

SUPPLEMENTAL LIBRARY PROCEDURE DESCRIPTIONS

This document contains a description of two small libraries for use in conjunction with *Fortran 90/95 for Scientists and Engineers*, by Stephen J. Chapman. The two libraries are **LAPACK_S**, which is a small subset of the free LAPACK library, and **BOOKLIB**, which is a library prepared especially for use with this book.

The **LAPACK_S** library contains four subroutines for solving systems simultaneous equations. They were extracted from the FORTRAN 77 version of the full LAPACK library and converted to Fortran 90 format. The full LAPACK library is free, and may be downloaded from the Internet at <http://www.netlib.org>. The **BOOKLIB** contains procedures written especially for use with the exercises in *Fortran 90/95 for Scientists and Engineers*.

Both of these libraries are distributed on a disk along with the Instructors Manual that accompanies the book, and they are available for download at the book's Web site <http://www.mhhe.com/engineering/chapman/>. Your instructor will probably have compiled the two libraries on the computer that you are using in this course.

The procedures in **LAPACK_S** are indexed by name and by function in the table shown below.

Table 1: Procedures included in the LAPACK_S library		
Name	Function	Page
SGESV	Solve a system of simultaneous equations. (single precision)	33
DGESV	Solve a system of simultaneous equations. (double precision)	33
CGESV	Solve a system of simultaneous equations. (single prec. complex)	33
ZGESV	Solve a system of simultaneous equations. (double prec. complex)	33

The procedures in **BOOKLIB** are indexed by name and by function in the table shown below.

Table 2: Procedures included in the BOOKLIB library		
Name	Function	Page
cross_prod	Calculate the cross product of two 3-element vectors.	3
deriv	Calculate derivative of a user-supplied function.	4
dft	Calculate Discrete Fourier Transform from its definition.	6
fft	Calculate Fast Discrete Fourier Transform.	8
heapsort	Sort an array into ascending order using the heapsort algorithm.	10
heapsort_2	Sort an array into ascending order while carrying along a second array, using the heapsort algorithm.	11
histogram	Print a histogram of an input data set on a line printer.	13
idft	Calculate inverse Discrete Fourier Transform from its definition.	15
ifft	Calculate inverse Fast Discrete Fourier Transform.	17
integ	Integrate a user-supplied function $f(x)$ between points x_1 and x_2 using rectangles of width Δx .	19

integ_d	Integrate a discrete function specified by a series of (x,y) values between points x_1 and x_2 , where x_1 and x_2 both lie within the range of input values in the (x,y) pairs.	20
interp	Linearly interpolate the value y_o at position x_o , given a set of (x,y) measurements organized in increasing order of x .	22
lcase	Shift a character string to lower case.	23
lsq_fit	Perform a least-squares fit of an input data set to the n th order polynomial.	24
mat_inv	Invert an $N \times N$ matrix using Gaussian elimination and the maximum pivot technique.	25
nxtmul	Calculate the next power of a base above a specific number.	27
plot	Print a line printer plot a function.	28
plotxy	Print a line printer cross-plot a set of (x,y) data points.	30
random_u	Uniform distribution random number generator (function).	32
random_n	Normal distribution random number generator (function).	32
random_r	Rayleigh distribution random number generator (function).	32
simul	Solve a system of simultaneous equations.	35
sinc	Calculate the sinc function: $\text{sinc}(x) = \sin(x) / x$.	36
spline_fit	Calculate the set of cubic spline polynomials that fit an input data set.	37
spline_int	Interpolate a point using the set of cubic spline polynomials generated by subroutine spline_fit .	39
ssort	Sort an array into ascending order using the selection sort algorithm.	41
statistics	Calculate the average, standard deviation, and mean of a data set.	42
ucase	Shift a character string to upper case.	23

Many of the procedures in these libraries are generic procedures, which work with multiple types of input data. The types of data supported by each procedure is shown in parentheses after the procedure name. The **keywords** associated with dummy procedure arguments are shown in capital letters in the calling sequences, and the keywords for optional arguments are shown in parentheses.

In the following procedure descriptions, data types are given by the following abbreviations:

Abbreviation	Type
R	Single Prec. Real
D	Double Prec. Real
C	Single Prec. Complex
D	Double Prec. Complex
I	Integer
L	Logical
Char	Character

cross_prod (Single/Double Precision Real)

Purpose: To calculate the cross product of two three-element real vectors.

Usage: `USE booklib`
`vector = cross_prod (VA, VB)`

Arguments:

Name	Type	Dim	I/O	Description
VA	R/D	3	I	First vector.
VB	R/D	3	I	Second vector.
cross_prod	R/D	3	O	Cross product of VA and VB.

