiguous string of N bits from any position in memory to any other position. Bits are numbered from left to right (most significant to least significant within words) and may be across word boundaries.

Structure:

SUBROUTINE subprogram
User Entry Names: BTMOVE
External References: UCOPY (V301) (CDC only)

Usage:

```
CALL BTMOVE(SOURCE,ISRC,TARGET,ITGT,N)
```

moves the string of N contiguous bits starting at position ISRC in word or array SOURCE to position ITGT in word or array TARGET. The other bits in TARGET are not changed, nor is SOURCE.

Notes:

Source and target strings must not overlap in storage, else the results may be unpredictable.

Examples:

IBM:
Move the highest bit (sign-bit) in word A to the lowest position of I so that it can be treated as an integer:

```
REAL A
INTEGER*4 I
I=0
CALL BTMOVE(A,1,I,32,1)
```

CDC:
Pack the five integers of array I5(5) into one word IPACK, using 12 bits per packed integer:

```
  DIMENSION I5(5)
  IPOS=1
  DO 1 I = 1,5
  CALL BTMOVE(I5(I),49,IPACK,IPOS,12)
1 IPOS=IPOS+12
```

Move a string of 20 characters from positions 41-60 in array A to positions 7-26 in array B:

```
DIMENSION A(6),B(3)
CALL BTMOVE(A,241,B,37,120)
```

•

GETBYT	CERN Program Library	M439

Author(s) : T. Lindelöf, R. Matthews, A. Shevel **Library:** KERNLIB
Submitter : T. Lindelöf **Submitted:** 01.07.1979
Language : Assembler **Revised:**

Set or Retrieve a Bit String

`GETBYT` extracts and right-adjusts a group of bits of any length up to a full word from a bit string which may extend across word boundaries. `SETBYT` is the inverse of `GETBYT`.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `GETBYT`, `SETBYT`
Internal Entry Names: `SHRERR`
Files Referenced: Printer

Usage:

```
CALL GETBYT(ADDR,IBEG,ILEN,IRES)
```

`ADDR` Name of an array containing the desired group of bits.

`IBEG` The bit position within `ADDR` of the left-most bit of the group (bits are numbered starting at 1 with the left-most or most significant bit in `ADDR(1)`).

`ILEN` Length of the group in bits (at most one word).

`IRES` Will contain the desired group, right-justified and zero-filled.

```
CALL SETBYT(ADDR,IBEG,ILEN,IBYT)
```

causes the `ILEN` right-most bits of `IBYT` to replace the group of bits of length `ILEN` starting at the `IBEG`-th bit in the array `ADDR` (bits are numbered starting at 1 with the left-most or most significant bit in `ADDR(1)`). Replacement goes across word boundaries, i.e. the most significant (left-most) bit of `ADDR(N+1)` is adjacent to the least significant (right-most) bit of `ADDR(N)`.

Error handling:

Calling either `GETBYT` or `SETBYT` with `IBEG` $<$ `1` or `ILEN` $>$ the number of bits in one word (errors) will result in a diagnostic message. After more than 20 such errors the job will come to a `STOP`.

Examples:

IBM:
If `ADDR(1)` and `ADDR(2)` contain the 32-bit configurations `'0...001110001'` and `'110100...0'` respectively, then

```
CALL GETBYT(ADDR,27,10,IRES)
```

will set `IRES` to `'0...001100011101'` or decimal `797`.
If `IBYT` contains the integer value 3 (binary `'11'`) and `ADDR(1)` = `ADDR(2)` = `0`, then

```
CALL SETBYT(ADDR,32,2,IBYT)
```

will set `ADDR(1)` to `0...001'` and `ADDR(2)` to `'100...0'`.
•

Author(s) : M. Metcalf **Library:** KERNLIB or Fortran library
Submitter : **Submitted:** 10.12.1984
Language : Fortran with ISA extensions **Revised:** 18.10.1985

Handling Bits and Bytes, Bit Zero the Least Significant

BITPAK handles bits and bytes in a single word, with bit zero being the least significant bit.

Structure:

SUBROUTINE and FUNCTION subprograms
User Entry Names: IOR, IAND, NOT, IEOR, ISHFT, ISHFTC, IBITS, MVBITS, BTEST, IBSET, IBCLR

Usage:

A numeric storage unit is considered to consist of a string of bits numbered from *right to left*, starting at *zero*. The standard MIL-STD-l753 defines ll bit manipulation functions on such units, 8 of which are the ANSI/ISA functions found as intrinsic functions in many compilers. This package complements the functions available in compilers, ensuring that the full range is available on all machines. This description includes all the functions for the sake of completeness.

Logical operations:

IOR(M,N)	provides the inclusive OR of the two integer arguments.
IAND(M,N)	provides the logical AND of the two integer arguments.
NOT(M)	provides the logical complement of the integer argument.
IEOR(M,N)	provides the exclusive OR of the two integer arguments.

Shift operations:

$$\text{A shift count K specifies} \begin{cases} \text{a left shift} & \text{for } K > 0 \\ \text{no shift} & \text{for } K = 0 \\ \text{a right shift} & \text{for } K < 0. \end{cases}$$

ISHFT(M,K)	provides the value of the integer argument M with the bits shifted. Bits shifted out to the left or right are lost, and zeros are shifted in from the opposite end.
ISHFTC(M,K,IC)	provides the value of the integer argument M with the rightmost IC bits shifted, and the remaining bits untouched. The shift is circular; no bits are lost.

Bit subfields:

IBITS(M,I,LEN)	provides, right justified, the value of the LEN bits of the integer argument M, starting from position I.
CALL MVBITS(M,I,LEN,N,J)	moves LEN bits of integer argument M, starting at position I, to the integer argument N, starting at position J. All other bits of N are left untouched. The arguments M and N may refer to the same numeric storage unit.

Bit testing:

`BTEST(N,I)`	has the value `.TRUE.` if bit `I` of the integer argument `N` is set, and `.FALSE.` otherwise. Note that many compilers require `BTEST` to be declared type `LOGICAL`.
`IBSET(N,I)`	has the value of the integer argument `N` with bit `I` set to `1`.
`IBCLR(N,I)`	has the value of the integer argument `N` with bit `I` set to `0`.

Notes:

If bits are specified outside the range of one numeric storage unit, or if fields are specified which are longer than one numeric storage unit or zero, or if shifts are specified which are longer than the fields being shifted, then the results are undefined.

•

Author(s) : J. Shiers
Library: KERNLIB
Submitter :
Submitted: 25.07.91
Language : Fortran
Revised:

Fortran Emulation of VM/CMS NAMEFIND Command

NAMEFD is a Fortran callable routine providing an emulation of the VM/CMS NAMEFIND command.

Structure:

SUBROUTINE subprogram
User Entry Names: NAMEFD

Usage:

```
CHARACTER*255 CHIN(NIN),CHOUT(NOUT)
CALL NAMEFD(LUN,CHFILE,CHIN,NIN,CHOUT,NOUT,IRC)
```

NAMEFD scans the specified file for entries that match the specified input tags and values. It returns the values of the specified output tags. Thus, given the example file shown below, one might call NAMEFD with input tag :NICK, value SNIFFLES and output tags :PHONE and :ADDRESS. If no match is found for the specified input, a code IRC is returned.

```
CHIN(1,1)  = ':NICK'
CHIN(2,1)  = 'SNIFFLES'
NIN        = 1

CHOUT(1,1) = ':PHONE'
CHOUT(1,2) = ':ADDRESS'
NOUT       = 2

CALL NAMEFD(1,'TEST.NAMES',CHIN,NIN,CHOUT,NOUT,IRC)
```

Return codes:	32	- no match for input tags and values,
	4	- not all requested output tags found,
	other	- IOSTAT from FORTRAN OPEN of specified names file.

Format of a Names File

A names file is a collection of entries, with each entry identied by a *nickname*. A nickname tag plus a series of other tags with associated values make up an entry.
The format of data lines in a names file is as follows:

```
tag.value (:tag.value...)
```

The only tag that is required is a :NICK tag, e.g.

```
:NICK.fatuser
```

This is the primary tag, one for each entry. It identifies the beginning of an entry and must be the first word on a line. Any tags that follow relate to the preceding :NICK tag.

Examples:

An example of a `NAMES` file.

```
:nick.SNOW      :userid.SNOWHITE :node.FOREST
                :name.Snow White              :phone.ZZZ-ZZZZ
                :addr.Forest Primeval
:nick.SNOOZY    :userid.SNOOZY   :node.COTTAGE
                :name.I. M. Dozing            :phone.777-7777
                :addr.Dwarf Cottage;Forest
:nick.DUMMY     :userid.DUMMY    :node.COTTAGE
                :name.S. A. What              :phone.777-7777
                :addr.Dwarf Cottage;Forest
:nick.BOSS      :userid.BOSS     :node.COTTAGE
                :name.T.O.P. Banana           :phone.777-7777
                :addr.Dwarf Cottage;Forest
:nick.SNIFFLES :userid.SNIFFLES :node.COTTAGE
                :name.A. H. Choo              :phone.777-7777
                :addr.Dwarf Cottage;Forest
:nick.GROUCHY   :userid.GROUCHY  :node.COTTAGE
                :name.E. B. Scrooge           :phone.777-7777
                :addr.Dwarf Cottage;Forest
:nick.SMILEY    :userid.SMILEY   :node.COTTAGE
                :name.H. A. Haas              :phone.777-7777
                :addr.Dwarf Cottage;Forest
:nick.WISTFUL   :userid.WISTFUL  :node.COTTAGE
                :name.R. U. Shy               :phone.777-7777
                :addr.Dwarf Cottage;Forest
:nick.WITCH     :userid.QUEEN    :node.CASTLE
                :name.Bad Queen               :phone.UGLY-1111
                :addr.Vanity Lane;Mirror City
:nick.GORGEOUS :userid.PRINCE   :node.ATLARGE :notebook.PRIVATE
                :name.Prince Charming         :phone.Area 111 111-1111
:nick.DWARFS
                :list.SNOOZY DUMMY BOSS SMILEY GROUCHY SNIFFLES WISTFUL
```

•

Author(s) : C. Letertre **Library:** KERNLIB
Submitter : **Submitted:** 21.08.1971
Language : Fortran or Assembler **Revised:** 15.09.1978

Locating a String of Same Words

IUSAME locates the first of a continuous sequence of identical words occuring at least a given number of times. It returns the number of contiguous identical words in the sequence.

Structure:

FUNCTION subprogram
User Entry Names: IUSAME

Usage:

```
NSAME = IUSAME(VECT,JL,JR,MIN,JSAME)
```

VECT(JL)	Start of the portion of the vector to be analysed.
VECT(JR)	End of the portion of the vector to be analysed.
MIN	Minimum length of a string to be considered a string.

The function returns the length of the string as function value, and also the position of the first element of the string: VECT(JSAME).

If no string of at least MIN elements has been found starting at or after VECT(JL), the function returns NSAME $=$ 0 and JSAME $=$ JR $+$ 1.

•

Author(s) : J. Zoll, P. Rastl **Library:** KERNLIB
Submitter : **Submitted:** 21.09.1971
Language : Fortran or Assembler **Revised:** 16.09.1991

Decoding Options Characters

UOPTC and UOPT compare a string of *actual* option-characters against a similar string of *possible* option-characters filling an INTEGER vector with 1's and 0's, indicating for each possible option whether or not it was taken.

Structure:

SUBROUTINE subprogram
User Entry Names: UOPTC, UOPT

Usage:

```
CALL UOPTC(CHACT,CHPOSS,IOPT)
```

CHACT (CHARACTER) String of actual option-characters.

CHPOSS (CHARACTER) String of possible option-characters.

IOPT (INTEGER) Vector of at least LEN(CHPOSS) words, the j-th word of which is set to 1 or 0, depending on whether the j-th possible character does or does not occur in CHACT.

```
CALL UOPT(IACT,IPOSS,IOPT,N)
```

IACT Hollerith string of actual option-characters. It is terminated by the first character not occuring in the string of possibilities.

IPOSS Hollerith string of N possible option-characters ($N \leq 30$).

IOPT A vector of at least N words, the j-th word of which is set to 1 or 0, depending on whether the j-th possible character does or does not occur in the IACT string.

Examples:

```
CALL UOPTC('+AB','ABC+/Y',IOPT)
CALL UOPT (4H+AB.,6HABC+/Y,IOPT,6)
```

will set the first 6 elements of IOPT to 1,1,0,1,0,0.

Notes:

UOPT was written for Fortran 4 and should no longer be used for new programs.

•

Author(s) : M. Metcalf, R. Matthews
Library: KERNLIB
Submitter :
Submitted: 01.02.1982
Language : Fortran or Assembler
Revised: 20.06.1985

Locate the One-Bits of a Word or an Array

UBITS locates and counts the 1-bits in the right-most NBITS bits in a word or full-word array, returning their positions. Bit numbering is right to left, bit number 1 being the least significant bit in the first full word, bit number NBPW+1 being the least significant bit in the second full word, where NBPW is the number of bits per machine word.

Structure:

SUBROUTINE subprogram
User Entry Names: UBITS
External References: UPKBYT (M422) (Fortran version only)

Usage:

```
    CALL UBITS(IWORDS,NBITS,IXV,NX)
```

IWORDS	Word or full-word array to be analysed.
NBITS	Bits 1 to NBITS of array IWORDS are inspected.
IXV	Bit positions of the 1-bits in IWORD are placed into IXV(1) through IXV(NX) in increasing order. IXV must be dimensioned to NBITS at least.
NX	Number of 1-bits found.

Examples:

```
     DIMENSION IXV(9)
     IWORD=1676
C    1676 in base 2 is 11010001100
     CALL UBITS(IWORD,9,IXV,NX)
```

sets

```
   NX = 3, IXV(1) = 3, IXV(2) = 4, IXV(3) = 8.
```

•

Author(s) : F. Rademakers, J. Zoll
Library: KERNLIB
Submitter :
Submitted: 27.11.1984
Language : Fortran or C
Revised: 05.05.1992

Occupied Length of a Character String

LENOCC returns the occupied length of a string of type CHARACTER.

Structure:

FUNCTION subprogram
User Entry Names: LENOCC

Usage:

In any arithmetic expression,

```
LENOCC(LINE)
```

has the value of the occupied length of the character string LINE, i.e. the length up to and including the last non-blank character. LENOCC = 0 if LINE contains blanks only. LINE is of type CHARACTER and LENOCC is of type INTEGER.

For few trailing blanks LENOCC is slower than LNBLNK of M432, but it may be substantially faster for very many trailing blanks; the break-even point depends on the machine and is usually around 25 trailing blanks.

Method:

On some machines LINE is first scanned backwards for machine words containing all blanks, and then the remaining string is scanned for the last non-blank character.

•

Author(s) : M. Metcalf, R. Matthews **Library:** KERNLIB
Submitter : **Submitted:** 01.02.1982
Language : Fortran and CDC: COMPASS, IBM: Assembler **Revised:** 20.06.1985

Find One-Bits in a String

BITPOS locates and counts the 1-bits in the right-most NBITS bits in a word or in a full-word array, returning their positions. Bit numbering is right-to-left, bit number 0 being the least significant bit in the first full word, bit number NBPW being the least significant bit in the second full word etc., where NBPW is the number of bits per machine word; this numbering is compatible with BITPAK (M441).

Structure:

SUBROUTINE subprogram
User Entry Names: BITPOS
External Entry Names: URKBYT (M422) (Fortran only)
COMMON Block Names and Lengths: /SLATE/ 40 (Fortran only)

Usage:

```
    CALL BITPOS(IWORDS,NBITS,IXV,NX)
```

IWORDS	Word or full-word array to be analysed.
NBITS	The first NBITS of array IWORDS are inspected.
IXV	Bit positions of the 1-bits in IWORD are placed into IXV(1) through IXV(NX) in increasing order. IXV must be dimensioned to NBITS at least. The positions are numbered from 0.
NX	Number of 1-bits found.

Notes:

The Fortran version contains a symbolic constant whose value must be set equal to the number of bits in a word (default 32).

Examples:

```
      DIMENSION IXV(9)
      IWORD = 1676
C     1676 in base 2 is 11010001100
      CALL BITPOS(IWORD,9,IXV,NX)
```

sets

```
      NX = 3, IXV(1) = 2, IXV(2) = 3, IXV(3) = 7.
```

•

Author(s) : H. Lipps
Submitter :
Language : Fortran

Library: KERNLIB
Submitted: 22.10.1984
Revised: 15.03.1993

Error Processing for Sections A-H of KERNLIB

PARTIALLY OBSOLETE

Please note that, as a consequence of transferring subprograms from KERNLIB to MATHLIB, this routine has been partially obsoleted in CNL 211. It can, for a transitional period, still be used for sections D (`D509` only), and for sections E and F of KERNLIB. Users are advised not to use it any longer for other cases and to replace it in older programs. With the foreseen transfer of the subroutines in sections D,E,F in KERNLIB to MATHLIB, it will eventually disappear.

Suggested replacement: `MTLSET` (N002)

Subroutine `KERSET` allows the user to redefine the action to be taken by subprograms in the Fortran version of sections A-H of KERNLIB when certain specified error conditions are detected. (This subroutine does not exist in the Fortran 66 version.) Error recovery may be performed either on each occurrence of the error, or only a specified number of times. Messages may be written either on each occurrence of the error, or only a specified number of times. Error messages may be written (by default) onto the system output unit, or may be re-routed to some other output file.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `KERSET`
Internal Entry Names: `KERMTR`
Files Referenced: Printer or user-defined
External References: `ABEND` (Z035)

Usage:

```
CALL KERSET(ER,LGFILE,LM,LR)
```

`ER` (`CHARACTER*6`) A character string that identifies the range of error conditions for which action is to be redefined.

`LGFILE` (`INTEGER`) The logical unit number to be used for error messages, or zero if error messages are to be written onto the system output unit.

`LM` (`INTEGER`) The number of occurrences of each error condition in the range `ER` for which an error message is to be written. $\mathtt{LM} \leq 0$ is treated as zero, $\mathtt{LM} \geq 100$ as infinity.

`LR` (`INTEGER`) The number of times that error recovery is to be performed for each error condition in the range `ER`. $\mathtt{LR} \leq 0$ is treated as zero. $\mathtt{LR} \geq 100$ is treated as infinity. If any error condition in the range `ER` occurs $\mathtt{LR}+1$ times a message is printed and the run is terminated by calling `ABEND` (Z035).

Notes:

1. KERSET applies to those KERNLIB error conditions which are specified by a six-character code (e.g., C204.2) in the **Error handling** section of the Short Write-ups.
2. If the string ER consists of six characters specifying a single error condition (e.g., ER='C204.2'), LM and LR apply only to this one error condition.

   If the six-character string ER ends with one or more blanks, LM and LR apply to all error conditions whose leftmost characters match the non-blank characters of ER.
   Thus ER = 'C2   ' (four blanks) applies to all error conditions in packages C200 to C299, and ER = '   ' (six blanks) applies to all error conditions under the control of KERSET.
3. The value of LGFILE applies to all error messages written under the control of KERSET.

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:** 15.03.1993

Error Processing for MATHLIB

Subroutine MTLSET allows the user to redefine the action to be taken by **certain** subprograms in MATHLIB when certain specified error conditions are detected. Error recovery may be performed either on each occurrence of the error, or only a specified number of times. Messages may be written either on each occurrence of the error, or only a specified number of times. Error messages may be written (by default) onto the system output unit, or may be re-routed to some other output file.

Structure:

SUBROUTINE subprogram
User Entry Names: MTLSET
Internal Entry Names: MTLMTR
Files Referenced: Printer or user-defined
External References: ABEND (Z035)

Usage:

```
CALL MTLSET(ER,LGFILE,LM,LR)
```

ER (CHARACTER*6) A character string that identifies the range of error conditions for which action is to be redefined.

LGFILE (INTEGER) The logical unit number to be used for error messages, or zero if error messages are to be written onto the system output unit.

LM (INTEGER) The number of occurrences of each error condition in the range ER for which an error message is to be written. LM $<$ 0 is ignored, LM $\geq$ 255 is treated as infinity.

LR (INTEGER) The number of times that error recovery is to be performed for each error condition in the range ER. LR $<$ 0 is ignored, LR $\geq$ 255 is treated as infinity. If any error condition in the range ER occurs LR + 1 times a message is printed and the run is terminated by calling ABEND (Z035).

Notes:

1. MTLSET applies to those MATHLIB error conditions which are specified by a six-character code (e.g. C204.2) in the **Error handling** section of the Short Write-ups.
2. If the string ER consists of six characters specifying a single error condition (e.g., ER='C204.2'), LM and LR apply only to this one error condition.

   If the six-character string ER ends with one or more blanks, LM and LR apply to all error conditions whose leftmost characters match the non-blank characters of ER.
   Thus ER = 'C2   ' (four blanks) applies to all error conditions in packages C200 to C299, and ER = '   ' (six blanks) applies to all error conditions under the control of MTLSET.
3. The value of LGFILE applies to all error messages written under the control of MTLSET.

•

Author(s) : CDC **Library:** KERNLIB
Submitter : J.Zoll **Submitted:** 01.03.1968
Language : Fortran or Assembler or C **Revised:** 16.09.1991

Address of a Variable

The function `LOCB` returns the absolute address of the variable given as its argument.
`LOCF` returns the absolute address measured in terms of Fortran machine words.

Structure:

`FUNCTION` subprogram
User Entry Names: `LOCF`, `LOCB`

Usage:

```
IAD = LOCB(X)
```

where `X` is the name of a variable of any type, or a name declared `EXTERNAL` in the calling program.

```
IAD = LOCF(X)
```

where `X` is the name of a variable of type `INTEGER` or `REAL`.

Notes:

On CDC, `LOCF` is included in the FTN library, and documented in the Fortran manual.
On all machines `LOCF` is intended to measure the displacement between variables, thus for example for:

```
COMMON /X/ M(12),A(4),LAST
N = LOCF(LAST) - LOCF(M(1))
```

`N` will be set to contain 16 on all machines, whilst `LOCB(LAST) - LOCB(M(1))` will give some multiple of 16.

•

Author(s) : C. Letertre
Submitter : J. Zoll
Language : Fortran or Assembler

Library: KERNLIB
Submitted: 01.09.1969
Revised: 15.09.1991

Detect Indefinite and Infinite in an Array

IUWEED scans a vector and returns the address of the first quantity which is either 'indefinite' or 'infinite'.