Algorithm:

This function calculates the cross product of two vectors according to the equations:

$$\begin{aligned}\text{cross_prod}(1) &= v1(2) * v2(3) - v2(2) * v1(3) \\ \text{cross_prod}(2) &= v1(3) * v2(1) - v2(3) * v1(1) \\ \text{cross_prod}(3) &= v1(1) * v2(2) - v2(1) * v1(2)\end{aligned}$$

Example:

This example calculates cross product of two vectors **va** = [1. 0. 1.] and **vb** = [-1 1 -1].

```
USE booklib
IMPLICIT NONE
REAL, DIMENSION(3) :: va = (/ 1., 0., 1. /)
REAL, DIMENSION(3) :: vb = (/ -1., 1., -1. /)
WRITE (*, '(A,3(2X,F10.4))') ' The cross product is ', &
    cross_prod (va, vb)
END PROGRAM
```

Result:

The cross product is -1.0000 .0000 1.0000

deriv (Single/Double Precision Real)

Purpose: To calculate the derivative of a function $f(x)$ at point x_0 using step size Δx . If $\Delta x = 0.0$, then take the derivative with as much accuracy as possible. This subroutine expects the function $f(x)$ to be passed as a calling argument.

Usage: `USE booklib`
`CALL deriv (F, X0, DX, DFDX, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
F	R/D Func.		I	Function to take derivative of
X0	R/D		I	Point at which to take derivative
DX	R/D		I/O	Step size to use when taking derivative (≥ 0.0). If DX = 0.0, then the subroutine calculates an optimal step size, and returns that step size in this variable.
DFDX	R/D		O	The derivative $df(x)/dx$
ERROR	I		O	Error flag: 0 = No error 1 = DX < 0.

Algorithm:

This subroutine calculates the derivative using the central difference method:

$$\frac{d}{dx} f(x) \approx \frac{f(x + \Delta x / 2) - f(x - \Delta x / 2)}{\Delta x}$$

The subroutine uses the user-specified Δx if it is > 0 . Otherwise, it tries values of $\Delta x = 0.1, 0.01$, *etc.* until roundoff errors start to dominate in the solution. If Δx is zero, then the actual Δx used to calculate the derivative is returned in variable **DX**.

Example:

This example calculates derivative of function $\sin(x)$ at $x_0 = 1.0$ using the default step size.

```
USE booklib
INTRINSIC SIN
INTEGER :: error
REAL :: dfdx, dx = 0.
CALL deriv ( SIN, 1.0, dx, dfdx, error )
WRITE (*,1000) dfdx
1000 FORMAT (' The derivative of SIN(X) at X0 = 1.0 is: ', F10.6)
```

Supplemental Library Descriptions to accompany
Fortran 90/95 for Scientists and Engineers, by Stephen J. Chapman

```
WRITE (*,1010) COS(1.0)
1010 FORMAT (' The theoretical value is:           ', F10.6)
WRITE (*,1020) dx
1020 FORMAT (' The step size used is:             ', F10.6)
```

Result:

```
The derivative of SIN(X) at X0 = 1.0 is:      .540316
The theoretical value is:                      .540302
The step size used is:                        .001000
```

dft (Single / Double Precision Complex)

Purpose: To perform a discrete Fourier transform on complex array **ARRAY_IN** with the result returned in array **ARRAY_OUT**. This subroutine calculates the DFT directly from its definition.

Usage: `USE booklib`
`CALL dft (ARRAY_IN, ARRAY_OUT, N)`

Arguments:

Name	Type	Dim	I/O	Description
ARRAY_IN	C/Z	N	I	Time series to analyze
ARRAY_OUT	Same as above	N	O	Frequency spectrum of data set
N	I		I	Number of values in array

Algorithm:

This subroutine calculates the DFT directly from its definition. It is very slow compared to subroutine **fft** for large arrays of data. Unlike subroutine **fft**, it does not require that the number of input points be a power of 2. If there are N input values, t_k is the k th value in the input time sequence, and F_n is the n th component in the output frequency spectrum, then

$$F_n = \sum_{k=0}^{N-1} t_k e^{-2\pi i k n / N}$$

WARNING: This subroutine is very slow for large array sizes. It is included in the library to support homework problems only. For real work, use subroutine **fft** instead.

Example:

This example calculates the frequency spectrum of a 16-point complex data set consisting of all (1.0,0.0). Because this data set is constant, the peak of the frequency spectrum of the data should be 0 Hz (DC).

```
USE booklib
COMPLEX, DIMENSION(16) :: array_in(16) = (/ ((1., 0.), i=1,16) /)
COMPLEX, DIMENSION(16) :: array_out(16)
CALL dft ( array_in, array_out, 16 )
WRITE (*,1000) (i,array_out(i), i=1,16)
1000 FORMAT (' array_out(',I2,') = (',F10.4,',',F10.4,')')
```

Result:

```
array_out( 1) = ( 16.0000, 0.0000)
array_out( 2) = ( 0.0000, 0.0000)
array_out( 3) = ( 0.0000, 0.0000)
array_out( 4) = ( 0.0000, 0.0000)
array_out( 5) = ( 0.0000, 0.0000)
array_out( 6) = ( 0.0000, 0.0000)
...
array_out(15) = ( 0.0000, 0.0000)
array_out(16) = ( 0.0000, 0.0000)
```

fft (Single / Double Precision Complex)

Purpose: To perform a fast discrete Fourier transform on complex array **ARRAY_IN**. The resulting frequency spectrum is returned in array **ARRAY_OUT**. The size of the data set in array **ARRAY_IN** must be a power of 2 (32, 64, 128, etc.).

Usage: `USE booklib`
`CALL fft (ARRAY_IN, ARRAY_OUT, N, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
ARRAY_IN	C/Z	N	I	Time series to analyze
ARRAY_OUT	Same as above	N	O	Frequency spectrum of data set
N	I		I	Number of data points (must be a power of 2)
ERROR	I		O	Error flag: 0 = No error 1 = N not a power of 2

Algorithm:

This subroutine employs a Radix 2, in-place, decimation in frequency algorithm. To avoid destroying the input data set, it copies the contents of **ARRAY_IN** to **ARRAY_OUT** before performing the FFT. For details, see Oppenheim and Shaffer, *Digital Signal Processing*, Prentice-Hall, 1975. If there are N input values, t_k is the k th value in the input time sequence, and F_n is the n th component in the output frequency spectrum, then

$$F_n = \sum_{k=0}^{N-1} t_k e^{-2\pi i k n / N}$$

Example:

This example calculates the frequency spectrum of a 16-point complex data set consisting of all (1.0,0.0). Because this data set is constant, the peak of the frequency spectrum of the data should be 0 Hz (DC).

```
USE booklib
INTEGER :: error
COMPLEX, DIMENSION(16) :: array_in(16) = (/ ((1., 0.), i=1, 16) /)
COMPLEX, DIMENSION(16) :: array_out(16)
CALL fft ( array_in, array_out, 16, error )
WRITE (*, 1000) (i, array_out(i), i=1, 16)
1000 FORMAT (' array_out(' , I2, ') = (', F10.4, ', ', F10.4, ')')
```


Result:

```
array_out( 1) = ( 16.0000, 0.0000)
array_out( 2) = ( 0.0000, 0.0000)
array_out( 3) = ( 0.0000, 0.0000)
array_out( 4) = ( 0.0000, 0.0000)
array_out( 5) = ( 0.0000, 0.0000)
array_out( 6) = ( 0.0000, 0.0000)
...
array_out(15) = ( 0.0000, 0.0000)
array_out(16) = ( 0.0000, 0.0000)
```

heapsort (Integer/Single Prec. Real/Double Prec. Real/Character)

Purpose: To sort an array into ascending order using the heapsort algorithm.

Usage: `USE booklib`
`CALL heapsort (ARRAY, N, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
ARRAY	I/R/D/C	N	I/O	Array to sort
N	I		I	Number of elements in array
ERROR	I		0	Error flag: 0 = No error 1 = N <= 0

Algorithm:

These subroutines sort arrays into ascending order using the heapsort algorithm. This algorithm is much more efficient than the selection sort algorithm. It should be used instead of the selection sort whenever large arrays are to be sorted.

Example:

This example declares an integer array `iarray` and initializes it with 15 values. It uses subroutine `heapsort` to sort the array into ascending order.

```
USE booklib
INTEGER, DIMENSION(15) :: iarray = &
    (/ -100, 0, -20, 1, -20, &
       90, -123, 602, 5, 17, &
       91, -4, 0, 37, -11 /)
INTEGER :: n = 15, error
WRITE (*,*) ' iarray before sorting: '
WRITE (*,'(3X,5I6)') iarray
CALL heapsort ( iarray, n, error )
WRITE (*,*) ' iarray after sorting: '
WRITE (*,'(3X,5I6)') iarray
```

Result:

```
iarray before sorting:
-100    0   -20    1   -20
  90 -123  602    5    17
  91   -4    0   37   -11

iarray after sorting:
-123 -100  -20  -20  -11
  -4    0    0    1    5
  17   37   90   91  602
```

heapsort_2

(Integer/Single Prec. Real/Double Prec. Real/Character)

Purpose: To sort an array into ascending order while carrying along a second array, using the heapsort algorithm.