Structure:

FUNCTION subprogram
User Entry Names: IUWEED

Usage:

```
IW = IUWEED(IVEC,N)
```

sets IW to the relative address, in the N element vector IVEC, of the first element containing either an 'indefinite' or 'infinite'. IW = 0 if there are no such elements. IVEC is not changed.

•

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 01.12. 1973
Language : Fortran **Revised:** 15.09.1978

Print Trace-Back

TRACEQ prints the Fortran trace-back leading to TRACEQ. The maximum number of trace-back levels is specified as an argument. Fewer levels may be printed either because the main program has been reached or because the trace-back linkage is invalid.

Structure:

SUBROUTINE subprogram
User Entry Names: TRACEQ
Internal Entry Names: TRAC1Q, TRAC2Q
Files Referenced: User defined
COMMON Block Names and Lengths: /SLATE/ 40

Usage:

```
    CALL TRACEQ(LUN,N)
```

LUN Logical unit number of the print file, LUN = 0 is accepted to mean the standard print file.

N Maximum number of trace-back levels to be printed.

Notes:

The implementation of TRACEQ depends on the machine; on some machines this cannot be done at all and the routine is a dummy. On some other machines the unit for printing or the number of levels printed is not under program control.

•

Author(s) : C. Letertre, J. Zoll
Submitter : C. Letertre
Language : Fortran

Library: KERNLIB
Submitted: 31.01.1972
Revised: 15.09.1978

Memory Dump

TCDUMP may be used for dumping sections of memory in octal (CDC) or hexadecimal (IBM), optionally combined with any or all of the other modes (INTEGER, REAL, or Hollerith).

The dump shows 5 words per line. The address of the first word of each line is given 3 times. The absolute address in memory (using LOCF), the relative address within the vector in decimal, and in octal (CDC) or hexadecimal (IBM).

Continous strings of identical content or strings of *preset indefinites* produce a single line.

Structure:

SUBROUTINE subprogram
User Entry Names: TCDUMP
Files Referenced: Printer
External References: UBLOW (M409), IUCOMP (V304), IUSAME (M501), LOCF (N100)

Usage:

```
CALL TCDUMP(TEXT,VECTOR,N,MODE)
```

TEXT	1 word of text printed as heading.
VECTOR	Variable address for start of dump.
N	Number of words for dumping.
MODE	1H dump in octal, 1HI dump in INTEGER and octal, 1HF dump in floating and octal, 1HH dump in Hollerith and octal, 2HIH dump in INTEGER, Hollerith and octal, etc...

Examples:

```
COMMON /TOC /A,B(12),D
CALL TCDUMP(5H/TOC/,A,14,1HF)
```

dumps the common block TOC in octal and floating.

•

Author(s) : R. Brun, M. Goossens, B. Holl, O. Schaile, J. Shiers, J. Zoll **Library:** PACKLIB
Submitter : **Submitted:** 18.04.1986
Language : Fortran **Revised:**

Dynamic Data Structure and Memory Manager

ZEBRA is a dynamic data structure and memory manager. It allows the management of large amounts of data in a computer store by providing the functions required to construct a logical graph of the data and their interrelations.

The data are stored in Fortran COMMON blocks, called ”stores”. Each store can be subdivided into up to 20 ”divisions”. Relations between the basic units of data, or ”banks”, are expressed by attaching a structural significance to part of a bank. A bank is accessed by specifying its address in a given store. Such addresses (called ”links”) are kept inside the banks or in ”link areas” inside a common block.

- The memory management part of ZEBRA is performed by the MZ package. Utilities are available for reorganizing, sorting and deleting banks and data structures.
- Individual banks, data structures or complete divisions can be output with the FZ package.
- Direct access files for data structures and the management of the data by keywords are provided by the RZ package.
- Dumps and verification of ZEBRA structures and documentation tools are available in the DZ package.

Structure:

SUBROUTINE subprograms
User Entry Names: ZEBRA
External References: KERNLIB (Q100) routines

Usage:

See **Long Write-up.**
•

Author(s) : O. Couet **Library:** GRAFLIB
Submitter : **Submitted:** 10.02.1988
Language : Fortran and C **Revised:** 01.11.1994

High Level Interface to Graphics and Zebra

The HIGZ package is part of PAW (Q121) (Physics Analysis Workstation), but can be used independently. HIGZ contains entries which look and act like many of the entries of GKS (Graphics Kernel System) and, in addition, has entries providing a higher level of functionality such as plotting whole histograms. HIGZ also contains an option to create a device independent metafile stored in ZEBRA (Q100) format which can hence be ported, and re-interpreted, on other machines and operating systems.

The complete HIGZ facilities are available in the PAW (Q121) system.

Structure:

SUBROUTINE subprograms

Usage:

See **Long Write-up**.

•

Author(s) : R. Brun, O. Couet, N. Cremel, A. Nathaniel, A. Rademakers, C. Vandoni **Library:** GRAFLIB
Submitter : R. Brun **Submitted:** 10.02.1988
Language : Interactive **Revised:** 01.11.1994

PAW - Physics Analysis Workstation Package

PAW is a program package to assist physicists in the analysis and presentation of their data. It provides interactive graphical presentation and statistical or mathematical analysis, working on objects familiar to physicists like histograms, event files (n-tuples) and vectors.

The PAW++ program provides a Motif interface to PAW.

Structure:

Interactive data analysis program.

Usage:

See **Long write-up**.

Notes:

The packages involved in the implementation of PAW and the platform availability are described in the **Reference Manual**.

•

Author(s) : C. Vandoni **Library:** PAWLIB
Submitter : **Submitted:** 14.11.1988
Language : Fortran **Revised:**

SIGMA - System for Interactive Graphical Mathematical Applications

SIGMA can be considered a system for interactive on-line numerical analysis problem-solving which has been designed essentially for mathematicians and theoretical physicists. The major characteristics of SIGMA are:

- The basic data units are scalars, one-dimensional arrays, and multi-dimensional rectangular arrays; SIGMA provides automatic handling of these arrays.
- The calculational operators of SIGMA closely resembles the operations of numerical mathematics; procedural operators are often analogous to those of Fortran.
- The system is designed to be used in interactive mode; it provides convenient facilities for graphical display of arrays in form of (sets of) curves.
- The user can construct his own programs within the system and has also access to a program library; he can store and retrieve his data and programs; he obtains on request hard copy of alphanumeric and graphical type.

SIGMA was operational for many years on the CYBER computers at CERN. Most of its functionality has been converted to run on other machines as part of the PAW (Q121) package.

Usage:

See Chapter 6 of the PAW Manual.

•

Author(s) : J. Shiers **Library:** PACKLIB
Submitter : **Submitted:** 01.10.1991
Language : Fortran, C **Revised:**

Distributed File and Tape Management System

The FATMEN package is a set of Fortran callable routines and utilities for the management of disk and tape files. In particular, the package provides location, operating system and medium transparency. A command line interface also exists.

Structure:

SUBROUTINE subprograms and command line shell.

Usage:

See **Long Write-up**.

•

Author(s) : Various **Library:** PACKLIB
Submitter : J. Shiers **Submitted:** 01.10.1991
Language : Fortran, C, Pascal, Assembler **Revised:**

Client Server Routines and Utilities

The `CSPACK` package is a set of Fortran callable routines and utilities. In particular, it provides remote file access and transfer with automatic conversion between data representations for commonly used HEP formats, such as `PAM` files, Zebra `FZ` and `RZ` files. A command line interface also exists (`ZFTP`).

This package also includes `TELNETG`, an enhanced `TELNET` utility with graphics capabilties and the `SYSREQ` facility, used at CERN for interaction with the Tape Management System.

Structure:

`SUBROUTINE` subprograms and command line shell.

Usage:

See **Long Write-up**.

•

Author(s) : L3, OPAL, CN
Submitter : J. Shiers
Language : Fortran, C

Library: PACKLIB
Submitted: 01.06.1992
Revised:

Distributed Database Management System

The HEPDB package is a set of Fortran callable routines and utilities for the management of database objects such as calibration data and detector geometry. One may store and retrieve objects such as Zebra structures, vectors, text files and help information. The package is heavily based upon the DBL3 and OPCAL systems, developed by the L3 and OPAL collaborations respectively. A command line interface also exists.

Structure:

SUBROUTINE subprograms and command line shell.

Usage:

See **Long Write-up**.

•

Author(s) : R. Brun, F. Carena, M. Hansroul, H. Grote, J.C. Lassalle, W. Wojcik
Library: PACKLIB
Submitter :
Submitted: 15.09.1978
Language : Fortran
Revised: 17.12.1991

Dynamic Memory Management

OBSOLETE

Please note that this routine has been obsoleted in CNL 219. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: ZEBRA (Q100)

ZBOOK provides facilities to create (at execution time) memory blocks of variable lengths, manage them and perform the following operations on them:

- create a block
- increase or decrease size of block
- set block to zero
- drop or delete block
- write block to file
- read from file
- print contents of block

Using ZBOOK, the total size of all blocks together cannot exceed the dimension of the array specified in the user's Fortran program. Using a subpackage YBOOK in connection with HBOOK (Y250), however, dynamic allocation of the total space is possible.

Structure:

SUBROUTINE package
User Entry Names: ZBOOK

Usage:

See **Long Write-up**.

•

Author(s) : M. Metcalf **Library:** PGMLIB
Submitter : **Submitted:** 01.04.1983
Language : Fortran **Revised:**

Indent Fortran Source

The program reads Fortran source from a specified input file and writes the indented source code to a specified output file.

Structure:

Complete `PROGRAM`
User Entry Names: `INDENT`
Files Referenced: Input and output units, either default or user defined.

Usage:

`INDENT` reads from the default input unit four integer values in a single record. The default values are taken if this record is absent.

Indenting shift	(Default = `3`)
Maximal indenting level	(Default = `10`)
File number of source input	(Default = `5`)
File number of transformed source output	(Default = `6`)

Note that the first column of the output file will be taken as carriage control information if the output unit is a line printer.

Method:

The program detects the beginning and end of each `DO`– and `IF`–block, and indents each following source line by a shift corresponding to the nesting level. Continuation lines are constructed when necessary, but variable names are never split across two lines.
`PATCHY` control records are treated as comment lines, and so complete `PAM`s can be handled.

Restrictions:

Lines containing `FORMAT` statements, or character strings with multiple embedded blanks are not indented.

Sequences of more than 200 comment lines may have their order with respect to the following statement modified.
Assembler code gets destroyed.

Error handling:

Primitive syntax checks protect the program from most non-Fortran source input.

References:

1. M. Metcalf, FORTRAN Optimization, Academic Press London (1982), Appendix B.

•

Author(s) : H. Grote **Library:** PGMLIB
Submitter : **Submitted:** 29.11.1988
Language : Fortran **Revised:**

FLOP - Fortran Language Oriented Parser

`FLOP` is best described as an "intelligent" editor that recognizes Fortran (`ANSI 77`) code, with a full coverage of `ANSI 66` and some of its extensions). To achieve this, `FLOP` has to perform part of the functions of a compiler, mainly the declaration and syntax analysis. The knowledge resulting from this then allows `FLOP` to edit the Fortran input file in various ways, and to provide useful information about its contents.

Structure:

Complete `PROGRAM`
Files Referenced: `Unit 11` (input), `Unit 5` (commands), `Unit 6` (output)
External References: `TIMEL` (Z007), `TIMEX` (Z007)

Usage:

See **Long Write-up**.

Refer also to the interactive help files or to the `FLOP DECKS` in the various Patches of the `INSTALL` Pam file for examples of usage.

The source code can be found in the `FLOP` Pam file on the various machines.

•

Author(s) : M. Metcalf
Submitter :
Language : Fortran

Library: PGMLIB
Submitted: 01.02.1992
Revised:

Fortran 77 to Fortran 90 source form conversion tool

Users of Fortran 90 can choose between two different styles of source form, the old (Fortran 77) and a new. This program reads code written according to the Fortran 77 *fixed* source form from a specified input file and writes it according to the Fortran 90 *free* source form to a specified output file. It also formats the code by indenting the bodies of `DO`-loops and `IF`-blocks, and performs a small number of syntax conversions.

Structure:

Complete `PROGRAM`
User Entry Names: `CONVERT`
Files Referenced: Input and output units, either default or user defined.

Usage:

`CONVERT` has the following calling sequence on all systems:

```
convert [-b] [-id n] [-il m] [-sb] ifile[.f] [ofile[.f90]]
        [+b]                 [+sb]
```

where the meaning of the arguments is as follows:

`-id` Indenting depth (default = `3`).

`-il` Maximal indenting level (default = `10`).

`-sb` Handle significant blanks (default).

`-b` Generate interface blocks only.

If no options are specified, significant blanks will be handled (`-sb`) and all code will be processed (`+b`). In order to do nothing but change the source form, type e.g.:

```
convert -id 0 -il 0 +sb mysource.f
```

Method:

The program converts between the old fixed Fortran 77 source form to the new Fortran 90 free source form. Note that blanks are significant in the new source form. In addition it is able to perform a few other useful operations on the fly.

Statement keywords are followed if necessary by a blank, and blanks within tokens are suppressed; this handling of blanks is optional, but the default (`-sb`).

If a `CONTINUE` statement terminates a single `DO` loop, it is replaced by `END DO`.

Procedure `END` statements have the procedure name added, if blanks are handled (`-sb`).

Statements like `INTEGER*2` are converted to `INTEGER(2)`, if blanks are handled (`-sb`). Depending on the target processor, a further global edit might be required (e.g. where 2 bytes correspond to `KIND=1`). Typed functions and assumed-length character specifications are treated similarly. The length specification `*4` is removed for all data types except `CHARACTER`, as is `*8` for `COMPLEX`. This treatment of non-standard type declarations includes any non-standard `IMPLICIT` statements.

Optionally, interface blocks only may be produced (`-b`); this requires blank processing to be requested (`-sb`). The interface blocks are written in a form compatible with both the old and the new source forms.

The program is able to handle Patchy Card files, as a `+` in column 1 is treated as a comment line

Restrictions:

The program does not indent `FORMAT` statements or any statement containing a character string with an embedded multiple blank. The order of comment lines and Fortran statements is slightly modified if there are sequences of more than 200 comment lines. If there are syntax errors, continued lines do not have a trailing `&`.

When producing interface blocks, a check is required that any dummy argument that is a procedure has a corresponding `EXTERNAL` statement. Also, since no `COMMON` blocks or `PARAMETER` statements are copied, part of an assumed-size array declaration may be missing. Similarly, parts of an assumed-length character symbolic constant might be copied and have to be deleted. `BLOCK DATA` statements are copied and must be deleted. These problems would normally be detected by a compiler and are trivially corrected.

Within a given keyword, the case must be all upper or all lower, and lower case programs require blank handling for correct indenting.

Error handling:

Primitive syntax checks protect the program from most non-Fortran source input.

References:

1. M. Metcalf and J.Reid, Fortran 90 explained, Oxford Science Publications (1990), Chapter 2

•

Author(s) : J. Zoll **Library:** None
Submitter : **Submitted:** 15.09.1994
Language : Fortran + C **Revised:**

Wylbur Phoenix – a Line Editor for ASCII Text Files

OBSOLETE

Please note that this routine has been obsoleted. Users are advised not to use it any longer. No maintenance for it will take place and it will eventually disappear.

`Wylbur Phoenix` is a portable command driven editor, capable of embedding a full-screen editor of the user's choice as a sub-system. It can operate with the simplest Telnet connection to some remote machine. It is designed to give maximum power for the development and maintenance of the source files of the large programs used in particle physics, where it is neccessary to easily find in a large volume what one is looking for. It has been written because no editor is available which combines all the features considered essential:

a) Ease of use for the casual user;
b) 'undo' a series of mistaken edit operations;
c) global changes displayed, and maybe confirmed individually;
d) column sensitive editing;
e) handling of program variable names, not only text strings, but without language syntax analysis;
f) direct handling of program units, ie. Fortran or C routines or Patchy decks.
g) 'master range' automatically limiting edit operations to an arbitrary fraction of the whole file;
h) usage of windows as monitors and for full-screen editing;
i) immediate, context-free, display of critical lines.
j) permanent line numbers, not hindering normal access to the files by programs other than the editor;
k) portability.

Although `Wylbur Phoenix` does have some aspects of 'full screen' and interactive operations, these are distinct features which can selectively be switched off in 'batch mode' or in 'nowindow mode'. Thus `Wylbur` can be used in shell scripts and across non-specialized computer links; indeed for some applications `Wylbur` in batch mode is very convenient.

Structure:

Complete program

Usage:

Shell command "use fn" calls the normal version of Wylbur into operation to act on file "fn". This version is typically capable of handling 60000 lines. For bigger files one may use "useb" on some machines, which allows for 120000 lines.
On the Unix machines "use" and "useb" are links in `/cern/pro/bin` pointing to the executable modules.
On the Vax "use" should be a symbol like

```
$ USE :== $CERN_ROOT:[EXE]WYLBUR
```

`Wylbur` has not been made to work on IBM with VM/CMS.
To print the file used for delivering on-line help proceed as follows:

type "use"	to call `Wylbur` into operation,
type "help -p temp 84"	to create file "temp" for printing,
type "help h"	for instructions on how to print file "temp".

•

Author(s) : C. Iselin
Submitter :
Language : Fortran

Library: PGMLIB
Submitted: 01.02.1982
Revised: 27.11.1984

Solution of Poisson's or Laplace's Equation in Two-Dimensional Regions

The POISCR program package consists of a set of programs designed for the solution of Poisson's or Laplace's equation in two-dimensional regions. The programs have originally been written to solve magnetostatic problems, but they can equally well be used for other potential problems. Material properties may be linear or non-linear. Polarized material (like permanent magnet material) is allowed.

Structure:

Complete PROGRAM package
User Entry Names: FORCCR, LATTCR, POISCR, TRIPCR
Files Referenced: As defined in the POISSON exec file. Unit 11, Unit 12

Usage:

See **Long Write-up**.

Source:

A program POISSON was originally written by R.F. Holsinger then working at LBL. It was based on an earlier program TRIM by A. Winslow and on theoretical work by K. Halbach (LBL). The CERN Program Library version POISCR is a revision of these programs by C. Iselin (CERN).

•

Author(s) : TC **Library:** KERNLIB
Submitter : J. Zoll **Submitted:** 01.03.1968
Language : Fortran **Revised:** 27.11.1984

Lorentz Transformation

This routine transforms momentum and energy of a particle from one Lorentz-frame to another.
Seen from the reference system Σ, the other system Σ' has the velocity $\vec{\beta}$, with $\vec{\eta} = \gamma\vec{\beta}$.
If a rest mass M is tied to system Σ', with energy E and momentum $\vec{P}$, we have:

$$\vec{\beta} = \vec{P}/E, \qquad \vec{\eta} = \vec{P}/M, \qquad \gamma = E/M.$$

The momentum and energy of a particle with mass m is

in system Σ : $\vec{p}$ and $e = \sqrt{p^2 + m^2}$,
in system Σ' : $\vec{p'}$ and $e' = \sqrt{p'^2 + m^2}$.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `LOREN4`

Usage:

```
CALL LOREN4(S,A,X)
```

with the 4–vectors `S` $= (\vec{P}, E)$ and `A` $= (\vec{p}, e)$ calculates the transformed 4–vector `X` $= (\vec{p'}, e')$.
`LOREN4` contains one square-root to derive M from P and E.

Method:

If we split $\vec{p} = \vec{p}_L + \vec{p}_T$ into components parallel and normal to $\vec{\beta}$, where

$$\vec{p}_L = \frac{\vec{p}\,\vec{\eta}}{\eta^2}\,\vec{\eta}, \qquad \vec{p}_T = \vec{p} - \vec{p}_L,$$

we can write the transformations as

$$\vec{p'}_L = \gamma\,\vec{p}_L - \vec{\eta}\,e, \qquad \vec{p'}_T = \vec{p}_T, \qquad e' = \gamma\,e - \vec{\eta}\,\vec{p}$$

and get

$$\begin{aligned}
\vec{p'} &= \vec{p} + (\gamma - 1)\vec{p}_L - e\,\vec{\eta} \\
&= \vec{p} + \vec{\eta}\,((\gamma - 1)\vec{p}\,\vec{\eta}/\eta^2 - e) \\
&= \vec{p} + \vec{\eta}\,(\vec{p}\,\vec{\eta}/(\gamma + 1) - e) \qquad \text{(because of} \qquad \eta^2 = \gamma^2 - 1) \\
&= \vec{p} + \vec{P}\,(\vec{p}\,\vec{P}/(E + M) - e)/M, \\
\vec{e'} &= \gamma e - \vec{\eta}\,\vec{p} \\
&= (eE - \vec{p}\,\vec{P})/M.
\end{aligned}$$

•

Author(s) : V. Framery, L. Pape
Library: KERNLIB
Submitter :
Submitted: 01.03.1968
Language : Fortran
Revised: 16.09.1991

Lorentz Transformations

LORENF transforms the momentum 4-vector of a particle from the Lorentz-frame Σ to the frame Σ' like LOREN4 (U101); it is faster than LOREN4 because the rest-mass M of Σ' is passed as an argument to save the square root.

LORENB executes the inverse transformation.

Structure:

SUBROUTINE subprograms
User Entry Names: LORENF, LORENB

Usage:

`CALL LORENF(SM,SP,PB,PF)`	forward transformation	`PB -> PF`
`CALL LORENB(SM,SP,PF,PB)`	backward transformation	`PF -> PB`

with

SM	Rest-mass M of system Σ' with $M^2 = E^2 - P^2$.
SP	Momentum 4-vector (P, E) of Σ' in Σ.
PB	Momentum 4-vector (p, e) in Σ.
PF	Momentum 4-vector (p', e') in Σ'.