Usage: **USE booklib**
 CALL heapsort_2 (ARRAY, ARRAY_2, N, ERROR)

Arguments:

Name	Type	Dim	I/O	Description
ARRAY	I/R/D/C	N	I/O	Array to sort
ARRAY_2	Same as above	N	I/O	Array to carry along
N	I		I	Number of elements in array
ERROR	I		0	Error flag: 0 = No error 1 = N <= 0

Algorithm:

This subroutine sorts arrays into ascending order using the heapsort algorithm, and carries along a second array. For example, if **ARRAY(i)** is moved to the top of array **ARRAY**, then **ARRAY_2(i)** is moved to the top of array **ARRAY_2**. This subroutine permits a user to sort the contents of one array according to the values in another one.

Example:

This example declares two integer arrays **iarray** and **ipoint**. It initializes **iarray** with 15 arbitrary values, and **ipoint** with the numbers 1 through 15. After sorting the arrays with subroutine **heapsort_2**, the values in **ipoint** are pointers to the original locations of the values in **iarray**.

```
USE booklib
INTEGER, DIMENSION(15) :: iarray = &
    (/ -100, 0, -20, 1, -20, &
       90, -123, 602, 5, 17, &
       91, -4, 0, 37, -11 /)
INTEGER, DIMENSION(15) :: ipoint = &
    (/ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 /)
INTEGER :: n = 15, error
CALL heapsort_2 ( iarray, ipoint, n, error )
WRITE (*,*) ' Sorted array and original locations: '
WRITE (*,1000) (iarray(i), ipoint(i), I = 1, 15)
1000 FORMAT (3X,2I6,8X,2I6,8X,2I6)
```

Result:

Sorted array and original locations:

-123	7	-100	1	-20	3
-20	5	-11	15	-4	12
0	13	0	2	1	4
5	9	17	10	37	14
90	6	91	11	602	8

histogram (Single Precision Real)

Purpose: Subroutine to print a histogram of an input data set on a line printer.

Usage: `USE booklib`
`CALL histogram (DATA1, NPTS, LU, ERROR, NBINS, MINBIN, MAXBIN)`

Arguments:

Name	Type	Dim	I/O	Description
DATA1	R	NPTS	I	Data set to analyze
NPTS	I		I	Number of points in input data set
LU	I		I	I/o unit to print histogram on.
ERROR	I		0	Error flag: 0 = No error 1 = Too few bins requested (<1) 2 = MAXBIN = MINBIN. These values must differ.
NBINS	I		I	Number of bins to accumulate statistics in. If present, NBINS must be greater than 1. If absent, it defaults to 20.
MINBIN	R		I	Value of the smallest bin in the histogram. The default value is the smallest number in the data set.
MAXBIN	R		I	Value of the largest bin in the histogram. The default value is the largest number in the data set.

Algorithm:

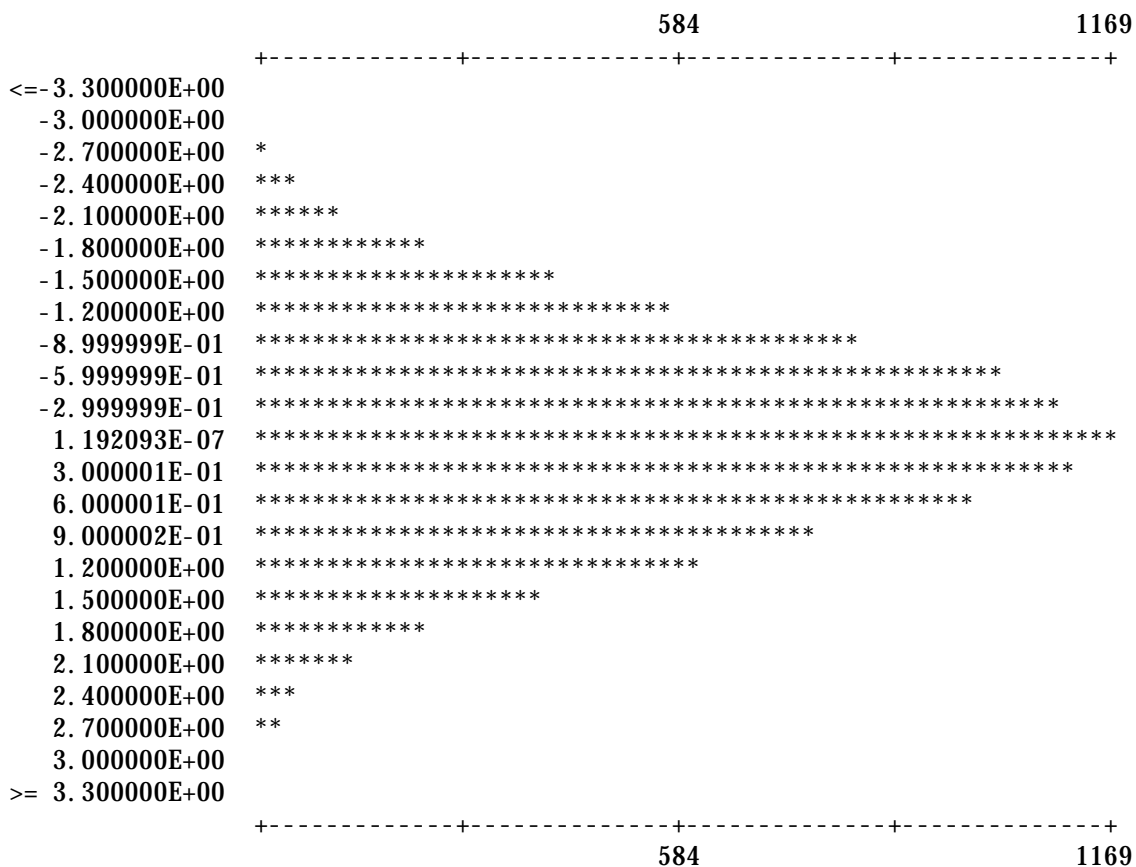
This subroutine calculates the range of values associated with each bin, and then accumulates statistics on the input data set. It then prints the resulting histogram on the device specified by i/o unit LU.