Method:

For LORENF (cf. LOREN4 (U101)):

$$\begin{aligned} e' &= (eE - pP)/M \\ p' &= p - P(e + e')/(E + M) \end{aligned}$$

because $pP = eE - e'M$ and $pP - e(E + M) = -M(e + e')$.
For LORENB :

$$\begin{aligned} e &= (e'E + p'P)/M \\ p &= p' + P(e + e')/(E + M) \end{aligned}$$

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.10.1994
Language : Fortran **Revised:**

Wigner 3-j, 6-j, 9-j Symbols; Clebsch-Gordan, Racah W-, Jahn U-Coefficients

Function subprograms `RWIG3J`, `DWIG3J`; `RWIG6J`, `DWIG6J`; `RWIG9J`, `DWIG9J`; `RCLEBG`, `DCLEBG`; `RRACAW`, `DRACAW` and `RJAHNU`, `DJAHNU` calculate the Wigner 3-j, 6-j and 9-j symbols, the Clebsch-Gordan coefficients, the Racah W-coefficients and the Jahn U-coefficients, respectively.

On CDC and Cray computers, the double-precision versions `DWIG3J` etc. are not available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RWIG3J`, `RWIG6J`, `RWIG9J`, `RCLEBG`, `RRACAW`, `RJAHNU`
`DWIG3J`, `DWIG6J`, `DWIG9J`, `DCLEBG`, `DRACAW`, `DJAHNU`

Usage:

In any arithmetic expression, for `t` = `R` (type `REAL`), or `t` = `D` (type `DOUBLE PRECISION`),

`tWIG3J(A,B,C,X,Y,Z)`	has the value of	$\begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix}$;
`tWIG6J(A,B,C,X,Y,Z)`	has the value of	$\begin{Bmatrix} a & b & c \\ x & y & z \end{Bmatrix}$;
`tWIG9J(A,B,C,P,Q,R,X,Y,Z)`	has the value of	$\begin{Bmatrix} a & b & c \\ p & q & r \\ x & y & z \end{Bmatrix}$;
`tCLEBG(A,B,C,X,Y,Z)`	has the value of	$(a\,b\,x\,y\,|\,a\,b\,c\,z)$;
`tRACAW(A,B,C,D,E,F)`	has the value of	$W(a\,b\,c\,d\,;\,e\,f)$;
`tJAHNU(A,B,C,D,E,F)`	has the value of	$U(a\,b\,c\,d\,;\,e\,f)$.

All the arguments must have integral or half-integral values (see **Notes**). They have the same type as the function name. For definitions and notations see **References**.
The following relations hold (see Refs. 1 and 3):
Clebsch-Gordan coefficient (in terms of the Wigner 3-j symbol):

$$(a\,b\,x\,y\,|\,a\,b\,c\,z) \;=\; (-1)^{a-b-z}\sqrt{2c+1}\begin{pmatrix} a & b & c \\ x & y & -z \end{pmatrix};$$

Racah W-coefficient (in terms of the Wigner 6-j symbol):

$$W(a\,b\,c\,d\,;\,e\,f) \;=\; (-1)^{a+b+c+d}\begin{Bmatrix} a & b & e \\ d & c & f \end{Bmatrix}.$$

Jahn U-coefficient (in terms of the Wigner 6-j symbol and the Racah W-coefficient):

$$\begin{aligned} U(a\,b\,c\,d\,;\,e\,f) &= (-1)^{a+b+c+d}\sqrt{(2e+1)(2f+1)}\begin{Bmatrix} a & b & e \\ d & c & f \end{Bmatrix} \\ &= \sqrt{(2e+1)(2f+1)}\,W(a\,b\,c\,d\,;\,e\,f). \end{aligned}$$

Method:

The Wigner 3-j symbol and the Clebsch-Gordan coefficient are calculated from formulas (5.1) and (5.10) of Ref. 1, respectively. The Wigner 6-j symbol, the Racah W- and the Jahn U-coefficient are calculated from formulas (5.23) and (5.24) of Ref. 1. In both cases, the factorials are replaced by their logarithms during the calculation. The Wigner 9-j symbol is calculated from formula (5.37) of Ref. 1 in terms of Wigner 6-j symbols.

Notes:

A Wigner-3j symbol $\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix}$ is considered to be zero unless simultaneously

(i) j_i and m_i have both either integral or half-integral values (each i),

(ii) $j_i \geq |m_i| \geq 0$ (each i),

(iii) $m_1 + m_2 + m_3 = 0$,

(iv) $j_1 - j_2 - m_3$ is an integer,

(v) $j_1 + j_2 + j_3$ is an integer and $j_1 + j_2 \geq j_3, \quad j_2 + j_3 \geq j_1, \quad j_3 + j_1 \geq j_2$.

The conditions (v) are often denoted by $\delta(j_1\, j_2\, j_3)$ and are called the *triangle relations*.

For a Clebsch-Gordan coefficient $(j_1\, j_2\, m_1\, m_2 \,|\, j_1\, j_2\, j_3\, m_3)$, condition (iii) reads $m_1 + m_2 = m_3$ and condition (iv) disappears.

A Wigner-6j symbol $\begin{Bmatrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \end{Bmatrix}$ is considered to be zero unless simultaneously

(i) all j_i and l_i have non-negative integral or half-integral values,

(ii) the four *triangle relations* $\delta(j_1\, j_2\, j_3), \quad \delta(j_1\, l_2\, l_3), \quad \delta(l_1\, j_2\, l_3), \quad \delta(l_1\, l_2\, j_3)$ hold.

A Wigner-9j symbol $\begin{Bmatrix} j_{11} & j_{12} & j_{13} \\ j_{21} & j_{22} & j_{23} \\ j_{31} & j_{32} & j_{33} \end{Bmatrix}$ is considered to be zero unless simultaneously

(i) all j_{ik} have non-negative integral or half-integral values,

(ii) the arguments in each row and in each column satisfy the *triangle relations*.

Restrictions:

The sum of arguments in any *triangle relation* must not exceed 100. No test is made.

References:

1. R.D. Cowan, The theory of atomic structure and spectra, (Univ. of California Press, Berkeley CA 1981).

2. A.F. Nikiforov, V.B. Uvarov and Yu.L. Levitan, Tables of Racah coefficients (Pergamon Press, Oxford 1965).

3. M. Rotenberg, R. Bivins, N. Metropolis and J.K. Wooten, Jr., The 3-j and 6-j symbols (Crosby Lockwood, London 1959).

4. D.A. Varshalovich, A.N. Moskalev and V.K. Khersonskii, Quantum theory of angular momentum (World Scientific, Singapore 1988).

•

Author(s) : K.S. Kölbig | **Library:** MATHLIB
Submitter : | **Submitted:** 15.10.1994
Language : Fortran | **Revised:**

Clebsch-Gordan Coefficients in Rational Form

Function subprogram RTCLGN calculates the (signed) square of the Clebsch-Gordan coefficient in rational form and in powers of prime numbers. In terms of the Wigner-3j symbol, this coefficient is defined by

$$C \quad = \quad (j_1\, j_2\, m_1\, m_2 \,|\, j_1\, j_2\, j_3\, m_3) \;=\; (-1)^{j_1-j_2+m_3}\,\sqrt{2j_3+1}\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & -m_3 \end{pmatrix}.$$

All j_i and m_i must have integral or half-integral values (see **Notes**). For definitions and notations see Ref. 1.

On computers other than CDC and Cray, only the double-precision version DTCLGN is available. On CDC and Cray computers, only the single-precision version RTCLGN is available.

Structure:

SUBROUTINE subprogram
User Entry Names: RTCLGN
Files Referenced: Unit 6

Usage:

For t = R (type REAL), t = D (type DOUBLE PRECISION),

```
CALL tTCLGN(JJ1,JJ2,JJ3,MM1,MM2,MM3,RNUM,RDEN,KPEX)
```

JJ1,JJ2,JJ3 (INTEGER) The j-parameters multiplied by two, i.e. JJ1$= 2j_1$ etc.

MM1,MM2,MM3 (INTEGER) The m-parameters multiplied by two, i.e. MM1$= 2m_1$ etc.

RNUM (type according to t) Contains, on exit, the signed numerator of C^2.

RDEN (type according to t) Contains, on exit, the denominator of C^2.

KPEX (INTEGER) Array of length 40 at least. Contains, on exit, the exponents k_n in the expression

$$C^2 \quad = \quad \prod_{n=1}^{40} p_n^{k_n},$$

where $p_1 = 2,\ p_2 = 3,\ p_3 = 5, \ldots,\ p_{40} = 173$ are the first 40 prime numbers.

Notes:

A Clebsch-Gordan coefficient $(j_1\, j_2\, m_1\, m_2 \,|\, j_1\, j_2\, j_3\, m_3)$ is considered to be zero unless simultaneously

(i) j_i and m_i have both either integral or half-integral values (each i),

(ii) $j_i \ge |m_i| \ge 0$ (each i),

(iii) $m_1 + m_2 = m_3$,

(iv) $j_1 + j_2 + j_3$ is an integer and $j_1 + j_2 \ge j_3,\quad j_2 + j_3 \ge j_1,\quad j_3 + j_1 \ge j_2$.

In this case, RNUM $= 0$, RDEN $= 1$ or DNUM $= 0$, DDEN $= 1$, respectively, and KPEX(n) $= 0$, (n $= 1, \ldots, 40$).

Source:

This subroutine is based on an earlier version by H. Yoshiki.

Error handling:

Error `U112.1`: The calculation requires a prime number p_n with $n > 40$.
In this case, $\texttt{DNUM} = \texttt{0}$, $\texttt{DDEN} = \texttt{1}$, $\texttt{KPEX(n)} = \texttt{0}$, $(\texttt{n} = \texttt{1}, \dots, \texttt{40})$. A message is written on `Unit 6` unless subroutine `MTLSET` (N002) has been called.

References:

1. R.D. Cowan, The theory of atomic structure and spectra, (Univ. of California Press, Berkeley CA 1981) 142–144

•

Author(s) : K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.02.1989
Language : Fortran **Revised:** 01.12.1994

Beta-Term in Wigner's D-Function

Function subprograms `RDJMNB` and `DDJMNB` calculate the β-term $d^j_{mn}(\beta)$ in the matrix element of the finite rotation operator (Wigner's D-function)

$$D^j_{mn}(\alpha,\beta,\gamma) \;=\; e^{-im\alpha}\, d^j_{mn}(\beta)\, e^{in\gamma}$$

by using the formula (Ref. 1, No. 4.3.1(3))

$$d^j_{mn}(\beta) \;=\; (-1)^{j+m}\sqrt{(j+m)!(j-m)!(j+n)!(j-n)!}\;\times$$

$$\sum_k (-1)^k \,\frac{\cos^{2k-m-n}(\frac{1}{2}\beta)\;\sin^{2j+m+n-2k}(\frac{1}{2}\beta)}{k!(j+m-k)!(j+n-k)!(k-m-n)!}$$

for arbitrary (either all integer or all half-integer) values of j, m, n such that $j \geq 0, |m| \leq j$ and $|n| \leq j$. The summation over k runs from $\max(0, m+n)$ to $\min(j+m, j+n)$.
On computers other than CDC or Cray, only the double-precision version `DDJMNB` is available. On CDC and Cray computers, only the single-precision version `RDJMNB` is available.

Structure:

`FUNCTION` subprograms
User Entry Names: `RDJMNB`, `DDJMNB`
Obsolete User Entry Names: `DJMNB` $\equiv$ `RDJMNB`
Files Referenced: `Unit 6`
External References: `MTLMTR` (N002), `ABEND` (Z035)

Usage:

In any arithmetic expression,

`RDJMNB(AJ,AM,AN,BETA)` or `DDJMNB(AJ,AM,AN,BETA)` has the value $d^j_{mn}(\beta)$,

where `AJ` $= j$, `AM` $= m$, `AN` $= n$ and `BETA` $= \beta$. `RDJMNB` is of type `REAL`, `DDJMNB` is of type `DOUBLE PRECISION`, and `AJ`, `AM`, `AN`, `BETA` have the same type as the function name. `BETA` has to be given in degrees.

Restrictions:

`0` $\leq$ `AJ` $\leq$ `25`, $|$`AM`$|$ $\leq$ `AJ`, $|$`AN`$|$ $\leq$ `AJ`, `0` $\leq$ `BETA` $\leq$ `360`.

Accuracy:

Approximately full single- or double-precision machine accuracy, at least for small values of the indices.

Error handling:

Error `U501.1`: If any of the restrictions is not satisfied, the function value is set equal to zero, and a message is written on `Unit 6`, unless subroutine `MTLSET` (N002) has been called.

References:

1. D.A. Varshalovich, A.N. Moskalev and V.K. Khersonskii, Quantum theory of angular momentum, (World Scientific, Singapore 1988) 76

•

Author(s) : CDC: H. von Eicken, IBM: T. Lindelöf
Library: KERNLIB
Submitter :
Submitted: 07.12.1970
Language : Assembler
Revised: 15.09.1978

Uniform Random Numbers

OBSOLETE

Please note that this routine has been obsoleted in CNL 215. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement:
RANMAR (V113) or RANECU (V114) or RANLUX (V115)

RNDM generates uniformly distributed pseudo-random numbers in the interval (0,1) in type REAL and in the interval $(1, 2^{47} - 1)$ (CDC) or $(1, 2^{31} - 1)$ (IBM) in type INTEGER. The CDC version has a period of more than 10^{13}. The IBM period, however, is only about 5×10^8 which may not be good enough for some calculations. In that case RNDM2 (V107) should be used instead.

Structure:

SUBROUTINE subprogram
User Entry Names: IRNDM, RNDM, RDMIN, RDMOUT

Usage:

```
Y = RNDM(X)
```

where X is a dummy argument (see **Notes**), sets Y to a pseudo-random number in the interval (0,1). X and Y are of type REAL.

```
I = IRNDM(X)
```

where X is a dummy argument (see **Notes**), sets I to an integer pseudo-random number in the interval $(1, 2^{47} - 1)$ on CDC, $(1, 2^{31} - 1)$ on IBM. X is of type REAL and I is of type INTEGER.

```
CALL RDMOUT(SEED)
```

replaces SEED by the current value of the integer pseudo-random number. This SEED may then be used to restart the sequence at this point, by a call to RDMIN. SEED is of type REAL.

```
CALL RDMIN(SEED)
```

replaces the current value of the integer pseudo-random number by the value of the variable SEED. SEED is of type REAL. The value of SEED should not be chosen by the user but should be obtained by a previous call to RDMOUT. If this is not complied with, the numbers generated may have serious defects in their randomness.

Method:

CDC:
Consider the sequence:

$$\begin{aligned} r_i &= \alpha r_{i-1} (\bmod 2^{47}) \text{ for } i = 1, 2, \ldots \\ \text{with } r_0 &= 2000\,0000\,0110\,6047\,1625_8 \\ \text{and } \alpha &= 2000\,0000\,3432\,7724\,4615_8 \end{aligned}$$

where r_0 and α are the unnormalised floating-point representation of the starting number and 5^{15} respectively. The j-th floating-point number R_j is obtained by packing r_j with an exponent (-47) and normalising it. This ensures that R_j falls in the interval (0,1).

The product αr_{j-1} is generated in a 96 bit accumulator. The integer number N_j returned is the low order 47 bits of the contents of this accumulator, except that the right-most 11 bits are replaced by those occupying bit positions 48-58. This replacement is done in order to increase the time period of the low order bits.

IBM: See write-up for `RNDM2` (V107).

Notes:

While the argument is dummy, in the sense that the generator makes no use of it, it must be noted that if a reference to `RNDM` occurs

- more than once within a Fortran statement, the argument to it should be different in each case;
- in a `DO`-loop, the argument must depend either directly or indirectly on the index of this loop.

These rules must be observed since the compilers, in their attempt to optimise the object code, assume that functions called with identical arguments return the same function value.

•

NRAN	CERN Program Library	V105

Author(s) : T. Lindelöf, F. James
Submitter :
Language : CDC: Compass, IBM: Fortran

Library: MATHLIB
Submitted: 15.06.1976
Revised:

Arrays of Uniform Random Numbers

OBSOLETE
Please note that this routine has been obsoleted in CNL 215. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.
Suggested replacement: `RANMAR` (V113) or `RANECU` (V114) or `RANLUX` (V115)

`NRAN` on CDC is about 4 times faster than `RNDM` when 'many' uniformly distributed random numbers are to be generated at once.

`NRAN` on IBM is not recommended. It is merely a Fortran interface to `RNDM`. Thus this description applies only to the CDC version.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `NRAN`, `NRANIN`, `NRANUT`

Usage:

```
CALL NRAN(VEC,N)
```

fills the array `VEC` (of length `N` at least) with `N` independent pseudo random numbers uniformly distributed in the interval (0,1), the end-points excluded. The other two entries may be used to retrieve and set the 'seed' as follows:

```
CALL NRANUT(SEED)
```

returns in `SEED` the current value of a quantitity which is changed after each call to `NRAN` and upon which the future random number sequence depends. Its initial default value is
17170000000000000001_8.

```
CALL NRANIN(SEED)
```

presets the above-mentioned quantity to `SEED`. `SEED` may be any number of the form
$\mathtt{1717xxxxxxxxxxxxxxxy}_8$ where `y` must be `1` or `5` and the `x`'s any octal digits.

Method:

Multiplicative congruential method with the multiplier 20001170673633457725_8. The sequence generated is independent of that of `RNDM` (V104) so that both may be used together.

References:

1. Computing **6**, (1970) 121.

•

Author(s) : G. Marsaglia, A. Zaman **Library:** MATHLIB
Submitter : F. Carminati, F. James **Submitted:** 08.06.1989
Language : Fortran **Revised:**

Fast Uniform Random Number Generator

RANMAR generates a sequence of 32-bit floating-point random numbers uniformly distributed in the interval (0,1), the end points excluded. These numbers are returned in a vector. The period is about 10^{43} and the quality is good but it fails some tests. For better quality use RANLUX (V115), which is slower.

Several independent sequences can be initialized and used in the same run.

Structure:

SUBROUTINE subprograms
User Entry Names: RMMAR, RMMAQ, RANMAR, RMARIN, RMARUT
COMMON Block Names and Length: /RANMA1/ 104, /RANMA2/ 104

Usage:

For a single sequence:

```
CALL RANMAR(VEC,LEN)
```

VEC (REAL) Array of length LEN at least. On exit, it will contain the in (0,1) uniformly distributed random numbers.

LEN (INTEGER) Number of random numbers to be generated. Unchanged on exit.

The initialization is made by

```
CALL RMARIN(IJKLIN,NTOTIN,NTO2IN)
```

IJKLIN (INTEGER) Seed from which to start the sequence. Every integer number from 1 to 900 000 000 originates an independent sequence of random numbers with operand of 2^{144} (about 10^{43}).

NTOTIN (INTEGER) Number (mod 10^9) of random number generated.

NTO2IN (INTEGER) Billions (10^9) of random numbers generated.

The arguments NTOTIN and NTO2IN are used to restart the generation from a given point by skipping over already performed extractions. They are returned by RMARUT and should not be touched by the user.

```
CALL RMARUT(IJKLUT,NTOTUT,NTO2UT)
```

IJKLUT (INTEGER) Seed from which the sequence was started.

NTOTUT (INTEGER) Number (mod 10^9) of random number generated so far.

NTO2UT (INTEGER) Billions (10^9) of random numbers generated so far.

For multiple sequences:

```
CALL RMMAR(VEC,LEN,ISEQ)
```

`VEC` (`REAL`) Array of length `LEN` at least. On exit, it will contain the in (0,1) uniformly distributed random numbers.

`LEN` (`INTEGER`) Number of random numbers to be generated. Unchanged on exit.

`ISEQ` (`INTEGER`) Number of the independent sequence from which the `LEN` numbers should be extracted. If $\leq$ `0`, the last valid sequence explicitely defined is used. Unchanged on exit.

Several independent sequences can be defined and used. Each sequence **must** be initialized by the user, otherwise the result is unpredictable. By default the routine contains a buffer of space to handle only one sequence. If more sequences are needed, then a bigger buffer should be allocated in the main program defining the `COMMON` block `/RANMA2/` to the appropriate size. The space needed is 1 word + 103 words for every random sequence initialized.
The sequences are initialized by

```
CALL RMMAQ(ISEED,ISEQ,CHOPT)
```

`ISEED` (`INTEGER`) Array of length `3` or `103` according to the option specified in `CHOPT`. The first location contains the integer seed from which to start the sequence. Every integer number from `1` to `900 000 000` originates an independent sequence of random numbers, with a period of 2^{144} (about 10^{43}). The second and the third location contain numbers used internally to re-initialize the generator by skipping and should not be touched by the user. The other numbers are a snapshot of the complete status of the generator. If saved, they can be used to restart the generator without skipping over numbers already generated.

`ISEQ` (`INTEGER`) This variable contains, on entry, the number of the independent random number sequence which should be addressed by the present call. If $\leq$ `0`, then the last valid sequence used will be addressed either for a save or a store. If option `'R'` is specified, on exit the variable will contain the sequence actually used.

`CHOPT` (`CHARACTER`) Specifies the action which `RMMAQ` should take. Possible options are:

- `' '` (Blank) The sequence number `1` will be initialized with a default seed. All arguments are ignored.
- `'R'` Get the present status of the generator. If option `'V'` is also present, then the complete status of the generator will be dumped in the array `ISEED`. This options will use 103 words in `ISEED` but has the advantage that the generator can be restarted immediately without skipping numbers. If option `'V'` is not present, then only 3 words will be used but the generator will have to be restarted by skipping the number of events generated so far.
- `'S'` Set the status of the generator to a previously saved state. If option `'V'` is also present, then an array `ISEED` of 103 words is expected, which comes from a previous call to the routine with option `'RV'`. This kind of initialization is very fast. If the option `'V'` is not specified then the generator will be restarted regenerating the same number of random extractions it generated at the time the status was saved. In this case only the first 3 locations of `ISEED` will be used.
- `'V'` Vector option. 103 words will be saved/restored. This allows to restart the generator without skipping over numbers already generated.

For `RMMAR` one seed is needed to initialize the random number, but it is a one-way initialization. The seed cannot be output and used to restart the sequence. In order to restart the generation, the number of random numbers generated is recorded by the generator. The sequence is restarted either generating this many random numbers or saving and restoring a vector of 103 words. The number of generations is stored in the two array elements `ISEED(2),ISEED(3)` as the period is bigger than the maximum number which can be represented by a 32-bit integer.