Example:

This example uses function `g_random1` to generate 10000 random numbers with a normal distribution, and then plots a histogram of the data using subroutine `histogram`. Note that the values of NBINS, MINBIN, and MAXBIN are defaulted.

```
USE booklib
INTEGER :: error
REAL, DIMENSION(10000) :: data1
DO i = 1, 10000
    data1(i) = g_random1()
END DO
CALL histogram(data1, 10000, 6, error )
```

Result:



Number of samples = 10000

idft (Single / Double Precision Complex)

Purpose: To perform an inverse discrete Fourier transform on complex array **ARRAY_IN** with the result returned in array **ARRAY_OUT**. This subroutine calculates the inverse DFT directly from its definition.

Usage: `USE booklib`
`CALL idft (ARRAY_IN, ARRAY_OUT, N)`

Arguments:

Name	Type	Dim	I/O	Description
ARRAY_IN	C/Z	N	I	Frequency spectrum of data set
ARRAY_OUT	Same as above	N	O	Resulting time series
N	I		I	Number of values in array

Algorithm:

This subroutine calculates the inverse DFT directly from its definition. It is very slow compared to subroutine **ifft** for large arrays of data. Unlike subroutine **ifft**, it does not require that the number of input points be a power of 2. If there are N input values, t_k is the k th value in the input time sequence, and F_n is the n th component in the output frequency spectrum, then

$$F_n = \frac{1}{N} \sum_{k=0}^{N-1} t_k e^{2\pi i k n / N}$$

WARNING: This subroutine is very slow for large array sizes. It is included in the library to support homework problems only. For real work, use subroutine **ifft** instead.

Example:

This example shows that **idft** is the inverse of **dft**. Here, we take both the DFT and the inverse DFT of a data set, and wind up with the data we started with.

```
USE booklib
COMPLEX, DIMENSION(16) :: c_in, c_inter, c_out
c_in = (/ (0.,0.), (1.,0.), (2.,0.), (1.,0.), &
          (0.,0.), (-1.,0.), (-2.,0.), (-1.,0.), &
          (0.,0.), (1.,0.), (2.,0.), (1.,0.), &
          (0.,0.), (-1.,0.), (-2.,0.), (-1.,0.) /)
CALL dft ( c_in, c_inter, 16 )
CALL idft ( c_inter, c_out, 16 )
WRITE (*,1000) (c_out(i), i=1, 16)
1000 FORMAT (' c_out = ',/,4(' (',F4.1,',',F4.1,')'))
```

Result:

```
c_out =  
( 0.0, 0.0) ( 1.0, 0.0) ( 2.0, 0.0) ( 1.0, 0.0)  
( 0.0, 0.0) (-1.0, 0.0) (-2.0, 0.0) (-1.0, 0.0)  
( 0.0, 0.0) ( 1.0, 0.0) ( 2.0, 0.0) ( 1.0, 0.0)  
( 0.0, 0.0) (-1.0, 0.0) (-2.0, 0.0) (-1.0, 0.0)
```


ifft (Single / Double Precision Complex)

Purpose: To perform a fast inverse discrete Fourier transform on the frequency spectrum in complex array **ARRAY_IN**. The resulting time series is returned in array **ARRAY_OUT**. The size of the data set in array **ARRAY_IN** must be a power of 2 (32, 64, 128, etc.).