Timing:

Time in μsec for extractions and skips:

Extractions per call	1	4	16	128	$1000/10^5$ skips
APOLLO 10000	7.4	6.0	5.6	5.5	15/4.6
APOLLO 4000	69	55	51	50	120/73
IBM390E	4.3	2.5	2.0	1.9	7.4/1.2
CRAY X-MP/48	4.1	2.1	1.7	1.5	6.9/1.6
VAX8650	14	7.3	5.9	5.8	4.7/4.6

References:

1. G. Marsaglia and A. Zaman, Toward a Universal Random Number Generator, Florida State University FSU-SCRI-87-50 (1987).
2. F. James, A Review of Pseudorandom Number Generators, Computer Phys. Comm. **60** (1990) 329–344.

•

RANECU	CERN Program Library	V114

Author(s) : P. l'Ecuyer
Submitter : F. Carminati
Language : Fortran

Library: MATHLIB
Submitted: 27.02.1989
Revised:

Uniform Random Number Generator

RANECU generates a sequence of uniformly distributed random numbers in the interval (0,1). The numbers are returned in a vector. Several independent sequences can be initialized and used in the same run.

Structure:

SUBROUTINE Subprograms
User Entry Names: RANECU, RANECQ
COMMON Block Names and Lengths: /RANEC1/ 402

Usage:

```
CALL RANECU(VEC,LEN,ISEQ)
```

VEC (REAL) Array of length LEN at least. On exit, it will contain the in (0,1) uniformly distributed random numbers.

LEN (INTEGER) Number of random numbers wanted. Unchanged on exit.

ISEQ (INTEGER) Number of the independent sequence from which the LEN numbers should be extracted. If ISEQ $\leq$ 0 then the extraction will be made from the sequence used last. Unchanged on exit.

Several independent sequences can be defined and used. Each sequence MUST be initialized by the user, otherwise the result is unpredictable. By default the routine contains a space buffer to handle only one sequence. If more sequences are needed, then a bigger buffer should be allocated in the calling program defining the COMMON block /RANEC1/ appropriately. *Two* words have to be allocated plus *four* words for every sequence initialized.

Two integer seeds are used to initialize a sequence. Not all pairs of integers define a good random sequence or one which is independent from others. Sections of the same random sequence can be defined as independent sequences. The period of the generator is $2^{60} \approx 10^{18}$. A generation has been performed in order to provide the seeds to start any of the generated sections. There are 100 possible seed pairs and they are all 10^9 numbers apart. Thus a sequence started from one of the seed pairs, after 10^9 numbers will start generating the next one. Each of these sequences is of the same order of magnitude as the basic sequence offered by RNDM (V104). Longer sequences will be generated and the corresponding seeds made available to users. Note that, while the numbers generated by the default sequence will always be the same, the introduction of more sequences may modify some of them. In order to handle the initialization of the package, the following routine is provided:

```
CALL RANECQ(ISEED1,ISEED2,ISEQ,CHOPT)
```

ISEED1 (INTEGER) On entry, it contains the first integer seed from which to start the sequence. Unchanged on exit.

ISEED2 (INTEGER) On entry, it contains the second seed from which to start the sequence. Unchanged on exit.

ISEQ (INTEGER) On entry, it contains the number of the independent sequence of random numbers to be addressed by this call. If ISEQ $\leq$ 0, then the last valid sequence used will be addressed either for a save or a store. In case the option 'R' is specified, on output the variable will contain the sequence actually used.

CHOPT (CHARACTER*1) A character specifying the action which RANECQ should take. Possible options are:

- ' ' If 1 $\leq$ ISEQ $\leq$ 100, the sequence number ISEQ will be initialized with the default seeds of the pre-computed independent sequence number ISEQ. ISEED1 and ISEED2 are ignored.

  If ISEQ $\leq$ 0 or ISEQ $>$ 100, then sequence number 1 will be initialized with the default seeds. ISEED1 and ISEED2 are ignored.
- 'R' Get the present status of the generator. The two integer seeds ISEED1 and ISEED2 will be returned for sequence ISEQ.
- 'S' Set the status of the generator to a previously saved state. The two integer seeds ISEED1 and ISEED2 will be used to restart the generator for sequence ISEQ.
- 'Q' Get the pre-generated seeds for ISEQ (1 $\leq$ ISEQ $\leq$ 100). There are 100 pre-generated sequences each one will generate 10^9 numbers before reproducing the following one.

Timing:

Time in μsec for extractions:

Extractions per call	1	4	16	128
Apollo 10000	6.2	4.4	3.9	3.8
Apollo 4000	52	37	34	33
IBM 3090E	4.9	2.9	2.5	2.4
IBM 3090EVF	3.4	2.3	2.0	1.8
Cray X-MP/48	4.2	2.2	1.7	1.5
VAX 8650	19	13	12	11.6

References:

1. P. l'Ecuyer, Efficient and Portable Random Number Generators, Comm. ACM **31** (1988) 742.
2. F. James, A Review of Pseudorandom Number Generators, Computer Phys. Comm. **60** (1990) 329–344.

•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 15.03.1994
Language : Fortran **Revised:**

Uniform Random Numbers of Guaranteed Quality

`RANLUX` generates pseudorandom numbers uniformly distributed in the interval (0,1), the end points excluded. Each call produces an array of single-precision real numbers of which 24 bits of mantissa are random. The user can choose a **luxury level** which guarantees the quality required for his application. The lowest luxury level (zero) gives a fast generator which will fail some sophisticated tests of randomness; The highest level (four) is about five times slower but guarantees complete randomness. In all cases the period is greater than 10^{165}. Independent subsequences can be generated. Entries are provided for initialization and checkpointing.

Structure:

`SUBROUTINE` Subprograms
User Entry Names: `RANLUX`, `RLUXGO`, `RLUXAT`, `RLUXIN`, `RLUXUT`

Usage:

```
CALL RANLUX(RVEC,LEN)
```

returns a vector `RVEC` of `LEN` 32-bit random floating point numbers in the interval (0,1), the end points excluded. `RVEC` is an array of type `REAL` and of length `LEN` at least.

Luxury levels:

For simplicity, five standard luxury levels may be chosen (t is the time factor relative to level zero; for the definition of p, see **References**). Ref. 1. explains the method, Ref. 2. describes the Fortran implementation in more detail.

Level	p	t	
0	24	1	Equivalent to the original `RCARRY` of Marsaglia and Zaman, very long period, but fails many tests.
1	48	1.5	Considerable improvement in quality over level 0, now passes the gap test, but still fails spectral test.
2	97	2	Passes all known tests, but theoretically still defective.
3	223	3	`DEFAULT VALUE`. Any theoretically possible correlations have very small chance of being observed.
4	389	5	Highest possible luxury, all 24 bits chaotic.

As a rough indication of timing, `RNDM` (V104) is about t=0.5, `RANMAR` (V113) t=1, and `RANECU` (V114) t=2. Concerning the quality scale, `RNDM` is maybe good enough for moving fish around on a screen saver (if you are not afraid of getting some diagonal lines on your screen), `RANMAR` and `RANECU` both have quality which probably corresponds to a luxury level between 1 and 2, but this is based only on empirical testing and true quality may be lower.

No initialization is necessary if the user wants default values. Otherwise the following are available:

```
CALL RLUXGO(LUX,INT,K1,K2)
```

When K1 $=$ K2 $=$ 0, this call initializes the RANLUX generator from one 32-bit integer INT and sets the Luxury Level. If LUX is an integer between 0 and 4, it sets the luxury level as defined above. If LUX $>$ 24, it is taken as the value of p, which then can take on other values than those given in the table. If INT $=$ 0, default initialization is used and only the luxury level is set by LUX. Otherwise, every possible value of INT gives rise to a valid, independent sequence which will not overlap any sequence initialized with any other value of INT. The integers K1 and K2 are used for restarting the generator from a break point saved by RLUXAT.

```
CALL RLUXAT(LUX,INT,K1,K2)
```

dumps the four integers which can be used to restart the generator at this point by calling RLUXGO. RANLUX will then skip over K1 $+ 10^9*$K2 numbers to reach the break point. A more efficient but less convenient method for restarting is offered by RLUXIN and RLUXUT.

```
CALL RLUXIN(IVEC)
```

restarts the generator from vector IVEC of 25 32-bit integers (see RLUXUT). IVEC is an array of type INTEGER and of length 25 at least.

```
CALL RLUXUT(IVEC)
```

outputs the current values of the 25 32-bit integer seeds, to be used for restarting.

References:

1. M. Lüscher, A portable high-quality random number generator for lattice field theory simulations, Computer Phys. Commun. **79** (1994), 100–110.
2. F. James, RANLUX: A Fortran implementation of the high-quality pseudorandom number generator of Lüscher, Computer Phys. Commun. **79** (1994) 111–114.

•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 15.03.1994
Language : Fortran **Revised:**

Double Precision Uniform Random Numbers

RM48 generates pseudorandom numbers using a double-precision (64-bit) adaptation of RANMAR (V113). The floating-point numbers in the interval (0,1), the end points excluded, have 48 significant bits of mantissa (additional bits of mantissa, if supported by the hardware, are zero). Both the code and the results are portable, provided the floating-point mode is adapted to the computer being used (for example, single-precision mode on 64-bit machines, double-precision mode on 32-bit machines).

Structure:

SUBROUTINE Subprograms
User Entry Names: RM48, RM48IN, RM48UT

Usage:

```
CALL RM48(RVEC,LEN)
```

returns a vector RVEC of LEN 64-bit random floating-point numbers in (0,1), the end points excluded. RVEC is an array of length LEN at least. It is of type DOUBLE PRECISION on 32-bit machines, and of type REAL otherwise.

```
CALL RM48IN(I1,N1,N2)
```

initializes the generator from one 32-bit integer I1, and number counts N1, N2 (for initializing, set $\mathtt{N1} = \mathtt{N2} = 0$, but to restart a previously generated sequence, use values output by RM48UT).

```
CALL RM48UT(I1,N1,N2)
```

outputs the value of the original seed and the two number counts, to be used for restarting by initializing to I1 and skipping $100000000 * \mathtt{N2} + \mathtt{N1}$ numbers.

Method:

The method is that of RANMAR (V113).
•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 15.03.1994
Language : Fortran **Revised:**

Gaussian-distributed Random Numbers

RNORML and RNORMX generate (vectors of) single-precision random numbers in a Gaussian distribution of mean zero and variance one. RNORML uses the uniform generator RANMAR underneath, and RNORMX allows the user to choose the uniform generator to be used underneath. The code is portable Fortran, but the results are not guaranteed to be identical on all platforms because there is branch on a floating-point compare which may (very rarely) cause the sequence produced on a given platform to be out of step with that of a different platform.

Structure:

SUBROUTINE Subprograms
User Entry Names: RNORML, RNORMX

Usage:

```
CALL RNORML(RVEC,LEN)
```

generates a vector RVEC of LEN Gaussian-distributed random numbers. RVEC is an array of type REAL and of length LEN at least.

The uniform generator used is RANMAR, so it may be initialized by calling RMARIN (V113), but beware that this also initializes RANMAR (V113)!

An alternative subroutine is supplied which allows the user to select the underlying uniform generator, for example RANLUX (V115).

```
EXTERNAL urng
...
CALL RNORMX(RVEC,LEN,urng)
```

where urng is a uniform random number generator of standard calling sequence: CALL urng(VEC,LENG).
For example,

```
DIMENSION RVEC(10)
LEN = 10
EXTERNAL RANLUX
CALL RLUXGO(4,7675039,0,0)
  DO ...
     CALL RNORMX(RVEC,LEN,RANLUX)
```

would generate vectors of 10 Gaussian-distributed pseudorandom numbers of the highest quality. Note that initialization is now performed by the initializing entry for RANLUX, which is RLUXGO.

Method:

The method used to transform uniform deviates to Gaussian deviates is that known as the ratio of random deviates, discovered by Kinderman and Monahan, and improved by Leva (see **References**). The generation of one Gaussian random number requires at least two, and on average 2.74 uniform random numbers, as well as one floating-point division and on average 0.232 logarithm evaluations.

References:

1. J.L. Leva, A fast normal random number generator, ACM Trans. Math. Softw. **18** (1992) 449–453.
2. J.L. Leva, Algorithm 712. A normal random number generator, ACM Trans. Math. Softw. **18** (1992) 454–455.

•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 15.03.1994
Language : Fortran **Revised:**

Correlated Gaussian-distributed Random Numbers

CORGEN generates vectors of single-precision random numbers in a Gaussian distribution of mean zero and covariance matrix V. The generator must first be set up by a call to CORSET which transforms the covariance matrix V to an appropriate *square root* matrix C which is then used by CORGEN. CORGEN uses the Gaussian generator RNORML (V120) underneath, which in turn uses the uniform generator RANMAR (V113) underneath, so initialization is performed as in V113, but beware that this also initializes both RANMAR and RNORML! The code is portable Fortran, but the results are not guaranteed to be identical on all platforms as explained in RNORML (V120).

Structure:

SUBROUTINE Subprograms
User Entry Names: CORSET, CORGEN

Usage:

```
DIMENSION V(n,n), C(n,n), X(n)
CALL CORSET(V,C,n)
DO ...
  CALL CORGEN(C,X,n)
```

The call to CORSET transforms covariance matrix V to C. The call to CORGEN uses C to generate vector X of correlated Gaussian variables with covariance matrix V.

The limitation $n \leq 100$ is imposed by the dimension of an intermediate storage vector in CORSET.

Note that CORSET takes longer than CORGEN (for medium to large matrices). If it is desired to generate numbers according to a few different matrices, then each pair Vi, Ci must be separately dimensioned and saved as long as it is needed.

Method:

The square root method seems to be an old one whose origins are not known to the author (Ref. 1, p. 1182).

References:

1. F. James, Monte Carlo theory and practice, Rep. Prog. Phys. **43** (1980) 1145–1189.

•

Author(s) : F. James
Submitter :
Language : Fortran
Library: MATHLIB
Submitted: 15.09.1978
Revised:

Random Three-Dimensional Vectors

OBSOLETE

Please note that this routine has been obsoleted in CNL 223. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: RN3DIM (V131)

RAN3D generates random vectors, uniformly distributed over the surface of a sphere of a given radius.

Structure:

SUBROUTINE subprogram
User Entry Names: RAN3D
External References: NRAN (V105)

Usage:

```
CALL RAN3D(X,Y,Z,XLONG)
```

X,Y,Z	(REAL) A random 3-dimensional vector of length XLONG.
XLONG	(REAL) Length of the vector (to be specified on entry).

Method:

A random vector in the unit cube is generated using NRAN (V105) and is rejected if it lies outside the unit sphere. This rejection technique uses on average about 6 random numbers per vector, where only two are needed in principle. However, it is faster than the classical two-number technique which requires a square root, a sine, and a cosine.

•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 22.04.1996
Language : Fortran **Revised:**

Random Two- and Three-Dimensional Vectors

RN3DIM generates random vectors, uniformly distributed over the surface of a sphere of given radius.
RN2DIM generates random vectors, uniformly distributed over the circumference of a circle of given radius.

Structure:

SUBROUTINE subprogram
User Entry Names: RN2DIM, RN3DIM
External References: RANLUX (V115)

Usage:

```
CALL RN3DIM(X,Y,Z,XLONG)
```

X,Y,Z (REAL) A random 3-dimensional vector of length XLONG.

XLONG (REAL) Length of the vector (to be specified on entry).

```
CALL RN2DIM(X,Y,XLONG)
```

X,Y (REAL) A random 2-dimensional vector of length XLONG.

XLONG (REAL) Length of the vector (to be specified on entry).

Method:

A random vector in the unit cube is generated using RANLUX (V115) and is rejected if it lies outside the unit sphere. In the case of RN3DIM, this rejection technique uses on average about 6 random numbers per vector, where only two are needed in principle. However, it is faster than the classical two-number technique which requires a square root, a sine, and a cosine.

•

RNGAMA	CERN Program Library	V135

Author(s) : F. James, K.S. Kölbig — **Library:** MATHLIB
Submitter : — **Submitted:** 15.10.1994
Language : Fortran — **Revised:**

Gamma or Chi-Square Random Numbers

Function subprogram RANGAM generates a positive random number x according to the gamma distribution with parameter $p > 0$, i.e., according to the density

$$P(t < x < t + dt) \;=\; \frac{1}{\Gamma(p)}\, t^{p-1} e^{-t} dt.$$

A special case is the χ^2-distribution with N degrees of freedom

$$\chi^2(t < 2x < t + dt) \;=\; \frac{1}{\sqrt{2^N}\,\Gamma(\frac{1}{2}N)}\, t^{\frac{1}{2}N-1}\, e^{-\frac{1}{2}t}\, dt.$$

Structure:

FUNCTION subprogram
User Entry Names: RNGAMA
External References: RANLUX (V115), RNORMX (V120)

Usage:

In any arithmetic expression,

```
RNGAMA(P)
```

has the value of a gamma-distributed random number, where P $>$ 0 is of type REAL. The value of P may vary from call to call without influencing the efficiency.

Method:

For integral values of $p \leq 15$, the logarithm of the product of p uniform random numbers is used. For any value of $p > 15$, the Wilson-Hilferty approximation (a transformed normal distribution) is used. For all other p, Johnk's algorithm is used.

Notes:

The routine is fast for small integer values of p, and for $p > 15$, (one Gaussian random number and one square root, plus a few multiplications). Non-integral values of $p < 15$ are rather slow.

Examples:

```
CHI2 = 2*RNGAMA(0.5*N)
```

sets CHI2 to a random number distributed as χ^2 with N degrees of freedom.

•

Author(s) : D. Drijard, K.S. Kölbig
Submitter :
Language : Fortran
Library: MATHLIB
Submitted: 15.10.1994
Revised: 10.05.1995

Poisson Random Numbers

Subroutine subprogram `POISSN` generates a random integer $N > 0$ according to the Poisson distribution

$$Prob(N) = \frac{1}{N!} e^{-\mu} \mu^N,$$

where $\mu > 0$ (the mean) is a constant specified by the user.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `RNPSSN`, `RNPSET`
External References: `RANLUX` (V115), `RNORMX` (V120)

Usage:

```
CALL RNPSSN(AMU,N,IERR)
```

`AMU` (`REAL`) Mean μ.

`N` (`INTEGER`) The generated random number N, Poisson-distributed, with mean `AMU`.

`IERR` (`INTEGER`) Error flag.
$=$ `0` : Normal case.
$=$ `1` : `AMU` $\leq$ `0`.

For `AMU` $>$ `AMAX`, a (faster) normal approximation is made. The default value for `AMAX` is `AMAX` $=$ `88.0`. It can be reset (to smaller values only) by

```
CALL RNPSET(AMAX)
```

Timing:

Time increases with μ roughly as $\mu^{0.7}$.
•

Author(s) : D. Drijard, K.S. Kölbig
Submitter :
Library: MATHLIB
Submitted: 15.10.1994
Language : Fortran
Revised:

Binomial Random Numbers

Subroutine subprogram RNBNML generates a random integer $N > 0$ according to the binomial distribution

$$Prob(N = n) \;=\; \binom{M}{n} P^n \,(1 - P)^{M-n}$$

where the 'sample size' $M > 0$ and the probability P ($0 \leq P \leq 1$) are specified by the user.

Structure:

SUBROUTINE subprogram
User Entry Names: RNBNML
External References: RANLUX (V115)

Usage:

```
CALL RNBNML(M,P,N,IERR)
```

M (INTEGER) Sample size M.

P (REAL) Probability P.

N (INTEGER) The generated random number N, binomially distributed in the interval $0 \leq N \leq M$ with mean $P \times M$.

IERR (INTEGER) Error flag.
$= 0$: Normal case,
$= 1$: $P \leq 0$ or $P \geq 1$.

Notes:

RNBNML should not be used when M is 'large' (say > 100). The normal approximation is then recommended instead (with mean $P * M + 0.5$ and standard deviation $\sqrt{M * P * (1 - P)}$).
•

RNMNML	CERN Program Library	V138

Author(s) : D. Drijard, K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 15.10.1994
Language : Fortran **Revised:**

Multinomial Random Numbers

Subroutine subprogram RNMNML generates a vector of random integers $n_i > 0\,(i = 1, 2, \ldots, N)$ with probabilities p_i according to the multinomial distribution

$$Prob(n_1, n_2, \ldots, n_N) = \frac{(n_1 + n_2 + \cdots + n_N)!}{n_1!\, n_2! \cdots n_N!}\, p_1^{n_1}\, p_2^{n_2} \cdots p_N^{n_N}\,.$$

Structure:

SUBROUTINE subprogram
User Entry Names: RNMNML
External References: RANLUX (V115)

Usage:

```
CALL RNMNML(N,NSUM,PCUM,NVEC,IERR)
```

N (INTEGER) Number N of random integers n_i requested.

NSUM (INTEGER) $\sum_{i=1}^{N} n_i$, specified by the user.

PCUM (REAL) One-dimensional array of length $\geq$ N. Must contains, on entry, the (normalized) cumulative channel probabilities $\sum_{j=1}^{i} p_j$ in PCUM(i) (i $= 1, \ldots,$ N). In particular, PCUM(N) $=$ 1.

NVEC (INTEGER) One-dimensional array of length $\geq$ N. On exit, NVEC(i), (i = 1, ... ,N) contains the generated random integers.

IERR Error flag.
= 0 : Normal case,
= 1 : PCUM(i) < PCUM(i − 1) for one i al least,
= 2 : PCUM(N) $\neq$ 1.

Notes:

For $N = 2$, use RNBNML (V137).
•

Author(s) : F. James, K.S. Kölbig **Library:** MATHLIB
Submitter : **Submitted:** 20.03.1996
Language : Fortran **Revised:**

Random Numbers According to Any Histogram

RNHRAN generates random numbers distributed according to any empirical (one-dimensional) distribution. The distribution is supplied in the form of a histogram. If the distribution is known in functional form, FUNLUX (V152) should be used instead.

Structure:

SUBROUTINE subprograms
User Entry Names: RNHRAN, RNHPRE
Files Referenced: Printer
External References: LOCATR (E106), RANLUX (V115)

Usage:

```
CALL RNHPRE(Y,NBINS)                (once for each histogram)
CALL RNHRAN(Y,NBINS,XLO,XWID,XRAN)  (for each random number)
```

Y — Array of length NBINS at least containing the desired distribution as histogram bin contents on input to RNHPRE.