Usage: `USE booklib`
`CALL ifft (ARRAY_IN, ARRAY_OUT, N, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
ARRAY_IN	C/Z	N	I	Frequency spectrum of data set
ARRAY_OUT	Same as above	N	O	Resulting time series
N	I		I	Number of data points (must be a power of 2)
ERROR	I		O	Error flag: 0 = No error 1 = N not a power of 2

Algorithm:

This subroutine employs a Radix 2, in-place, decimation in frequency algorithm. To avoid destroying the input data set, it copies the contents of **ARRAY_IN** to **ARRAY_OUT** before performing the inverse FFT. For details, see Oppenheim and Shaffer, *Digital Signal Processing*, Prentice-Hall, 1975. If there are N input values, t_k is the k th value in the input time sequence, and F_n is the n th component in the output frequency spectrum, then

$$t_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{2\pi i k n / N}$$

Example:

This example shows that **ifft** is the inverse of **fft**. Here, we take both the FFT and the inverse FFT of a data set, and wind up with the data we started with.

```
USE booklib
INTEGER :: error
COMPLEX, DIMENSION(16) :: c_in, c_inter, c_out
c_in = (/ (0.,0.), (1.,0.), (2.,0.), (1.,0.), &
          (0.,0.), (-1.,0.), (-2.,0.), (-1.,0.), &
          (0.,0.), (1.,0.), (2.,0.), (1.,0.), &
          (0.,0.), (-1.,0.), (-2.,0.), (-1.,0.) /)
CALL fft ( c_in, c_inter, 16, error )
CALL ifft ( c_inter, c_out, 16, error )
WRITE (*,1000) (c_out(i), i=1, 16)
1000 FORMAT (' c_out = ',/,4(' (',F4.1,',',F4.1,')'))
```

Result:

```
c_out =  
( 0.0, 0.0) ( 1.0, 0.0) ( 2.0, 0.0) ( 1.0, 0.0)  
( 0.0, 0.0) (-1.0, 0.0) (-2.0, 0.0) (-1.0, 0.0)  
( 0.0, 0.0) ( 1.0, 0.0) ( 2.0, 0.0) ( 1.0, 0.0)  
( 0.0, 0.0) (-1.0, 0.0) (-2.0, 0.0) (-1.0, 0.0)
```

integ (Single / Double Precision Real)

Purpose: To integrate a function $f(x)$ between points x_1 and x_2 using rectangles of width Δx . This subroutine expects the function $f(x)$ to be passed as a calling argument.

Usage: `USE booklib`
`CALL integ (F, X1, X2, DX, AREA, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
F	R/D FUN		I	Name of function to integrate
X1	Same as above		I	Starting point for integration
X2	Same		I	Ending point for integration
DX	Same		I	Step size for integration
AREA	Same		0	Integrated value
ERROR	I		0	Error flag: 0 = No error 1 = $X1 > X2$

Algorithm:

This subroutine calculates the area under the curve $f(x)$ by dividing the distance between x_1 and x_2 into steps of size Δx and calculating the area under the curve for each step. The area calculation is done by approximating the area under the curve as a rectangle whose height is the value of $f(x)$ at the center of the step interval.

Example:

This example uses **integ** to integrate the intrinsic function $\sin(x)$ from 0 to π . (The theoretical area of this integral is 2.0.)

```
USE booklib
INTRINSIC SIN
INTEGER :: error
REAL :: x1 = 0., x2 = 3.141592, dx = 0.05, area
CALL integ ( SIN, x1, x2, dx, area, error )
WRITE (*,1000) area
1000 FORMAT ( ' The area under curve SIN(x) from 0. to PI is: ', F10.6)
```

Result:

The area under curve SIN(x) from 0. to PI is: 2.000208

integ_d (Single / Double Precision Real)

Purpose: To integrate a discrete function specified by a series of (x,y) values between points x_1 and x_2 , where x_1 and x_2 both lie within the range of input values in the (x,y) pairs. The (x,y) pairs must be passed to this subroutine in increasing order of x .

Usage: `USE booklib`
`CALL integ_d (X, Y, NPTS, X1, X2, AREA, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
X	R/D	NPTS	I	Values of independent variable x
Y	Same as X	NPTS	I	Values of dependent variable y
NPTS	I		I	Number of (x,y) values passed to the subroutine
X1	Same as X		I	Starting point for integration
X2	Same as X		I	Ending point for integration
AREA	Same		0	Integrated value
ERROR	I		0	Error flag: 0 = No error 1 = $X1 > X2$ 2 = $X1 < X(1)$ 3 = $X1 > X(NPTS)$

Algorithm:

This subroutine calculates the area under a curve specified by a series of discrete (x,y) points by calculating the area under the trapezoids formed by adjacent pairs of (x,y) values.