NBINS — Number of bins.

XLO — Lower edge of first bin.

XWID — Bin width.

XRAN — A random number returned by RNHRAN.

Method:

A uniform random number is generated using RANLUX (V115). The uniform number is then transformed to the user's distribution using the cumulative probability distribution constructed from his histogram. The cumulative distribution is inverted using a binary search for the nearest bin boundary and a linear interpolation within the bin. RNHRAN therefore generates a constant density within each bin.

Notes:

RNHPRE changes the values Y to form the cumulative distribution which is needed by RNHRAN. If Y already contains the cumulative distribution rather than the probability density, then RNHPRE should not be called, but in that case Y(NBINS) must be exactly equal to one. Numbers may be drawn from several different distributions in the same run by calling RNHRAN with different arrays Y1, Y2, etc. and (if desired) different values of NBINS, XLO, XWID (but always the same values for a given array Y). The routine RNHPRE should be used to initialize each array Yi.
The performance of the above method is nearly independent of the shape of the function or number of bins.

Error handling:

If the the input data to RNHPRE are not valid (some values negative or all values zero), an error message is printed, the input values are printed, and zero is returned instead of a random number. As many as five such messages may be printed to allow for possible errors in more than one distribution.
If RNHPRE is not called, and the input data are not already in cumulative form, RNHRAN performs the initialization itself and prints a warning message. RNHRAN recognizes that the data are not in cumulative form if Y(NBINS) $\neq$ 1.
•

HISRAN	CERN Program Library	V150

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 15.09.1978
Language : Fortran **Revised:**

Random Numbers According to Any Histogram

OBSOLETE

Please note that this routine has been obsoleted in CNL 223. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: RNHRAN (V149)

HISRAN generates random numbers distributed according to any empirical (one-dimensional) distribution. The distribution is supplied in the form of a histogram. If the distribution is known in functional form, FUNRAN (V151) should be used instead.

Structure:

SUBROUTINE subprograms
User Entry Names: HISRAN, HISPRE
Files Referenced: Printer
External References: LOCATR (E106), RNDM (V104)

Usage:

```
CALL HISPRE(Y,NBINS)                    (once for each histogram)
CALL HISRAN(Y,NBINS,XLO,XWID,XRAN)      (for each random number)
```

Y	Array of length NBINS at least containing the desired distribution as histogram bin contents on input to HISPRE.
NBINS	Number of bins.
XLO	Lower edge of first bin.
XWID	Bin width.
XRAN	A random number returned by HISRAN.

Method:

A uniform random number is generated using RNDM (V104). (The user may therefore use RDMOUT and RDMIN (V104) to restart a run.) The uniform number is then transformed to the user's distribution using the cumulative probability distribution constructed from his histogram. The cumulative distribution is inverted using a binary search for the nearest bin boundary and a linear interpolation within the bin. HISRAN therefore generates a constant density within each bin.

Notes:

HISPRE changes the values Y to form the cumulative distribution which is needed by HISRAN. If Y already contains the cumulative distribution rather than the probability density, then HISPRE should not be called, but in that case Y(NBINS) must be exactly equal to one. Numbers may be drawn from several different distributions in the same run by calling HISRAN with different arrays Y1, Y2, etc. and (if desired) different values of NBINS, XLO, XWID (but always the same values for a given array Y). The routine HISPRE should be used to initialize each array Yi.

The performance of the above method is nearly independent of the shape of the function or number of bins.

Error handling:

If the the input data to `HISPRE` are not valid (some values negative or all values zero), an error message is printed, the input values are printed, and zero is returned instead of a random number. As many as five such messages may be printed to allow for possible errors in more than one distribution.

If `HISPRE` is not called, and the input data are not already in cumulative form, `HISRAN` performs the initialization itself and prints a warning message. `HISRAN` recognizes that the data are not in cumulative form if $\mathtt{Y(NBINS)} \neq \mathtt{1}$.

•

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 27.11.1984
Language : Fortran **Revised:**

Random Numbers According to Any Function

OBSOLETE

Please note that this routine has been obsoleted in CNL 219. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: FUNLUX (V152)

FUNRAN generates random numbers distributed according to any (one-dimensional) distribution $f(x)$. The distribution is supplied by the user in the form of a FUNCTION subprogram. If the distribution is known as a histogram only, HISRAN (V150) should be used instead.

Structure:

SUBROUTINE subprograms
User Entry Names: FUNRAN, FUNPRE
Internal Entry Names: FUNZER
Files Referenced: Printer
External References: GAUSS (D103), RNDM (V104), user-supplied FUNCTION subprogram
COMMON Block Names and Lengths: /FUNINT/ 1

Usage:

```
CALL FUNPRE(F,FSPACE,XLOW,XHIGH)      (once for each function)
CALL FUNRAN(FSPACE,XRAN)              (for each random number)
```

F	(REAL) A name of a FUNCTION subprogram declared EXTERNAL in the calling program. This subprogram must calculate the (non-negative) density function f(X), for all X in the interval XLOW $\leq$ X $\leq$ XHIGH.
FSPACE	(REAL) One-dimensional array of length 100.
XLOW	(REAL) Lower limit of the requested interval.
XHIGH	(REAL) Upper limit of the requested interval.
XRAN	(REAL) A random number returned by FUNRAN.

A call to FUNPRE calculates the percentiles of F between XLOW and stores them into the array FSPACE.

Method:

In FUNPRE, the percentiles are calculated using a combination of trapezoidal and Gaussian integration to a rather high accuracy, which is printed out by FUNPRE. If the desired accuracy is not obtained, an warning is printed in addition.

Subroutine FUNRAN finds the desired random number by calling RNDM (V104) and doing a 4-point interpolation on FSPACE to transform the uniform random number to the distribution specified. This method produces quite accurately distributed numbers even when the function F is badly skew or spiked as long as the width of a spike is not less than 1/1000 of the total range.

Error handling:

An error message is printed

- – if the integral of the user-supplied function `F` is zero or negative,
- – if `XLOW` $\geq$ `XHIGH`,
- – if `F(X)` $<$ `0` somewhere between `XLOW` and `XHIGH`.

Notes:

Some additional information which may be of use is contained in

```
COMMON / FUNINT/ FINT
```

After a call to `FUNPRE`, `FINT` contains the integral of `F` from `XLOW` to `XHIGH`.
After a call to `FUNRAN`, `FINT` contains the integral of `F` from `XLOW` to `XRAN`, divided by the total integral to `XHIGH` (i.e., it will be a number uniformly distributed between zero and one).

•

FUNLUX	CERN Program Library	V152

Author(s) : F. James **Library:** MATHLIB
Submitter : **Submitted:** 22.02.1996
Language : Fortran **Revised:**

Random Numbers According to Any Function

FUNLUX generates random numbers distributed according to any (one-dimensional) distribution $f(x)$. The distribution is supplied by the user in the form of a FUNCTION subprogram. If the distribution is known as a histogram only, HISRAN (V150) should be used instead.

Structure:

SUBROUTINE subprograms
User Entry Names: FUNLUX, FUNLXP
Internal Entry Names: FUNPCT, FUNLZ
Files Referenced: Printer
External References: RADAPT (D102), RANLUX (V115), user-supplied FUNCTION subprogram
COMMON Block Names and Lengths: /FUNINT/ 1

Usage:

```
CALL FUNLXP(F,FSPACE,XLOW,XHIGH)      (once for each function)
CALL FUNLUX(FSPACE,XRAN,LEN)          (for each vector of random numbers)
```

F (REAL) A name of a FUNCTION subprogram declared EXTERNAL in the calling program. This subprogram must calculate the (non-negative) density function f(X), for all X in the interval XLOW $\leq$ X $\leq$ XHIGH.

FSPACE (REAL) One-dimensional array of length 200.

XLOW (REAL) Lower limit of the requested interval.

XHIGH (REAL) Upper limit of the requested interval.

XRAN (REAL) A vector of random numbers returned by FUNRAN.

LEN (INTEGER) Length of the vector XRAN.

A call to FUNLXP calculates the percentiles of F between XLOW and XHIGH and stores them into the array FSPACE.

Method:

In FUNLXP, the 100 percentiles of the integral of f(X) are calculated using a combination of trapezoidal and Gaussian integration to a rather high accuracy, which is printed out by FUNLXP. Then both the left-hand and right-hand 2 percentiles are expanded to 50 percentiles each in order to cater for functions with long tails. If the desired accuracy is not obtained, a warning is printed in addition.
Subroutine FUNLUX finds the desired random number by calling RANLUX (V115) and doing a 4-point interpolation on FSPACE to transform the uniform random number to the distribution specified. This method produces quite accurately distributed numbers even when the function F is badly skew or spiked as long as the width of a spike is not less than 1/1000 of the total range.

Error handling:

An error message is printed

– if the integral of the user-supplied function F is zero or negative,

– if XLOW $\geq$ XHIGH,

– if F(X) $<$ 0 somewhere between XLOW and XHIGH.

Notes:

Some additional information which may be of use is contained in

```
COMMON / FUNINT/ FINT
```

After a call to FUNLXP, FINT contains the integral of F from XLOW to XHIGH.

After a call to FUNLUX, FINT contains the integral of F from XLOW to XRAN(LEN), divided by the total integral to XHIGH (i.e., it will be a number uniformly distributed between zero and one).

•

Author(s) : F. Beck, T. Lindelöf
Submitter : K.S. Kölbig
Language : Fortran

Library: MATHLIB
Submitted: 15.09.1978
Revised: 07.06.1992

Permutations and Combinations

Successive calls to subroutine subprogram `PERMU` will generate all permutations of a set of integers of total length N consisting of n_1 repetitions of the integer 1, followed by n_2 repetitions of the integer $2, \ldots$ etc, concluding with n_m repetitions of the integer m, where $\sum_{j=1}^{m} n_j = N$.

Subroutine subprogram `PERMUT` generates *directly* a single member of the set of all lexicographically ordered permutations of the first integers $1, 2, \ldots, N$, as specified by its lexicographical ordinal.

Successive calls to subroutine subprogram `COMBI` will generate all the $\binom{N}{J}$ possible combinations without repetition of $J \leq N$ integers from the set $\{1, 2, \ldots, N\}$.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `PERMU`, `PERMUT`, `COMBI`
Files Referenced: `Unit 6`

Usage:

Subroutine PERMU:

```
CALL PERMU(IA,N)
```

`IA` (`INTEGER`) One-dimensional array of length $\geq$ `N`. On entry, `IA(i)`, (`i` $= 1, 2, \ldots,$ `N`), must contain the initial set of integers to be permuted (see **Examples**). A call with `IA(1)` $= 0$ will place the set $\{1, 2, \ldots, \mathtt{N}\}$ in `IA`. On exit, `IA` contains the "next" permutation. If all the permutations have been generated, the next call sets `IA(1)` $= 0$.

`N` (`INTEGER`) Length of the set to be permuted.

Subroutine PERMUT:

```
CALL PERMUT(NLX,N,IP)
```

`NLX` (`INTEGER`) Lexicographical ordinal of the permutation desired.

`N` (`INTEGER`) Length of the set to be permuted.

`IP` (`INTEGER`) One-dimensional array of length $\geq$ `N`. On exit, `IP(i)`, (`i` $= 1, 2, \ldots,$ `N`), contains the `NLX`-th lexicographically ordered permutation of the integers $1, 2, \ldots, N$ (see **Examples**).

Subroutine COMBI:

```
CALL COMBI(IC,N,J)
```

`IC` (`INTEGER`) One-dimensional array of length $\geq$ `N` $+ 1$. The first call must be made with `IC(1)` $= 0$. This generates the first combination `IC(i)` $=$ `i`, (`i` $= 1, 2, \ldots,$ `J`). Each successive call generates a new combination and places it in the first `J` elements of `IC`. If all the combinations have been generated, the next call sets `IC(1)` $= 0$.

`N` (`INTEGER`) Length of the set from which the combinations are taken.

`J` (`INTEGER`) Length of the combinations.

Examples:

1. Consider the following set of $N = 12$ objects, only 8 are different:

$$\{y_1, y_2, y_3, y, y, r_1, r_2, r, r, b, b, b\}.$$

This set consists of $m = 8$ sequences of length $n_1 = n_2 = n_3 = n_5 = n_6 = 1$, $n_4 = n_7 = 2$, $n_8 = 3$. Thus, in order to get the possible permutations, set

$$\mathtt{IA} = \{\mathtt{1\ 2\ 3\ 4\ 4\ 5\ 6\ 7\ 7\ 8\ 8\ 8}\}$$

before calling `PERMU(IA,12)` the first time.

2. To generate all permutations of N indistinguishable objects, set $\mathtt{IA(1)} = \mathtt{0}$, which is equivalent to $\mathtt{IA(i)} = \mathtt{i}$, $(\mathtt{i} = 1, 2, \ldots, \mathtt{N})$, before calling `PERMU(IA,N)` the first time.

3. To compute the, lexicographically second, third and last $(4! = 24)$ permutions of the set $\{1, 2, 3, 4\}$:

`CALL PERMUT( 2,4,IP)`	sets	$\mathtt{IP} = \{1, 2, 4, 3\}$
`CALL PERMUT( 3,4,IP)`	sets	$\mathtt{IP} = \{1, 3, 2, 4\}$
`CALL PERMUT(24,4,IP)`	sets	$\mathtt{IP} = \{4, 3, 2, 1\}$

4. To generate and print all 20 combinations of 3 integers from the set $\{1, 2, 3, 4, 5, 6\}$ one could write:

```
      ...
      IA(1)=0
    1 CALL COMBI(IC,6,3)
      IF(IC(1) .NE. 0) THEN
       PRINT *, IC(1),IC(2),IC(3)
       GO TO 1
      ENDIF
      ...
```

Restrictions:

`PERMUT`: $1 \leq \mathtt{NLX} \leq \mathtt{N}!$, $\mathtt{N} \leq 12$.
`COMBI`: $\mathtt{J} \leq \mathtt{N}$.

Error handling:

If any of the above conditions is not satisfied, a message is written on `Unit 6`.

Notes:

1. If $\mathtt{N} \leq 0$ or $\mathtt{J} \leq 0$, the subprograms return control without action.

2. The number of distinct permutations of a set of N numbers which can be decomposed into m groups of $n_1, n_2, \ldots, n_m$ indistinguishable elements is given by

$$\frac{N!}{n_1!\, n_2! \cdots n_m!}$$

where $n_1 + n_2 + \cdots + n_m = N$. This number can become large even for seemingly simple cases, e.g. in Example 1 above,

$$\frac{12!}{1!\,1!\,1!\,2!\,1!\,1!\,2!\,3!} = 19958400.$$

•

Author(s) : J. Zoll
Submitter : C. Letertre
Language : Fortran or Assembler

Library: KERNLIB
Submitted: 01.03.1968
Revised: 16.09.1991

Preset Parts of an Array

These routines fill each word of an array with zero, 'blank', or a quantity given in the argument list.

Structure:

SUBROUTINE subprograms
User Entry Names: UBLANK, UZERO, UFILL

Usage:

Required $0 < J1 \leq J2$.

```
CALL UZERO(A,J1,J2)
```

sets A(J1) until A(J2) to zero.

```
CALL UBLANK(A,J1,J2)
```

sets A(J1) until A(J2) to BCD blank.

```
CALL UFILL(A,J1,J2,STUFF)
```

loads A(J1) until A(J2) with the contents of STUFF.

•

Author(s) : R.K. Böck, C. Letertre
Submitter :
Language : Fortran or Assembler

Library: KERNLIB
Submitted: 01.03.1968
Revised: 16.09.1991

Copy an Array

These routines copy a continuous string of words into a continuous set of locations.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `UCOPY`, `UCOPIV`, `UCOPYN`, `UCOPY2`, `USWOP`
External References: `LOCF` (N100) (Fortran version of `UCOPY2` only)

Usage:

```
CALL UCOPY(A,X,N)
```

copies `N` words from `A` into `X`; the beginning of `A` may overlap the end of `X`.

```
CALL UCOPY2(A,X,N)
```

copies `N` words from `A` into `X`, any overlap is allowed.

```
CALL UCOPYN(IA,IX,N)
```

transfers into `IX` the negative values of `N` integer words from `IA`; the beginning of `IA` may overlap the end of `IX`. (For numbers of type `REAL`, use `VCOPYN` (F121).)

```
CALL UCOPIV(A,X,N)
```

copies `N` words from `A` into `X`, in reverse order, i.e. $\mathtt{X(1)} = \mathtt{A(N)}, \ldots, \mathtt{X(N)} = \mathtt{A(1)}$. No overlapping is allowed.

```
CALL USWOP(A,B,N)
```

exchanges the first $\mathtt{N} \geq 0$ words of arrays `A` and `B`. `A` and `B` must not overlap.

For $\mathtt{N} = 0$ the above routines act as 'do-nothing'.

•

UCOCOP	CERN Program Library	V302

Author(s) : F. Bruyant **Library:** KERNLIB
Submitter : C. Letertre **Submitted:** 21.08.1971
Language : Fortran or Assembler **Revised:** 16.09.1991

Copy a Scattered Vector

UCOCOP and UDICOP copy the contents of a scattered vector into a new scattered vector.

Structure:

SUBROUTINE subprograms
User Entry Names: UCOCOP, UDICOP

Usage:

```
CALL UCOCOP(A,X,IDO,IW,NA,NX)
CALL UDICOP(A,X,IDO,IW,NA,NX)
```

extract IDO times IW consecutive words from A, every NA words, and place them into X, every NX words. Both routines have the same effect if the vectors A and X do not overlap. UCOCOP allows concentration, UDICOP allows dilation of a vector *in situ*.

For IDO = 0 or IW = 0, the routines act as 'do-nothing'.

Examples:

```
DIMENSION IA(14),IX(12)
DATA IA /1,2,3,4, 5,6,7,8, 9,10,11,12, 13,14/

CALL UCOCOP(IA,IX,4,2,4,3)
CALL UCOCOP(0,IX(3),4,1,0,3)
```

gives

```
IX = 1,2,0, 5,6,0, 9,10,0, 13,14,0
```

•

Author(s) : J. Zoll, C. Letertre
Submitter :
Language : Fortran or Assembler

Library: KERNLIB
Submitted: 01.03.1968
Revised: 16.09.1991

Search a Vector for a Given Element

These routines all search through a vector for a given element. The calling sequences and the default returns are different.

Structure:

FUNCTION subprograms
User Entry Names: IUCOMP, IUCOLA, IUFIND, IUFILA, IUHUNT, IULAST

Usage:

```
IUCOMP(IT,IVEC,N)     or     IUCOLA(IT,IVEC,N)
```

returns the relative address in the array IVEC of the first (or the last) word which is equal to IT, or zero if IT is not contained in IVEC(1), ... ,IVEC(N) or if N = 0.

```
IUFIND(IT,IVEC,JL,JR)     or     IUFILA(IT,IVEC,JL,JR)
```

returns the relative address in the array IVEC of the first (or the last) element between IVEC(JL) and IVEC(JR) (JL $\leq$ JR) which equals IT, or JR + 1 if IT is not contained in IVEC(JL),IVEC(JL+1), ... , IVEC(JR) or if JL > JR.

```
IUHUNT(IT,IVEC,N,INC)
```

returns the relative address of the first word among IVEC(1),IVEC(INC+1),IVEC(2*INC+1), ... of array IVEC (the search does not go beyond IVEC(N)) which equals IT, or zero if IT is not found or if N = 0.

```
IULAST(IT,IVEC,N)
```

returns the relative address of the last word which, in the array IVEC of N elements, is not equal to IT, or zero if N = 0 or if all elements in IVEC equal IT.

Notes:

IVEC and IT above may be of type INTEGER or REAL, but the comparison is done in type INTEGER.
•

Author(s) : J. Zoll, K.S. Kölbig **Library:** KERNLIB
Submitter : **Submitted:** 15.03.1976
Language : Fortran **Revised:** 15.02.1989

Adjusting an Angle to Another Angle

Function subprogram `PROXIM` computes, for two angles α, β given as arguments, and by adding a suitable multiple of 2π to β, an angle β^* such that

$$\alpha - \pi \leq \beta^* \leq \alpha + \pi.$$

Structure:

`FUNCTION` subprogram
User Entry Names: `PROXIM`

Usage:

In any arithmetic expression,

```
PROXIM(B,A)
```

has the value β^* for `B` $= \beta$ and `A` $= \alpha$. `PROXIM`, `B` and `A` are of type `REAL` and in radians.

Notes:

The Fortran statement function

```
PROXIM(B,A)=B+C1*ANINT(C2*(A-B))
```

with `C1` $= 2\pi$, `C2` $=$ `1/C1` has the same effect.

•

Author(s) : A. Regl
Submitter : H. Grote
Language : Fortran

Library: MATHLIB
Submitted: 01.02.1974
Revised: 15.09.1978

Find Compatible Node-Nets in an Incompatibility Graph

`GRAPH` finds all compatible sets of events (nodes) in an incompatibility graph (in which incompatible events or nodes are connected). It is useful, for example, in track-matching programs for eliminating spurious tracks.

On each call, one compatible node-set is returned. The user may decide in the first call whether the solutions should be evaluated over the whole graph or subgraph by subgraph. Indications on "end-of-graph" and, if applicable, "end-of-subgraph" are given.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `GRAPH`
Internal Entry Names: `PGRAPH`, `GETBIT`, `SETBIT`, `TUP`, `IGET`, `TREVNI`
External References: `JBIT` (M421), `SBIT` (M421), `JBYT` (M421), `SBYT` (M421), `UFILL` (V300), `UZERO` (V300)
`COMMON` Block Names and Lengths : `/BITSXB/ 2`

Usage:

See **Long Write-up**.

•

Author(s) : K.S. Kölbig, F. Lamarche, C. Leroy **Library:** MATHLIB
Submitter : **Submitted:** 07.06.1992
Language : Fortran **Revised:**

Volume of Intersection of a Circular Cylinder with a Sphere

Function subprograms `RVNSPC` and `DVNSPC` calculate the volume of intersection $V(r, \rho, d)$ of a circular cylinder of radius $r \geq 0$ with a sphere of radius $\rho \geq 0$, the distance from the center of the sphere to the axis of the cylinder being $d \geq 0$.

This volume is given by

$$V(r, \rho, d) \;=\; 2 \iint \sqrt{\rho^2 - x^2 - y^2}\, dx dy,$$

where the integration is performed over the intersection, if any, of the two circular disks $(x - d)^2 + y^2 \leq r^2$ and $x^2 + y^2 \leq \rho^2$. If $r \neq 0 \wedge \rho \neq 0 \wedge d < r + \rho$ this is equal to

$$V(r, \rho, d) \;=\; 4 \int_{\max(d-r,-\rho)}^{\min(d+r,\rho)} \int_0^{\min\left(\sqrt{r^2-(x-d)^2},\sqrt{\rho^2-x^2}\right)} \sqrt{\rho^2 - x^2 - y^2}\, dx dy.$$

Otherwise $V(r, \rho, d) = 0$.

On CDC and Cray computers, the double-precision version `DVNSPC` is not provided.

Structure:

`FUNCTION` subprograms
User Entry Names: `RVNSPC`C347, `DVNSPC`C347
External References: `DELI3C` (C347), `DELIKC` (C347), `DELIEC` (C347)

Usage:

In any arithmetic expression,

`RVNSPC(R,RHO,D)` or `DVNSPC(R,RHO,D)` has the value $V(\mathtt{R}, \mathtt{RHO}, \mathtt{D})$.

`RVNSPC` is if type `REAL`, `DVNSPC` is of type `DOUBLE PRECISION`, and `R`, `RHO` and `D` are of the same type as the function name.

Method:

The integral given above can be expressed in closed form in terms of complete elliptic integrals of the first, second, and third kind. For details see Ref. 1.

Notes:

Any negative sign in the parameters is ignored.

In the single-precision version `RVNSPC` on machines other than CDC or Cray, the complete elliptic integrals are calculated inside the subprogram. This version, faster than `DVNSPC`, is intended mainly for applications in experimental physics, where its limited accuracy of about 6 digits can be tolerated.

References:

1. F. Lamarche and C. Leroy, Evaluation of the volume of intersection of a sphere with a cylinder by elliptic integrals, Computer Phys. Comm. **59** (1990) 359–369.

•

Author(s) : C.H. Moore, D.C. Carey
Submitter : C. Iselin
Language : Fortran 4
Library: PGMLIB
Submitted: 27.11.1984
Revised:

Transport, Second-Order Beam Optics

TRSPRT is a first- and second-order matrix multiplication program for the design of magnetic beam transport systems. It has been in use in various versions since 1963. The present version, written by D.C. Carey at FNAL and extensively modified at CERN is described in CERN 80-04, NAL 91 and SLAC 91. It includes both first- and second-order fitting capabilities. A beam line is described as a sequence of elements. Such elements may represent magnets or the intervals separating them, but also specify calculations to be done, or special conditions to be applied. The program works in six-dimensional phase space $(x, x', y, y', l, dp/p)$; it is therefore also capable of calculating coupling between planes.

Structure:

Complete PROGRAM
User Entry Names: TRSPRT
Files Referenced: INPUT, OUTPUT,
External References: UBUNCH (M409), ABEND (Z035), DATIMH (Z007)

Usage:

See **Long Write-up**. TRSPRT is accessed from PGMLIB as described in section 'Execution of Complete Programs, PGMLIB' in Chapter 1 of the Program Library Manual.

Source:

SLAC and FNAL, USA

References:

1. K.L. Brown, D.C. Carey, C. Iselin and F. Rothacker, Designing Charged Particle Beam Transport Systems, CERN 80-04 (1980)

A copy of Ref. 1 is available as **Long Write-up**.

•

TURTLE	CERN Program Library	W151

Author(s) : D.C. Carey, C. Iselin
Submitter : C. Iselin
Language : Fortran 4

Library: PGMLIB
Submitted: 01.07.1974
Revised: 27.11.1984

Beam Transport Simulation, Including Decay

TURTLE is designed to simulate charged particle beam transport systems. It allows evaluation of the effects of aberrations in beams with a *small phase space volume*. These include higher-order chromatic aberrations, non-linearities in magnetic fields and higher-order geometric aberrations due to the accumulation of second-order effects. The beam at any point in the system may be represented by one- or two-dimensional histograms. TURTLE also provides a simulation of decay of pions or kaons into muons and neutrinos.

TURTLE uses the same input format as TRSPRT (W150). An input stream set up for TRSPRT can thus be used for TURTLE with only a few additions.

Structure:

Complete PROGRAM
User Entry Names: TURTLE
Files Referenced: INPUT, OUTPUT
External References: RANF (G900), UBUNCH (M409), TIMEL (Z007), ABEND (Z035)

Usage:

See **Long Write-up**. TURTLE is accessed from PGMLIB as described in 'Execution of Complete Programs, PGMLIB' in Chapter 1 of the Program Library Manual. Page 50 of the **Long write-up** is obsolete.

Source:

FNAL. The parts concerning decay have been written at CERN.

References:

1. K.L. Brown and C. Iselin DECAY TURTLE, a Computer Program for Simulating Charged Particle Beam Transport Systems, including Decay Calculations, CERN 74-2 (1974).

A copy of Ref. 1 is available as **Long Write-up**.

•

Author(s) : F. James
Library: POOL
Submitter :
Submitted: 13.11.1972
Language : Fortran
Revised: 01.12.1981

General Monte-Carlo Phase-Space

FOWL uses the Monte-Carlo method to calculate phase space distributions arising from particle interactions. The events are generated according to Lorentz-invariant phase space, and after each event the user may calculate (in a subroutine) all quantities (effective masses, angles, moments, delta squared, etc.) whose distribution he wants.

Moreover, the user may calculate, for each quantity, a weight (or 'matrix element', for example a Breit-Wigner) which is in general a function of the kinematic quantities for the event. In addition, one can investigate the effects of cutoffs, selections or biases in an actual experiment by imposing the same selections on events in FOWL. The program then prints histograms and/or scatter plots of quantities calculated by the user.

Structure:

SUBROUTINE subprogram
User Entry Names: FOWL
Files Referenced: INPUT, OUTPUT, PUNCH
External References: RNDM (V104), UBLANK (V300), IUCHAN (Y201), DATIME (Z007),
user-supplied subroutine USER.

Usage:

See **Long Write-up**.

Source:

Event generator GENEV was adapted by K. Kajantie from a program by G. Lynch.

•

GENBOD	CERN Program Library	W515

Author(s) : F. James **Library:** POOL
Submitter : **Submitted:** 20.10.1975
Language : Fortran **Revised:**

N-Body Monte-Carlo Event Generator

GENBOD generates a multi-particle weighted event according to Lorentz-invariant Fermi phase space. It is a modification of the routine GENEV (in FOWL (W505)) and uses the method of Raubold and Lynch (see Ref. 1). The total CM energy as well as the number and masses of the outgoing particles are specified by the user, but may be changed from event to event. GENBOD generates the CM vector momenta (and energies) of the outgoing particles and gives the weight which must be associated with each event. The weight may then be multiplied by any 'matrix element' or geometrical detection function calculated by the user.

Structure:

SUBROUTINE subprogram
User Entry Names: GENBOD
Files Referenced: Printer
External References: FLPSOR (M103), RNDM (V104), PDK (W505), ROTES2 (W505)
COMMON Block Names and Lengths : /GENIN/ 21, /GENOUT/ 91

Usage:

```
COMMON /GENIN /NP,TECM,AMASS(18),KGENEV
COMMON /GENOUT/ PCM(5,18),WT
CALL GENBOD
```

Input:

NP	(INTEGER) Number of outgoing particles ($2 \leq$ NP ≤ 18).
TECM	(REAL) Total CM energy.
AMASS	(REAL) Array where element I contains the mass of the I-th outgoing particle.
KGENEV	(INTEGER) $= 1$ for cross section constant with energy, $= 2$ for Fermi energy dependence.

Output:

PCM(1,I)	(REAL) P_x of I-th particle.
PCM(2,I)	(REAL) P_y of I-th particle.
PCM(3,I)	(REAL) P_z of I-th particle.
PCM(4,I)	(REAL) Energy of I-th particle.
PCM(5,I)	(REAL) P of I-th particle.
WT	(REAL) Weight of the event.

See also the **Long Write-up** for FOWL (W505).

References:

1. F. James, Monte Carlo Phase Space, CERN 68-15 (1968)

•

Author(s) : J. Zoll, P. Rastl
Submitter : C. Letertre
Language : Fortran or Assembler

Library: KERNLIB
Submitted: 01.09.1969
Revised: 16.09.1991

Find Histogram-Channel

`IUCHAN`, `IUBIN`, `IUHIST` all find the histogram-channel for a given quantity in the same way. They differ only slightly in the way in which the parameters are passed.

Structure:

`FUNCTION` subprograms
User Entry Names: `IUCHAN`, `IUBIN`, `IUHIST`

Usage:

All routines need the the following parameters:

`X`	(`REAL`) Quantity to be histogrammed.
`XLOW`	(`REAL`) Lower limit of the histogram.
`DX`	(`REAL`) Channel width.
`NX`	(`INTEGER`) Number of channels.

and they return the channel number $\mathtt{N} = (\mathtt{X} - \mathtt{XLOW})/\mathtt{DX} + 1 + \varepsilon$ normally, or $\mathtt{N} = 0$ for underflow ($\mathtt{X} < \mathtt{XLOW}$), or $\mathtt{N} = \mathtt{NX} + 1$ for overflow ($\mathtt{X} \geq \mathtt{XLOW} + \mathtt{NX} * \mathtt{DX}$).
$\varepsilon > 0$ is a small bias to counteract rounding effects when `X` is exactly on a bin edge, a likely and serious problem when compressed data are histogrammed.
$\varepsilon = 10^{-5}$ on 32-bit machines, $\varepsilon = 10^{-6}$ on machines with a larger word size.

Function IUCHAN:

```
N = IUCHAN(X,XLOW,DX,NX)
```

Functions IUBIN and IUHIST:

```
DIMENSION PAR(3)
EQUIVALENCE(NX,PAR(1))
LOGICAL SPILL

N = IUBIN (X,PAR,SPILL)
N = IUHIST(X,PAR,SPILL)
```

with

`PAR`	Histogram parameters: $\mathtt{PAR(1)} \equiv \mathtt{NX}$ $\mathtt{PAR(2)} \equiv \mathtt{DX}$ (for `IUBIN`), or $\equiv 1/\mathtt{DX}$ reciprocal of the channel width (for `IUHIST`). $\mathtt{PAR(3)} \equiv \mathtt{XLOW}$
`SPILL`	(`LOGICAL`) Flag set to `.TRUE.` or `.FALSE.` depending on whether `X` is outside or inside the histogram.

•

HBOOK	CERN Program Library	Y250

Author(s) : R. Brun, I. Ivanchenko, P. Palazzi **Library:** PACKLIB
Submitter : **Submitted:**
Language : Fortran **Revised:**

Statistical Analysis and Histogramming

HBOOK offers as basic options the booking, filling and printing of a histogram, scatter plot or table. Other available facilities are:

- Projections and slices of scatter plots and tables.
- Wide choice of editing options (what to print and how).
- Easy access to the information.
- Operations on histograms (arithmetic, smoothing, filling, fitting).
- Packing of several channels in 1 computer word/or extension of the memory on disk file, to allow simultaneous handling of a very large number of plots.

Structure:

SUBROUTINE and FUNCTION subprograms

Usage:

See **Long Write-up**.
•

Author(s) : O. Couet
Submitter :
Language : Fortran
Library: GRAFLIB
Submitted: 01.03.1976
Revised: 01.11.1994

HPLOT : HBOOK Graphics Interface for Histogram Plotting

HPLOT is a FORTRAN-callable facility for producing HBOOK (Y250) output on all kind of graphic devices. The output is of a quality suitable for publications.

Structure:

SUBROUTINE subprograms

Usage:

A full description of the system is given in the **HIGZ-HPLOT** Manual (Q120, Y251). The full HPLOT facilities are available in the PAW (Q121) system.

•

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 19.09.1991
Language : Fortran **Revised:**

Print KERNLIB Version Numbers

`KERNGT` prints the titles of the PAM-files which have been used to make the general part of `KERNLIB`.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `KERNGT`
Files Referenced: Parameter

Usage:

```
CALL KERNGT(LUN)
```

with:

LUN Fortran logical unit number for printing, if zero: use 'standard output'.

Examples:

```
CALL KERNGT(0)
```

gives something like:

```
KERNGT.  KERNLIB from:  KERNAPO  1.23  910719 13.00
                        KERNFOR  4.29  910731 19.17
```

•

DATIME	CERN Program Library	Z007

Author(s) : See below **Library:** KERNLIB
Submitter : **Submitted:** 15.01.1977
Language : Fortran or C or Assembler **Revised:** 18.09.1991

Job Time and Date

Authors: J. Harms, E. Jansen, A. Michalon, J. Zoll, A. Berglund, T. Cass, C. Wood, H. Renshall.

The DATIME package interfaces with the system of any particular machine to obtain the current calendar date and time, as well as the central processor time used by and remaining to the job.

Structure:

SUBROUTINE subprograms
User Entry Names: DATIME, DATIMH, TIMEX, TIMEL, TIMED, TIMEST
External References: Machine dependent
COMMON Block Names: /SLATE/ ISL(40)

Usage:

```
CALL DATIME(ID,IT)
```

returns decimal INTEGER date and time: ID=yymmdd, IH=hhmm. It also stores the components into /SLATE/ as small integers:

$$\mathtt{ISL}(1) = \mathtt{19yy},\quad \mathtt{ISL}(2) = \mathtt{mm},\quad \mathtt{ISL}(3) = \mathtt{dd},\quad \mathtt{ISL}(4) = \mathtt{hh},\quad \mathtt{ISL}(5) = \mathtt{mm},\quad \mathtt{ISL}(6) = \mathtt{ss}$$

for convenience of further processing by the user.

```
CALL DATIMH(ND,NT)
```

returns Hollerith date and time: ND = 8Hdd/mm/yy and NT = 8Hhh.mm.ss

```
CALL TIMEX(T)
```

returns the execution time used by the job so far; T is the central processor time in seconds, a REAL number with fractional part. In supported interactive systems the time returned is that relative to the first call to TIMEST.

```
CALL TIMEL(T)
```

returns the execution time remaining until time-limit; T in seconds as for TIMEX. In supported interactive systems the time returned is the time left until the time-limit set by the first call to TIMEST. See **Note 4** below.

```
CALL TIMED(T)
```

returns the execution time interval since the last call to TIMED; T in seconds as for TIMEX.

```
CALL TIMEST(TLIM)
```

This routine is necessary to initialise the timing operations in the interactive mode of VM-CMS. In other systems (including VM-CMS batch) it is a dummy do-nothing routine.

It must be called once (subsequent calls are ignored) before any calls to `TIMEX` and `TIMEL`. Before this routine is called `TIMEX` will return zero and `TIMEL` will return `999.0`. `TLIM` is an input floating point value which will be used inside `TIMEL` as if it were the job time-limit. The first call to `TIMEST` also establishes the time origin for subsequent calls to `TIMEX` and `TIMEL`.

Accuracy:

IBM: The RMS error returned in consecutive calls to `TIMED` without any intermediate code is of the order of 3 μsec on the the CERN IBM 3090 with a minimum time for one call of 20 μsec. The timing distribution has a long tail, however, and any individual measurement could take as long as four or five times this value. `TIMEX` is accurate to within a tenth of a second and `TIMEL` only to the nearest second.

Notes:

1. The symbols `yy,mm,dd,hh,mm,ss` used above stand for the two decimal digits of *year, month, day, hours, minutes, seconds*.
2. `NT` and `ND` in the call to `DATIMH` are 2-word vectors on machines with a character-capacity of less than 8 characters per word.
3. The information returned by these routines is obtained by a system request. On some machines this is expensive in real time, so one should avoid too many calls, to `TIMEL` in particular.
4. Some machine/operating system configurations do not have a value for timelimit, for example interactive work under VM-CMS (IBM) or VMS (VAX) or no-limit batch job classes under VMS. In these cases a constant time-left of 999.0 seconds is returned, unless the time limit has been set with `TIMEST`.

Examples:

Suppose the date is Sept 16, 1976, and the time of day 19h 24m 55s.

```
CALL DATIME(ID,IT)
```

returns `ID` = `760916`, `IT` = `1924`, `ISL` = `1976, 9, 16, 19, 24, 55`

```
CALL DATIMH(ND,NT)
```

returns `ND` = `8H16/09/76` and `NT` = `8H19.24.55`.

•

CALDAT	CERN Program Library	Z009

Author(s) : O. Hell
Library: KERNLIB
Submitter :
Submitted: 27.11. 1984
Language : Fortran
Revised:

Calendar Date Conversion

CALDAT converts any calendar date represention in a set of such representations to all other calendar date representations in the set; in addition a few extra bits of information are produced.

Structure:

SUBROUTINE subprogram
User Entry Names: CALDAT
Internal Entry Names: CDMON, CLEAP, CYDIY, CYEARY
External References: DATIME (Z007)

Usage:

```
    CALL CALDAT(IINDEX,CHREP,IBNREP,IERR)
```

IINDEX (INTEGER) Integer specifying which of the possible date representations is being given as the input representation. This input may either be as type CHARACTER within the CHREP string or as type INTEGER within the IBNREP array.

CHREP (CHARACTER*119) A character string containing, as substrings, the possible date representations. One such substring may be filled as the input representation, in which case it should be pointed to by IINDEX.

IBNREP (INTEGER) Array of length 8 containing various binary date representations. One such date representation may be filled as the input representation, in which case it should be pointed to by IINDEX.

IERR (INTEGER) Error flag. $\texttt{IERR} = 0$ success, $\texttt{IERR} \neq 0$ failure of the conversion.

The substrings of CHREP can be accessed directly, using CHARACTER substring operations. Alternatively all, or part, of the EQUIVALENCE statements below may be used:

```
    CHARACTER   DMY14*14,DMY11*11,DMY9*9,DMY10*10
    CHARACTER*8 DMY8A,DMY8B,YMD8,MDY8,YDM8
    CHARACTER*6 DMY6,       YMD6,MDY6,YDM6
    CHARACTER   YD5*5,W4*4,W2*2
    EQUIVALENCE
   *   (CHREP(  1: 14), DMY14), (CHREP( 15: 25), DMY11),
   *   (CHREP( 26: 34), DMY9 ), (CHREP( 35: 44), DMY10),
   *   (CHREP( 45: 52), DMY8A), (CHREP( 53: 60), DMY8B),
   *   (CHREP( 61: 66), DMY6 ), (CHREP( 67: 74), YMD8 ),
   *   (CHREP( 75: 80), YMD6 ), (CHREP( 81: 88), MDY8 ),
   *   (CHREP( 89: 94), MDY6 ), (CHREP( 95:102), YDM8 ),
   *   (CHREP(103:108), YDM6 ), (CHREP(109:113), YD5  ),
   *   (CHREP(114:117), W4   ), (CHREP(118:119), W2   )
```

Details of the substrings in argument CHREP and the corresponding IINDEX values:

	EXAMPLE	IINDEX		EXAMPLE	IINDEX
DMY14	16. APRIL 1982	1	YMD6	820416	9
DMY11	16 APR 1982	2	MDY8	04/16/82	10
DMY9	16 APR 82	3	MDY6	041682	11
DMY10	16. 4.1982	4	YDM8	82/16/04	12
DMY8A	16. 4.82	5	YDM6	821604	13
DMY8B	16/04/82	6	YD5	82106	14
DMY6	160482	7	W4	FRI.	
YMD8	82/04/16	8	W2	FR	

Details of the elements in argument IBNREP and the corresponding IINDEX values:

Element	Contents	Type	Example	IINDEX
1,2,3	d, m, y	binary	16, 4, 1982	101
4	day in the year	binary	106	102
5	00YYDDDC	packed dec	0082106C	103
6	Julian date	binary	723651	104
7	weekday, MO=0,...	binary	4	
8	week in the year	binary	15	
3,4	y, day in year	binary	1982, 106	105

Notes: Julian date = days since 1/1/1, without Gregory's pause. Week 1 of the year contains the 1st Thursday in the year (ISO).

Names of the months:

3 characters: 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN',
'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC'

5 characters: 'JAN. ', 'FEB. ', 'MARCH', 'APRIL', 'MAY ', 'JUNE ',
'JULY ', 'AUG. ', 'SEPT.', 'OCT. ', 'NOV. ', 'DEC. '

Names of the week days:

2 characters: 'MO', 'TU', 'WE', 'TH', 'FR', 'SA', 'SU'.

4 characters: 'MON.', 'TUE.', 'WED.', 'THUR', 'FRI.', 'SAT.', 'SUN.'.

Method:

Two arguments are used for passing the calendar dates: a character string and an array of full words. The various representations are numbered, and an input parameter ('input index') specifies the representation containing the input calendar date.
An extra output parameter receives a return code.

Special cases:

- Input index = 0 designates *today* which CALDAT will find.
- Input year yy rather than yyyy, designates *this century*.
- Input index or input data invalid:
  – output character string with all '*';
  – output numbers all X'81818181' = -2 122 219 135.

Restrictions:

CALDAT will give incorrect dates and weekdays for dates prior to the reformation of the Calendar by pope Gregory (16th century).

Error handling:

IERR	Meaning
0	everything fine
4	IINDEX < 0
8	upper bound for CHREP < IINDEX < lower bound for IBNREP
12	upper bound for IBNREP < IINDEX
16	ddd out of bounds
20	mm | dd out of bounds
24	yyyy out of bounds

Syntax errors:

IERR	in	IINDEX	IERR	in	IINDEX	IERR	in	IINDEX
1001	DMY14	1	1006	DMY8B	6	1011	MDY6	11
1002	DMY11	2	1007	DMY6	7	1012	YDM8	12
1003	DMY9	3	1008	YMD8	8	1013	YDM6	13
1004	DMY10	4	1009	YMD6	9	1014	YD5	14
1005	DMY8A	5	1010	MDY8	10	1103	Julian	103

Notes:

Element 5 of IBNREP is not a Fortran type. Nevertheless this calendar date format may show up in data from the 'real world'. Element 7 of IBNREP is especially well suited for arithmetical calculations with dates.