Example:

This example uses **integ_d** to integrate the intrinsic function $\sin(x)$ from 0 to π . Note that $\sin(x)$ is specified by (x,y) values in a pair of arrays. (The theoretical area of this integral is 2.0.)

```
USE booklib
INTEGER :: i, npts = 101, error
REAL, DIMENSION(101) :: x, y
REAL :: x1 = 0., x2 = 3.141592, dx, area
dx = ( x2 - x1 ) / REAL(npts - 1)
DO i = 1, npts
    x(i) = dx * REAL(i-1)
    y(i) = SIN(x(i))
END DO
CALL integ_d ( x, y, npts, x1, x2, area, error )
WRITE (*,1000) area
1000 FORMAT ( ' The area under curve SIN(x) from 0. to PI is: ', F10.6)
```

Supplemental Library Descriptions to accompany
Fortran 90/95 for Scientists and Engineers, by Stephen J. Chapman

Result:

The area under curve $\text{SIN}(x)$ from 0. to PI is: 1.999836

interp (Single / Double Precision Real)

Purpose: To linearly interpolate the value y_o at position x_o , given a set of (x,y) measurements organized in increasing order of x .

Usage: `USE booklib`
`CALL interp (X, Y, NPTS, X0, Y0, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
X	R/D	NPTS	I	Values of independent variable x
Y	Same as X	NPTS	I	Values of dependent variable y
NPTS	I		I	Number of (x,y) measurements
X0	Same as X		I	Point at which to interpolate Y0
Y0	Same as X		O	Interpolated value at point X0
AREA	Same		O	Integrated value
ERROR	I		O	Error flag: 0 = No error -1 = $X0 < X(1)$ 1 = $X0 > X(NPTS)$

Algorithm:

Find points $X(I)$ and $X(I+1)$ that straddle $X0$
 $\text{slope} \leftarrow (Y(I+1) - Y(I)) / (X(I+1) - X(I))$
 $Y0 \leftarrow \text{slope} * (X0 - X(I)) + Y(I)$

This subroutine requires that $X0$ fall between two points in array X . If $X0$ is outside the range of the points in X , the subroutine returns an error. (Also see subroutines `spline_fit` and `spline_int`.)

Example:

This example interpolates the value at $X0 = 5.2$.

```
USE booklib
INTEGER :: npts = 4, error
REAL, DIMENSION(4) :: x = (/ 3., 4., 5., 6. /)
REAL, DIMENSION(4) :: y = (/ 2.0, 0.9, 0.0, -0.9 /)
REAL :: x0 = 5.2, y0
CALL interp ( x, y, npts, x0, y0, error )
WRITE (*,1000) ' x0 = ', x0, ' y0 = ', y0
1000 FORMAT (1X,A,F8.3,A,F8.3)
```

Result:

$X0 = 5.200 \quad Y0 = -.180$

lcase/ucase (Character)

Purpose: Subroutine to shift a character string to lower/upper case.

Usage: **USE booklib**
 CALL lcase (STRING)
 CALL ucase (STRING)

Arguments:

Name	Type	Dim	I/O	Description
STRING	CHAR		I/O	Input: Input character string Output: Lower/upper case character string

Algorithm:

Subroutine **lcase** shifts all upper case letters in an input character string to lower case, and leaves all other letters unchanged. Subroutine **ucase** shifts all lower case letters in an input character string to upper case, and leaves all other letters unchanged. They work for both ASCII and EBCDIC collating sequences.

Example:

```
USE booklib
CHARACTER(len=30) :: string = 'This is a Test: 12345%!?. '
WRITE (*,'(A,A)') ' Before LCASE: ', string
CALL lcase ( string )
WRITE (*,'(A,A)') ' After LCASE: ', string
CALL ucase ( string )
WRITE (*,'(A,A)') ' After UCASE: ', string
```

Result:

```
Before LCASE: This is a Test: 12345%!?.
After LCASE:  this is a test: 12345%!?.
After UCASE:  THIS IS A TEST: 12345%!?.
```

lsq_fit (Single/Double Precision Real)

Purpose: Subroutine to perform a least-squares fit of an input data set to the n th order polynomial:

$$y(x) = c_0 + c_1 x + c_2 x^2 + \dots + c_n x^n.$$

Usage: **USE booklib**
 CALL lsq_fit (X, Y, NVALS, ORDER, C, ERROR)

Arguments:

Name	Type	Dim	I/O	Description
X	R/D	NVALS	I	Values of independent variable x
Y	Same as X	NVALS	I	Values of dependent variable y
NVALS	I		I	Number of (x,y) measurements
ORDER	I		I	Order (highest power) of polynomial to fit
C	Same as X	0: ORDER	0	Coefficients of least squares fit
ERROR	I		0	Error flag: 0 = No error 1 = Singular equations 2 = Not enough input values 3 = Illegal polynomial order specified

Algorithm:

Subroutine **lsq_fit** performs a least squares fit of an input data set consisting of (x,y) pairs of data points to an n th order polynomial. The algorithm implemented is described in Exercise 12-6.