Examples:

```
C     Initialize substring CHREP(15:25)
      DMY11='16 APR 1982'
C     Define this substring to be the input format
      IINDEX=2
      CALL CALDAT(IINDEX,CHREP,IBNREP,IERR)
```

•

Author(s) : F. Carminati **Library:** KERNLIB, VAX/VMS only
Submitter : **Submitted:** 01.03.1989
Language : VAX Fortran **Revised:**

Usage Monitor for VAX/VMS

UMCOM is an usage monitor package for VAX/VMS systems. Usage log requests are performed either via Fortran calls or via DCL commands.

Structure:

Complete PROGRAM and SUBROUTINE subprograms
User Entry Names: UMCOM, UMLOG

Usage:

```
CALL UMCOM(CMD,MONITOR,TEXT)
```

CMD (CHARACTER) The first two letters of CMD are interpreted as a command to UMON. See the **Long Write-up** for possible commands.

MONITOR (CHARACTER) Name of the monitor to be affected by the command. If this name is longer than 8 characters, only the first 8 will be taken into account.

TEXT (CHARACTER) A character string containing information about the command given. If this string is longer than 80 characters, only the first 80 will be taken into account.

```
CALL UMLOG(MONITOR,TEXT)
```

MONITOR (CHARACTER) Name of the monitor to be affected by the command. If this name is longer than 8 characters, only the first 8 will be taken into account.

TEXT (CHARACTER) A character string containing the text to be logged. If this string is longer than 80 characters, only the first 80 will be taken into account.

See also the **Long Write-up**.
•

Author(s) : B. Lautrup, R. Matthews
Submitter : C. Letertre
Language : Fortran or Assembler or C

Library: KERNLIB
Submitted: 06.01.1971
Revised: 20.01.1986

Abnormal Termination of Fortran Programs

ABEND causes abnormal termination of a program. (On CDC all subsequent JCL control cards up to the next EXIT card will be ignored by the system).

Structure:

SUBROUTINE subprogram
User Entry Names: ABEND

Usage:

Not IBM:

```
CALL ABEND
```

causes abnormal termination of execution and prints the dayfile message ABEND. The output files are closed and INPUT is correctly positioned.

IBM:

```
CALL ABEND(KODEU)
```

The optional argument KODEU is used as the user completion code and must be an integer expression with a value in the range 0 – 4095. If the argument is omitted, or does not have a value in this range, a default value of 1 will be used.

•

Author(s) : R. Matthews, A. Cass
Submitter :
Language : Assembler

Library: KERNLIB, IBM only
Submitted: 01.02.1983
Revised: 19.07.1988

Intercept a Fortran Abend on IBM

ABUSER enables a user-supplied subroutine to receive control when the user's program abends. A call to ABUSER identifies the user-supplied subroutine which is to receive control. The identified subroutine will be called if the user's program abends and can perform pre-termination processing such as printing summaries or plotting histograms.

Structure:

SUBROUTINE subprogram
User Entry Names: ABUSER

Usage:

```
CALL ABUSER(NAME)
```

NAME Name of a user-supplied SUBROUTINE subprogram declared EXTERNAL in the calling program.

This subprogram receives control via a call of the form

```
CALL NAME(KODES,KODEU)
```

KODES A 4-byte integer containing, if available, the system completion code as hexadecimal number (use Z format for printing).

KODEU A 4-byte integer containing, if available, the user completion code as integer number (use I format for printing).

Restrictions:

This subprogram is compiler and system dependent.

MVS:

The Fortran 4 version relies on modifications to the IBM H-extended compiler library and is therefore not portable. The Fortran 77 version uses a standard interface into the FACOM compiler library.

CMS:

The subprogram is compiler independent but KODES and KODEU are not available and so are set to zero. Note that the routine uses storage in the CMS nucleus – the NUSERFWD field and also 8-bytes at NCCOPYR – which must not be overwritten. (No other CERN Library routine uses these locations.)

Notes:

ABUSER can be called at any time during normal processing, (i.e. before an abend occurs), to re-specify the name of the user-supplied subroutine. Alternatively, the effect of previous calls can be cancelled by CALL ABUSER(0). A call to ABUSER after an abend will have no effect.

A secondary abend which occurs while the user is processing the primary abend will cause program termination.

Under MVS the user-supplied subroutine will not receive control for the following completion codes:

`122`	– job cancelled with dump
`222`	– job cancelled
`322`	– cpu time exceeded
`522`	– wait time limit exceeded

Examples:

In the following example, `ABUSER` is called to identify a subroutine called `FATAL` as the subroutine which is to receive control when the user's program abends. If an abend occurs, subroutine `FATAL` will be called and will print the completion codes and then call `HISTDO` to plot histograms.

```
EXTERNAL FATAL
...
CALL ABUSER(FATAL)
...
END
SUBROUTINE FATAL(KODES,KODEU)
WRITE(6,'(1X,''PROGRAM ABENDING WITH CODES '',Z3,I5)') KODES,KODEU
CALL HISTDO
RETURN
END
```

•

Author(s) : C. Mekenkamp
Submitter : R. Veenhof
Language : Fortran, Vax Macro

Library: KERNLIB
Submitted: 10.03.1988
Revised:

Routines to Handle Control-C Interrupts on Vax

These routines allow you to write a program that, when interrupted with a *control*-C, resumes execution in a routine that you specify, which is higher up in the calling tree.

Structure:

Vax Macro and Vax Fortran routines
User Entry Names: ASTINT, ASTXIT, ASTDCC, ASTECC, ASTSCS, ASTECS
Internal Entry Names: ASTCCH

Usage:

VAXAST should be initialised at the beginning of the program by

```
CALL ASTINT
```

The routine to which control should be returned after a *control*-C has been typed, should have in its header

```
EXTERNAL ASTCCH
CALL LIB$ESTABLISH(ASTCCH)
```

When a *control*-C is typed on the terminal, ASTCCH is called. This routine is part of VAXAST, its main job is to unwind the stack of routine calls until the routine is found in which the LIB$ESTABLISH was issued. Your program then continues execution just after the call to the routine that was interrupted. You may have several routines with the header shown above. Only the last call to LIB$ESTABLISH has effect.

When you no longer wish to make use of the VAXAST routines:

```
CALL ASTXIT
```

You may not wish to have *control*-C trapped all the time, for instance when the program is waiting for input. To suspend trapping for a short while, do the following:

```
    CALL ASTDCC
    ...
    CALL ASTECC
```

Between ASTDCC and ASTECC a *control*-C typed on the terminal has the same effect as a *control*-Y, i.e. stopping the program and returning to DCL. Execution can, as with *control*-Y, be resumed at the point it was interrupted, via the CONTINUE command.

Not all programs survive the stack unwind ASTCCH performs. A classical example is the set of I/O routines in the Vax Fortran run time library (RTL). VAXAST replaces those routines by variants that are stack unwind proof but perform otherwise identical tasks. You will see 29 messages about multiply defined symbols when you LINK your program, you can safely ignore them.

If there is a part in your own program where the stack should not be unwound but during which you would like a *control*-`C` to be stored, do the following:

```
CALL ASTSCS
...
CALL ASTECS
```

A *control*-`C` typed between the `ASTSCS` and `ASTECS` calls remains 'dormant' and takes effect only at the `ASTECS` call.

Notes:

Carlo Mekenkamp, President Krugerstraat 42, NL-1975 EH IJmuiden.
•

QNEXTE	CERN Program Library	Z041

Author(s) : W. Jank, D. Lellouch, R. Matthews, E. Pagiola, J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 28.08.1984
Language : Assembler or C **Revised:**

Restart of Next Event

This interface routine allows the user to restart his program at the entry point `QNEXTE`, provided he has initiated it at this same entry point.

For first entry, `QNEXTE` remembers all necessary internal Fortran parameters, such as registers, trace-back, stack pointers, signal mask, whatever is needed on a given machine, and then calls a user-supplied routine `QNEXT`.

On any subsequent entry, `QNEXTE` resets all internal parameters so as to cancel all open `CALL`s below its own level, and then transfers again control to `QNEXT`. If in `QNEXT` a `RETURN` statement is reached this will lead back to the routine which did the first call to `QNEXTE`, usually the `MAIN` program.

Structure:

(Pseudo) `SUBROUTINE` subprogram
User Entry Names: `QNEXTE`
Internal Entry Names: `QNEXTD` (on Vax)
External References: User-supplied `SUBROUTINE` subprogram `QNEXT` (Z041)

Usage:

```
CALL QNEXTE
```

will transfer control to the routine `QNEXT` supplied by the user, via a `CALL QNEXT` (no parameter list).

Notes:

`QNEXT` is a user routine which cannot be loaded implicitly from a library. If to be used at all, it has to be loaded explicitly, either from a load file (such as produced by the compiler) or by some form of `INCLUDE` from a user library.

Because `QNEXTE` is referenced by some general packages, whose user may not want to supply a `QNEXT`, the reference from `QNEXTE` to `QNEXT` has been made 'weak' (to avoid the 'missing external' message from the loader) on the Vax (and probably also on some other machines in the future). In this case `QNEXTE` has a call to a Fortran dummy routine `QNEXTD` to print a message if it is reached without the user having supplied a routine `QNEXT`.

On most UNIX machines the loader is not able to start a module with missing externals; in this case, the user is obliged to provide a routine `QNEXT`, to stop the run, for example.

Examples:

Schema of Fortran `CALL` levels :

```
MAIN     CALL QNEXTE       ...      EVLOOP   CALL MATCH
QNEXTE   CALL QNEXT         .       MATCH    CALL TEST
QNEXT    CALL EVLOOP     ...        TEST     CALL QNEXTE
```

The last `CALL QNEXTE` abandons the current event.

•

Author(s) : J.Zoll, R.Brun et al. **Library:** KERNLIB
Submitter : J. Zoll **Submitted:** 27.04.1988
Language : Fortran or C or Assembler **Revised:** 20.02.1995

Calling a Subroutine by its Address

The purpose of this package is to provide a (limited) tool to connect what is called a user-routine with an arbitrary name to a `CALL` in a package, pre-existing on a library.

Because on most machines `JUMPXn` is implemented in Fortran or C, separate entries are needed for calling the user-routine with zero, one, two, ..., nine parameters.

Structure:

`SUBROUTINE` subprogram
User Entry Names: `JUMPAD`, `JUMPST`, `JUMPXn`, ($n = 0, 1, \ldots, 9$)
Internal Entry Names: `JUMPYn` (Z042) ($n = 0, 1, \ldots, 9$) (if not Assembler or C)

Usage:

Three steps are necessary:

1) Get the transfer address `IAD` of the routine (for example `TARGET`) to be called:

```
EXTERNAL TARGET
IAD=JUMPAD(TARGET)
```

2) Set the transfer address for the next transfer(s):

```
CALL JUMPST(IAD)
```

3) Execute a transfer, for a call with $n = 0, 1, \ldots, 9$ parameters:

```
    CALL JUMPX0
or  CALL JUMPX1(P1)
    ...
or  CALL JUMPX9(P1,P2,P3,P4,P5,P6,P7,P8,P9)
```

Restrictions:

Since on most machines `JUMPXn` is written in Fortran or C, the call to `JUMPXn` will be found in the trace-back of routine `TARGET`, and `RETURN` from `TARGET` will pass through `JUMPXn`. Hence, normally (i.e. unless recursion is handled by a particular machine), `TARGET` or any of its called routines may not again call `JUMPXn`.

•

Author(s) : F. Carminati, T. Lindelöf, R. Matthews, C. Vosicki, J. Zoll
Submitter :
Library: KERNLIB
Submitted: 01.12.1974
Revised: 01.06.1993
Language : Fortran or C or Assembler

Identify Job as Interactive

`INTRAC` allows an executing module to determine whether it is running interactively or not.

Structure:

`FUNCTION` subprogram
User Entry Names: `INTRAC`

Usage:

In any logical expression,

```
INTRAC()
```

has the value `.TRUE.` if the module is executing interactively and `.FALSE.` otherwise. Note that `INTRAC` must be declared `LOGICAL` in the calling routine.

Method:

On `UNIX` machines execution is interactive if 'standard input' (System `Unit 0`, i.e. Fortran `Unit 5` normally) is connected to a terminal. The same is true on VAX as from June 1993.

•

Author(s) : J. Shiers, C. Vosicki
Library: KERNLIB, VAX only
Submitter :
Submitted: 01.04.1994
Language : Fortran
Revised:

Identify Job as Running in Batch Mode

`IFBATCH` allows an executing module to determine whether it is running in batch mode or not.

Structure:

`FUNCTION` subprogram
User Entry Names: `IFBATCH`

Usage:

In any logical expression,

```
IFBATCH()
```

has the value `.TRUE.` if the module is executing in batch mode and `.FALSE.` otherwise. Note that `IFBATCH` must be declared `LOGICAL` in the calling routine.

•

Author(s) : R. Matthews, J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 15.07.1978
Language : Fortran **Revised:** 18.09.1991

Short List Reading and Writing

The 'long list' form `WRITE(LUN) (A(J),J=1,N)` is translated into slow object code by some compilers. Normally, these compilers handle the 'short list' form

```
DIMENSION A(N)
WRITE(LUN) A
```

correctly, compiling just one system request, rather than `N` requests.

Furthermore, some machines require the calling program to know the record size beforehand, if reading is done in Fortran. The problem can be solved by adding the record size as the first word of the record, thus for

writing: `WRITE(LUN) N,(B(J),J=1,N)`
reading: `READ (LUN) N,(B(J),J=1,N)`

This way of reading and writing is an extra convention; it is called 'variable length' in the descriptions below.

Sometimes it is convenient to prefix each record with some identifiers, always the same number of words, say `NA` words:

writing: `WRITE(LUN) N,(A(J),J=1,NA),(B(J),J=1,N)`
reading: `READ (LUN) N,(A(J),J=1,NA),(B(J),J=1,N)`

This mode is called 'split mode' in the descriptions below.

The routines of `XINOUT` provide 'short list' reading and writing for split mode, variable length mode and also for fixed length mode.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `XINB`, `XINBF`, `XINBS`, `XOUTB`, `XOUTBF`, `XOUTBS`
`COMMON` Block Names and Lengths: `/SLATE/ NR,DUMMY(39)`
Files Referenced: Parameter

Notes:

The routines `XINCF` and `XOUTCF` to handle formatted files are obsolete.

Usage:

Reading:

The vectors to be read are XAV and XV of length NA and NX; the read routines contain effectively

```
DIMENSION XV(NX) [,XAV(NA)]
```

Before calling, NX must be preset to the maximum number of words to be accepted into XV with, say, NX = NWMAX.

`CALL XINB(LUN,XV,NX)`	Read binary, variable length: `READ(LUN) NR,(XV(J),J=1,MIN(NR,NX))`
`CALL XINBF(LUN,XV,NX)`	Read binary, fixed length: `READ(LUN) XV`
`CALL XINBS(LUN,XAV,NA,XV,NX)`	Read binary, split mode: `READ(LUN) NR,XAV,(XV(J),J=1,MIN(NR,NX))`

On return NX contains:

NX > 0 : Read successful, number of words transmitted into XV.
$= 0$: End-of-file.
< 0 : Read error, its value contains the IOSTAT error code on most machines.

For XINB and XINBS the record length NR read from the file is stored into the first word of /SLATE/.

Writing:

The vectors to be written are AV and V of length NA and N; the write routines contain

```
DIMENSION V(N) [,AV(NA)]
```

`CALL XOUTB(LUN,V,N)`	Write binary, variable length: `WRITE(LUN) N,V`
`CALL XOUTBF(LUN,V,N)`	Write binary, fixed length: `WRITE(LUN) V`
`CALL XOUTBS(LUN,AV,NA,V,N)`	Write binary, split mode: `WRITE(LUN) N,AV,V`

•

IARGC	CERN Program Library	Z264

Author(s) : F. Carminati, M. Marquina **Library:** KERNLIB or Fortran Run-Time Library
Submitter : **Submitted:** 13.07.1988
Language : Fortran + C **Revised:** 15.03.1993

Returns Command Line Arguments

IARGC is used to return arguments that the user has given to an executable module on the command line.

Structure:

FUNCTION subprograms
User Entry Names: GETARG, IARGC

Usage:

```
NPAR = IARGC()
```

sets NPAR to the number of blank delimited arguments present after the program name on the command line. NPAR and IARGC are of type INTEGER.

```
CALL GETARG(IARG,GOTEXT)
```

IARG (INTEGER) Contains, on entry, the number of the argument to retrieve. Unchanged on exit.

GOTEXT (CHARACTER) Contains, on exit, the IARG-th argument.

Notes:

1. Arguments surrounded by double quotes (") are treated as single, e.g.

   ```
   "a variable here"
   ```

   is equivalent to one argument.
2. On VM/CMS, due to technical restrictions, at least one of the routines must be called before any I/O (typically a PRINT statement).
3. GETARG(0,GOTEXT) returns name of executing program (not VM).

Example:

```
      CHARACTER*100 STRING
C
C-- Retrieve the number of arguments given to this program
C
      NPAR=IARGC()
C-- and then get one by one, storing it in STRING
      DO 10 N = 1,NPAR
      CALL GETARG(N,STRING)
      PRINT *, STRING(1:LENOCC(STRING))
   10 CONTINUE
      END
```

•

Author(s) : see below
Library: KERNLIB
Submitter :
Submitted: 19.09.1991
Language : Fortran + C
Revised: 01.04.1994

Immediate Interface Routines to the C Library

Authors: F. Carminati, M. Marquina, A. Rademakers, J. Shiers, J. Zoll.

The routines of this package are Fortran callable routines which in turn call their corresponding C Library routines, after having taken care of the Fortran way of passing parameters.

The names of the interface routines are exactly the names of the C functions with the letter F added; the parameters are in one-to-one correspondence with the C functions; thus "`man <name>`" gives the exact details also for the interface routine.

Most Fortran systems on Unix machines are clever, they protect the Fortran user against name-clashes with the C library, for example a "`CALL RENAME (...)`" compiles as a reference to "`rename_`" (or to "`RENAME`" on the Cray).

If this is not strictly true, and/or if moreover the Fortran Run-time library does itself contain an interface routine "`rename`" then there might be trouble because it is not obvious which "`rename`" will be linked to the interface routine `RENAMEF`. The IBM 6000 machine has succeeded in creating this problem, it has both "`rename`" and "`rename_`" on the Fortran Run-time library. In this case one has to give an explicite `-lc` on the link statement to ensure that the C library is searched before the Fortran library (but after the Kernlib library).

Structure:

`SUBROUTINE` and `FUNCTION` subprograms

User Entry Names: `ACCESSF`, `CHDIRF`, `CTIMEF`, `EXITF`, `GETENVF`, `GETGIDF`, `GETPIDF`, `GETUIDF`, `GETWDF`, `GMTIMEF`, `KILLF`, `LSTATF`, `PERRORF`, `READLNF`, `RENAMEF`, `SETENVF`, `SLEEPF`, `STATF`, `SYSTEMF`, `UNLINKF`

`COMMON` Block Names and Lengths: `/SLATE/ ISLATE(40)`

Usage:

The types of all variables and functions follow from the Fortran default typing convention (unless typed explicitly), except that variables starting with the letters `CH` are of type `CHARACTER`.

The symbol `*` designates an output parameter.

For convenience, routines which return a `CHARACTER` string also return the occupied useful length of this string in `ISLATE(1)` of `/SLATE/`.

'access' — determine accessibility of file

```
LOGICAL ACCESSF
truth = ACCESSF(CHNAME,MODE)

  CHNAME  the path-name of the file
    MODE  a bit pattern specifying the type of access:

          bit 1 (1): execution permission
              2 (2): write permission
              3 (4): read permission
           all zero: existence
```

'chdir' — set current working directory

```
INTEGER CHDIRF
ISTAT = CHDIRF(CHNAME)

  CHNAME  the path-name of the new working directory
   ISTAT  function value returns zero if successful.
```

'ctime' — convert encoded time to ASCII

```
CHARACTER CHTIME*24
CALL CTIMEF(ITIME, CHTIME)

   ITIME  encoded time (as returned by STATF)
 CHTIME*  decoded time string of length 24
```

'exit' — terminate the process with a status code

```
CALL EXITF(IRC)
```

stops setting return status `IRC`. This should not be used for normal run termination. On the IBM VM this had to be implemented with a computed `GOTO`, hence if `IRC` $>$ `20` a `STOP 255` is executed.

On the Unix machines `IRC` will appear in the shell variable "status" which is reset after execution of each command, thus for more complicated logic the value of status has to be saved like (in the C shell):

```
set rc = $status
if   (rc != 0)  then
  if (rc == 1)  then
    echo '  not quite happy, but continue'
   else
    echo '  stop for trouble'
    exit
   endif
 endif
```

'getenv' — get the text of an environment variable

```
CHARACTER CHTEXT*(big enough)
CALL GETENVF(CHNAME, CHTEXT)

  CHNAME  the name of the environment variable,
 CHTEXT*  returns its value, with blank-fill
          ISLATE(1) occupied length, =0 if not found
```

'getgid' — get group identification

```
CALL GETGIDF(IDG)

  IDG  returns the real group ID of the current process.
```

'getpid' — get process identification

```
CALL GETPIDF(IDP)

  IDP  returns the process ID of the current process.
```

'getuid' - get user identification

```
CALL GETUIDF(IDU)

  IDU  returns the real user ID of the current process.
```

'getwd' — get the path-name of the working directory

```
CHARACTER CHTEXT*(big enough)
CALL GETWDF(CHTEXT)

 CHTEXT*  returns the path-name, with blank-fill
          ISLATE(1) occupied length, =0 if not found
```

'gmtime' — blow encoded time to time elements for Greenwich Mean Time

```
INTEGER ITMELS(9)
CALL GMTIMEF(ITIME, ITMELS)

   ITIME  encoded time (as returned by STATF)
 ITMELS*  decoded time elements:
          (1) sec, (2) min, (3) hour, (4) day, (5) month, (6) year,
          (7) weekday, (8) yearday, (9) isdst
```

'kill' — send a signal to a process

```
ISTAT = KILLF(IPID,ISIG)

    IPID  process ID
    ISIG  signal number
   ISTAT  function value returns zero if successful.
```

’perror’ — print message for the most recent C Library error

```
CALL PERRORF(CHTEXT)

  CHTEXT  the text to be printed before the error message
```

’readlink’ — read value of a symbolic link

```
INTEGER   READLNF
CHARACTER VAL*(big enough)

NCH = READLNF(CHNAME,VAL)

      CHNAME  path-name of the link
  VAL(1:NCH)  returns the value of the link
         NCH  useful length returned,
              = -1 if trouble, PERRORF may be called.
```

’rename’ — rename a file

```
INTEGER RENAMEF
ISTAT = RENAMEF(CHFROM,CHTO)

  CHFROM  old file name
    CHTO  new file name
   ISTAT  function value returns zero if successful.
```

’setenv’ - set environment variable

```
INTEGER SETENVF

ISTAT = SETENVF(CHNAME,CHVAL)

   CHNAME  name of the environment variable
    CHVAL  its value to be set
    ISTAT  function value returns zero if succesful.
```

On machines where the `setenv` function of system BSD is not available, `putenv` is used instead on a string constructed from `CHNAME` and `CHVAL` in allocated memory, hence one should avoid re-defining the same variable very many times.

’sleep’ — suspend execution

```
CALL SLEEPF(NSECS)

   NSECS  number of seconds to wait
```

'stat' — get file status

```
INTEGER INFO(12)
INTEGER STATF

ISTAT = STATF(CHNAME, INFO)

  CHNAME  path-name of the file
   INFO*  information returned
   ISTAT  function value returns zero if successful.
```

This routine returns the properties of a given file in a 12-word integer vector:

```
INFO(1) =  dev       device inode resides on
INFO(2) =  ino       this inode's number
INFO(3) =  mode      protection
INFO(4) =  nlink     number or hard links to the file
INFO(5) =  uid       user-id of owner
INFO(6) =  gid       group-id of owner
INFO(7) =  size      total size of file
INFO(8) =  atime     file last access time
INFO(9) =  mtime     file last modify time
INFO(10) = ctime     file last status change time
INFO(11) = blksize   optimal blocksize for file system i/o ops
INFO(12) = blocks    actual number of blocks allocated
```

On machines where `'blksize'` and `'blocks'` are not available (like Silicon Graphics) the words `INFO(11/12)` will always be zero.

'lstat' — get file status

`LSTATF` is like `STATF` except in the case where the named file is a symbolic link, in which case `LSTATF` returns information about the link, while `STATF` returns information about the file the link references.

For convenience `LSTATF` stores into `/SLATE/` some information about the nature of `CHNAME`:

```
ISLATE(1) = 0  if CHNAME is a regular file
ISLATE(2) = 0  if CHNAME is a symbolic link
ISLATE(3) = 0  if CHNAME is a directory
```

'system' — issue a shell command

```
INTEGER SYSTEMF
ISTAT = SYSTEMF(CHTEXT)

  CHTEXT  the command to be executed
   ISTAT  returns the exit status of the shell
```

'unlink' — remove directory entry

```
INTEGER UNLINKF
ISTAT = UNLINKF(CHNAME)

  CHNAME  the path-name of the file to be unlinked
   ISTAT  function value returns zero if successful.
```

Normally this deletes file `CHNAME`. If `CHNAME` is a soft link, the link is deleted, but not the file pointed to.

Notes:

The routine `SIGNALF`, which belongs to this family, will be described separately in the next paper

These routines have also been implemented on some machines which are not running Unix. The present state is as follows:
VAX system `VMS` has :

```
CHDIRF, EXITF, GETENVF, GETWDF, RENAMEF, SLEEPF, SYSTEMF
```

Presently `GETENVF` looks in the symbol table, except if the name of the environment variable is `"HOME"` for which it will return the value of the logical name `SYS$LOGIN`.
Some other routines are available through the C run-time library.
IBM 3090 system `VM/CMS` has :

```
CHDIRF, CTIMEF, EXITF, GETENVF, GETPIDF, GETWDF, GMTIMEF,
KILLF, PERRORF, RENAMEF, SLEEPF, STATF, SYSTEMF
```

•

Author(s) : F. Carminati, J. Zoll **Library:** KERNLIB, VAX only
Submitter : **Submitted:** 01.04.1994
Language : Fortran **Revised:**

Get the name of the executing module

This routine will figure out the path-name of the executing image. On the VAX this is done with a system call, on UNIX by scanning the search path until it finds the module whose name is in `argv[0]`.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `WHOAMI`
Common Blocks: `COMMON /SLATE/ ND,NE,NF,DUMMY(37)`

Usage:

```
    CALL WHOAMI(NAME)
```

On exit, `NAME` contains the full path-name of the module.
Status and various lengths are returned in `/SLATE/`:

```
    ND = 0  if the call failed,
       > 0  the number of characters in the path-name
```

On the VAX:

```
ND =  number of characters in the path-name with .EXE;n stripped
NE =  number of characters before the semicolon,
NF =  number of characters in the complete name.
```

For example:

```
      if NAME is  DISK:[CERN]WYLBUR.EXE;4
                  _:.=+=.: 1_:.=+=.: 2_:.=+=

      we will get  ND=17, NE=21, NF=23.
```

Note: At the moment this is available only on the VAX; the code exists for UNIX but is not yet in the library.
•

Author(s) : J. Zoll
Submitter :
Language : Fortran

Library: KERNLIB, VAX only
Submitted: 01.09.1990
Revised: 01.11.1994

Convert File-name to and from UNIX Syntax

These routines convert a file name from `UNIX` form to `VAX VMS` form, and vice versa. The correspondance is as follows:

```
VAX:    node::disk:[a.b.c]file.ext;cy
UNIX:   //node/disk/a/b/c/file.ext;cy

VAX:     [a.b.c]file.ext;cy   and   [.a.b.c]file.ext;cy
UNIX:   /(a/b/c/file.ext;cy            a/b/c/file.ext;cy
```

Forms like `../file.ext;cy` and `/dir/file.ext;cy` are also handled.
For back-compatibility `/=disk` is handled as `/disk`.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `FTOVAX`, `FFRVAX`
Common Blocks: `COMMON /SLATE/ ISTAT,DUMMY(39)`

Usage:

Convert to VAX form

```
CALL FTOVAX(CHNAME,NCH)

   *CHNAME*  file-name to be converted in situ
      *NCH*  significant length of the name
```

No conversion is done if the file-name does not contain a character `"/"` on input.

Convert to UNIX form

```
CALL FFRVAX(CHNAME,NCH)

   *CHNAME*  file-name to be converted in situ
      *NCH*  significant length of the name
```

No conversion is done if the file-name does already contain a character `"/"` on input.
This routine does some tidying up if necessary, thus for example the troublesome

`disk:[a][b.c]fn.ext` becomes the correct `/disk/a/b/c/fn.ext`

Both routines return `ISTAT=0` if no conversion was needed, `ISTAT=1` for successful conversion, and `ISTAT=-1` if a syntax error was detected.
Note that both routines update both the file-name and its useful length `NCH` in situ.

Examples:

```
tex/z267.tex                     [.tex]z267.tex
../wyl/kernfor.car               [-.wyl]kernfor.car
/(julia/kern/wyl/kernvax.car     [julia.kern.wyl]kernvax.car
/cern_root/pam/kernfor.car       cern_root:[pam]kernfor.car
```

•

Author(s) : C. Ciapetti, J. Zoll **Library:** KERNLIB, VAX only
Submitter : **Submitted:** 01.09.1983
Language : VAX Fortran **Revised:**

VAX Fortran Interface for Reading and Writing 'Foreign' Tapes

VAXTIO handles non-native tapes on the VAX; it is needed because VAX Fortran does not provide a U format.

If the tape to be handled is on logical unit 11, mounted on MTA0, with physical records of 3600 bytes maximum, for example, the following commands have to be given:

```
$ MOUNT MTA0:/FOREIGN/BLOCKSIZE=3600/RECORDSIZE=3600
$ ASSIGN MTA0: QIOUNIT11
```

Structure:

SUBROUTINE subprogram
User Entry Names: VAXTIO
Internal Entry Names: WAIT2S
Files Referenced: User defined parameter
COMMON Block Names and Lengths: /VAXTIO/ 240

Usage:

```
CALL VAXTIO(LUN,MODOP,IBUF,NDO,NDONE,NCODE,LUNMSG)
```

Input parameters:

LUN	Logical unit number (0 < LUN < 61).
MODOP	Operation mode, indicating the kind of operation to be performed; for details, see below.
IBUF	Data area for read and write.
NDO	Number of units to be done.
LUNMSG	Fortran logical unit number for printing diagnostic messages; if zero, printing is suppressed.

Output parameters:

NDONE	Number of units done; error if negative.
NCODE	QIO System status code.

The following operations are provided at present:

MODOP = −2 : Write EOF (3 tape marks are written and the tape is positioned after the first tape mark).

NDONE = 1	Successful.
NDONE = 0	End-of-tape.
NDONE = −7	Trouble.

MODOP = −1 : Write one record, tranfer NDO bytes from IBUF to tape.

NDONE > 1	Number of bytes written.
NDONE = 0	End-of-tape, but record written.
NDONE = −7	Trouble.

MODOP = 0 : Read one record, transfer at most NDO bytes from tape to IBUF, excess data are lost.

NDONE > 0	Number of bytes transferred.
NDONE = 0	EOF, end-of-tape.
NDONE = −1	Read error, record skipped.
NDONE = −7	Trouble.

MODOP = 1 : Assign a channel for logical unit (if not done explicitly, assignment occurs on first contact).

NDONE = 1	Successful.
NDONE = 0	Channel already assigned.
NDONE = −7	Trouble.

MODOP = 2 : Skip |NDO| records, forward if NDO > 0, reverse if NDO < 0. (|NDO| < 32768)

NDONE > 0	Number of records skipped.
NDONE < NDO	EOF seen, skipped, counted.
NDONE = −7	Trouble.

MODOP = 3 : Skip |NDO| files, forward if NDO > 0, reverse if NDO < 0.

NDONE > 0	Number of files skipped.
NDONE < NDO	End-of-tape seen.
NDONE = −7	Trouble.

MODOP = 4 : Rewind.

MODOP = 5 : Rewind and unload.

NDONE = 1	Successful.
NDONE = −7	Trouble.

MODOP = 6 : De-assign channel; this should be done if a logical unit is no longer needed.

NDONE = 1	Successful.
NDONE = −7	Trouble.

•

Author(s) : R. Matthews
Submitter :
Language : Fortran

Library: PACKLIB
Submitted: 25.08.1983
Revised: 07.02.1986

Random Access I/O Using Keywords

OBSOLETE

Please note that this routine has been obsoleted in CNL 219. Users are advised not to use it any longer and to replace it in older programs. No maintenance for it will take place and it will eventually disappear.

Suggested replacement: `ZEBRA` (Q100) or `HEPDB` (Q180)

A package of Fortran-callable subprograms for manipulating a random access file in which the records are of variable length and identified by a two-component name. This package may be used as the basis of a data base or bookkeeping system.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `KAADD`, `KAADDM`, `KACOPY`, `KADEL`, `KADELM`, `KAFREE`, `KAGET`, `KAGETM`, `KAHOLD`, `KALEN`, `KALIST`, `KALOC`, `KAMAKE`, `KAMSG`, `KAOPTN`, `KAPRE`, `KAPREM`, `KAPRIK`, `KAPUT`, `KAPUTM`, `KARLSE`, `KASEQ`, `KASEQM`, `KASTOP`

Usage:

See **Long Write-up**.

•

Author(s) : J. Zoll
Submitter :
Library: KERNLIB, UNIX and VMS
Submitted: 19.09.1991
Language : Fortran + C
Revised:

Handle Fixed-length Records on Unix Streams

The routines of this package are an interface to the C library functions open, read, write, lseek, close, to permit a Fortran program to handle an unstructured Unix file as a string of fixed-length binary records. Both sequential and direct-access READ / WRITE can be simulated.

These routines are simple little interface routines, there is no book-keeping done of the files which have been opened, the properties of the file have to be specified on each call, and the user is responsible for the consistency of all his calls for a particular file.

Processing has to be different for a disk file or for a tape file; therefore the medium must be indicated in the calls. Also, a user could take the source of these routines and modify them to add other branches for special processing.

New files are opened with the default permissions 644; one may set different permissions by calling `CFPERM` just before calling `CFOPEN`, which resets to the default after every call.

Three parameters are common to almost all routines :

```
LUNDES  is the file-descriptor of C to identify the file;
        with CFOPEN this is an output parameter,
        for all other routines it is an input parameter.

MEDIUM  = 0  for disk file, normal
          1      tape file, normal
          2      disk file, user coded I/O
          3      tape file, user coded I/O

NWREC   is the number of machine words for each one
        of the fixed-length records.
```

In the examples below it is assumed that for a given file these three parameters are available in something like `COMMON` storage.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `CFOPEN`, `CFGET`, `CFPUT`, `CFSIZE`, `CFTELL`, `CFSEEK`, `CFREW`, `CFCLOS`, `CFPERM`
Files Referenced: Parameter

Usage:

Note: the symbol * designates output parameters.

Open a file

```
CALL CFOPEN(LUNDES,MEDIUM,NWREC, CHMODE, NBUF, CHNAME, ISTAT)

    LUNDES* file-descriptor returned

    CHMODE  CHARACTER string selecting the IO mode :

            = 'r'   open for reading
              'r+'  open for read/write
              'w'   create or truncate for writing
              'w+'  open for write/read, create or truncate
              'a'   append
              'a+'  open for append/read

            [ add the letter "l" if labeled tape,
              action on this is not yet implemented ]

    NBUF    not used for the time being, always give zero
    CHNAME  name of the file, CHARACTER variable
    ISTAT*  status, =zero if success
```

For example, create a new file in the current directory :

```
MEDIUM = 0
NWREC  = 900
CALL CFOPEN(LUNDES,MEDIUM,NWREC, 'w', 0, 'run201.dat', ISTAT)
IF(ISTAT .NE. 0)          GO TO trouble
```

Read next record

```
CALL CFGET(LUNDES,MEDIUM,NWREC, NWTAK, MBUF, ISTAT)

   *NWTAK*  input:  number of words to be read
            output: number of words actually read

    MBUF*   vector to be read into
    ISTAT*  status, = zero if success,
            = -1   if end-of-file
```

To simulate direct-access reading one has to call `CFSEEK` first.
For example:

```
<< if the 7th record of the file is to be read:
   CALL CFSEEK(LUNDES,MEDIUM,NWREC, 6, ISTAT)
   IF(ISTAT .NE. 0)              GO TO trouble   >>

   NWTAK = NWREC
   CALL CFGET(LUNDES,MEDIUM,NWREC, NWTAK, MBUF, ISTAT)
   IF(ISTAT .EQ.-1)         GO TO eof
   IF(ISTAT .NE. 0)         GO TO trouble
```

Write next record

```
CALL CFPUT(LUNDES,MEDIUM,NWREC, MBUF, ISTAT)

    MBUF    vector to be written, NWREC words
    ISTAT*  status, =zero if success
```

Get the size of the file

```
CALL CFSIZE(LUNDES,MEDIUM,NWREC, NRECT, ISTAT)

    NRECT*  number of records on the file
    ISTAT*  status, =zero if success

Careful : this will position the file to the end.
```

Get the current file position

```
CALL CFTELL(LUNDES,MEDIUM,NWREC, NRECC, ISTAT)

    NRECC*  number of records before current
    ISTAT*  status, =zero if success
```

Set the current file position

```
CALL CFSEEK(LUNDES,MEDIUM,NWREC, NRECC, ISTAT)

    NRECC   number of records before current
    ISTAT*  status, =zero if success
```

For example :

```
CALL CFSEEK(..., 0, ISTAT)    position to start-of-file
CALL CFSEEK(..., 6, ISTAT)    position to 7th record
use CFSIZE to position to end-of-file
```

Rewind the file

```
CALL CFREW(LUNDES,MEDIUM)
```

Close the file

```
CALL CFCLOS(LUNDES,MEDIUM)
```

Set the permissions for the next open

```
CALL CFPERM(IPERM)

    IPERM   the permissions as a decimal integer,
            as returned by STATF (Z265) for example
```

For example (using `NCOCTI` of `M432`) :

```
CALL CFPERM(NCOCTI('660'))

      set read and write for owner and group only.
```

•

Author(s) : J. Zoll **Library:** KERNLIB, VAX and UNIX systems only
Submitter : **Submitted:** 31.10.1991
Language : Fortran + C **Revised:** 01.04.1994

Handle Unix Disk Files

The routines of this package are an interface to the C library functions open, read, write, lseek, close, to permit a Fortran program to handle an unstructured Unix file as a string of bytes. Both sequential and direct-access READ / WRITE can be done.

New files are opened with the default permissions 644; one may set different permissions by calling `CIPERM` just before calling `CIOPEN`, which resets to the default after every call.

One parameter is common to almost all routines : `LUNDES` is the file-descriptor of C to identify the file; with CIOPEN this is an output parameter, for all other routines it is an input parameter.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `CIOPEN`, `CIGET`, `CIGETW`, `CIPUT`, `CIPUTW`, `CISIZE`, `CITELL`, `CISEEK`, `CIREW`, `CICLOS`, `CIPERM`
Files Referenced: Parameter

Usage:

Note: the symbol * designates output parameters.

Open a file

```
    CALL CIOPEN(LUNDES, CHMODE, CHNAME, ISTAT)

        LUNDES* file-descriptor returned

        CHMODE  CHARACTER string selecting the IO mode :

                = 'r'   open for reading
                  'r+'  open for read/write
                  'w'   create or truncate for writing
                  'w+'  open for write/read, create or truncate
                  'a'   append
                  'a+'  open for append/read

        CHNAME  name of the file, of type CHARACTER
        ISTAT*  status, =zero if success
```

For example, create a new file in the current directory :

```
        CALL CIOPEN(LUNDES, 'w', 'concert.car', ISTAT)
        IF(ISTAT .NE. 0)          GO TO trouble
```

Read next string of bytes

```
CALL CIGET (LUNDES, CHBUF, NBDO, NBDONE, ISTAT)

    CHBUF*  text vector to be read into
     NBDO   maximum number of bytes to be read
   NBDONE*  number of bytes actually read
    ISTAT*  status, = zero if success,
            = -1   if end-of-file
```

Read next string of full words

```
CALL CIGETW(LUNDES, MBUF, NWDO, NWDONE, ISTAT)

     MBUF*  vector to be read into
     NWDO   maximum number of words to be read
   NWDONE*  number of words actually read
    ISTAT*  status, = zero if success,
            = -1   if end-of-file
```

A full word is normally 4 bytes; on the CRAY it is 8 bytes.
To simulate direct-access reading one has to call `CISEEK` first.
For example:

```
To read the next 2048 bytes:

 << starting at byte 8193 :
    CALL CISEEK(LUNDES, 8192, ISTAT)
    IF(ISTAT .NE. 0)              GO TO trouble   >>

    CALL CIGET(LUNDES, CHBUF, 2048, NBDONE, ISTAT)
    IF(ISTAT .EQ. -1)         GO TO eof
    IF(ISTAT .NE. 0)          GO TO trouble
```

Write next string of bytes

```
CALL CIPUT(LUNDES, CHBUF, NBDO, ISTAT)

    CHBUF   text vector to be written, NBDO bytes
    ISTAT*  status, =zero if success
```

Write next string of full words

```
CALL CIPUTW(LUNDES, MBUF, NWDO, ISTAT)

     MBUF   vector to be written, NWDO words
    ISTAT*  status, =zero if success
```

Get the size of the file

```
CALL CISIZE(LUNDES, NBYTT, ISTAT)

    NBYTT*  number of bytes on the file
    ISTAT*  status, =zero if success

Careful : this will position the file to the end.
```

Get the current file position

```
CALL CITELL(LUNDES, NBYTC, ISTAT)

    NBYTC*  number of bytes before current
    ISTAT*  status, =zero if success
```

Set the current file position

```
CALL CISEEK(LUNDES, NBYTC, ISTAT)

    NBYTC   number of bytes before current
    ISTAT*  status, =zero if success
```

For example :

```
CALL CISEEK(LUNDES, 0, ISTAT)    position to start-of-file
CALL CISEEK(LUNDES, 8, ISTAT)    position to 9th byte
use CISIZE to position to end-of-file
```

Rewind the file

```
CALL CIREW(LUNDES)
```

Close the file

```
CALL CICLOS(LUNDES)
```

Set the permissions for the next open

```
CALL CIPERM(IPERM)

    IPERM   the permissions as a decimal integer,
            as returned by STATF (Z265) for example
```

For example (using `NCOCTI` of `M432`) :

```
CALL CIPERM(NCOCTI('664'))

      set read for everybody, and write for owner and group.
```

Note: formally the buffer for reading and writing should be of type `CHARACTER` for `CIGET` and `CIPUT`, and of type `INTEGER` for `CIGETW` and `CIPUTW`. On most machines there is no difference, but on the VAX this must be observed, because the parameter passing mechanisme differs crucially for the two cases. Also, on the CRAY there would be problems if one were using `CIGETW` to read into a Character address other than a word boundary.

•

Author(s) : J. Zoll **Library:** KERNLIB
Submitter : **Submitted:** 01.11.1994
Language : Fortran **Revised:**

Terminal Dialogue Routines

These routines prompt the user on-line to the executing program for input from the terminal, and read it. The prompt is written to standard output by calling `TMPRO`, the input is read from standard input with `TMREAD`. Whether or not standard input is in fact a terminal can be detected with `INTRAC` (Z044); if it is not the call to `TMPRO` should be by-passed.

Structure:

`SUBROUTINE` subprograms
User Entry Names: `TMINIT`, `TMPRO`, `TMREAD`
Files references: standard input, standard output

Usage:

Initialize the dialogue

On some machines it is necessary to switch off buffered mode on standard output, this is done by calling once, and before the first call to `TMPRO`:

```
    CALL TMINIT (IFINIT)

        IFINIT*  is reset to non-zero by TMINIT
```

Put the prompt to standard output

```
    CALL TMPRO (TEXT)

          TEXT   is the character string to be written
```

Read next line from standard input

```
    CALL TMREAD (MAXCH, CHLINE, NCH, ISTAT)

         MAXCH   maximum number of char. to be stored into LINE
          LINE*  text read, of type CHARACTER
           NCH*  number of characters read into LINE
         ISTAT*  status returned:
                 = 0  success
                 < 0  end-of-file seen
                 > 0  read error
```

•

Index