Example:

```
! This code fits a 3rd order polynomial to 6 input data points.
! The data points were produced by the eqn:
!      y(x) = 1. - x + x**2 - x**3
USE booklib
REAL, DIMENSION(6) :: x = (/ 0., 1., 2., 3., 4., 5. /)
REAL, DIMENSION(6) :: y = (/ 1., 0., -5., -20., -51., -104. /)
REAL, DIMENSION(0:3) :: c
INTEGER :: nvals = 6, order = 3, error
CALL lsq_fit ( x, y, nvals, order, c, error )
WRITE (*, '(A,4(F10.5,1X))') ' The coefficients are: ', c
```

Result:

The coefficients are: 0.99999 -0.99999 1.00000 -1.00000

mat_inv (single prec. real/double prec. real/single complex/double complex)

Purpose: To invert an $N \times N$ matrix using Gauss-Jordan elimination and the maximum pivot technique.

Usage: `USE booklib`
`CALL mat_inv (A, B, NDIM, N, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
A	R/D/C/Z	ndim x ndim	I	Matrix to invert (May be any kind of real or complex.)
B	Same as A	ndim x ndim	O	Inverse matrix a^{-1} (Same kind as a.)
NDIM	I		I	Declared size of matrices.
N	I		I	No. of rows and columns actually used in a
ERROR	I		O	Error flag: 0 = No error 1 = No inverse found (pivot too small)

Algorithm:

This subroutine uses Gauss-Jordan elimination and the maximum pivot technique to construct the inverse of an $n \times n$ matrix. It initializes matrix **B** to the identity matrix, and then performs Gauss-Jordan elimination on a copy of matrix **A**, applying exactly the same operations to matrix **B** that were applied to matrix **A**. When the operation is over and the copy of **a** contains the identity matrix, **B** will contain matrix A^{-1} . These matrix inversion subroutines suffer from the same conditioning problems as Gaussian elimination subroutines, so the double precision version will be required for large and/or ill-conditioned matrices.

Example:

This example declares two 10 x 10 arrays **a** and **b**, initializes array **a** with a 2 x 2 matrix, and inverts the matrix using subroutine **mat_inv**.

```
USE booklib
IMPLICIT NONE
INTEGER, PARAMETER :: ndim = 10
REAL, DIMENSION(ndim,ndim) :: a, b
INTEGER :: error, i, j, n = 2
a(1,1) = 1.; a(2,1) = 2.; a(1,2) = 3.; a(2,2) = 4.
CALL mat_inv (a, b, ndim, n, error)
WRITE (*,1000) ((b(i,j), j=1, n), i=1, n)
1000 FORMAT (1X, 'b = ', /, (4X, F10.4, 4X, F10.4))
```

Supplemental Library Descriptions to accompany
Fortran 90/95 for Scientists and Engineers, by Stephen J. Chapman

Result:

b =		
	-2. 0000	1. 5000
	1. 0000	- . 5000

nxtmul (Integer)

Purpose: Subroutine to calculate the smallest exponent **EXP** that satisfies the expression **VALUE** <= **MUL** (= **BASE**EXP**). This calculation is useful for sizing FFT's, etc.

Usage: `USE booklib`
`CALL NXTMUL (VALUE, BASE, EXP, MUL)`

Arguments:

Name	Type	Dim	I/O	Description
VALUE	I		I	First matrix to multiply
BASE	I		I	Base value for the exponent
EXP	I		0	Smallest exponent satisfying the inequality given above
MUL	I		0	The next power of BASE that is greater than VALUE : MUL = BASE**EXP

Algorithm:

This subroutine calculates successive powers of the base number **BASE** until one of the exceeds the value **VALUE**. When that happens, the subroutine returns both the exponent **EXP** and the base raised to the exponent **MUL**. The subroutine is useful for calculating the next power of two when working with FFTs. For example, the call

```
CALL nxtmul ( 48, 2, EXP, MUL )
```

would return with **EXP** = 6 and **MUL** = 64, since $2^{**}6 = 64$, which is greater than 48.

Example:

This example calculates the next power of two greater than the number 997:

```
USE booklib
INTEGER :: exponent, mult
CALL nxtmul ( 997, 2, EXP=exponent, MUL=mult )
WRITE (*,1000) exponent, mult
1000 FORMAT ( ' Exponent = ', I6, '    Multiple = ', I6 )
```

Result:

```
Exponent =      10    Multiple =    1024
```

plot (Single Prec. Real/Double Prec. Real)

Purpose: Subroutine to print a line printer plot of a function.

Usage: `USE booklib`
`CALL plot (DATA1, NPTS, LU, MNAMP, MAXAMP)`

Arguments:

Name	Type	Dim	I/O	Description
DATA1	R/D	NPTS	I	Data set to plot
NPTS	I		I	Number of points in input data set
LU	I		I	I/O unit to print plot on.
<i>MNAMP</i>	Same as DATA1		I	Smallest value to plot. The default value is the smallest number in the data set.
<i>MAXAMP</i>	Same as DATA1		I	Largest value to plot. The default value is the largest number in the data set.

Algorithm:

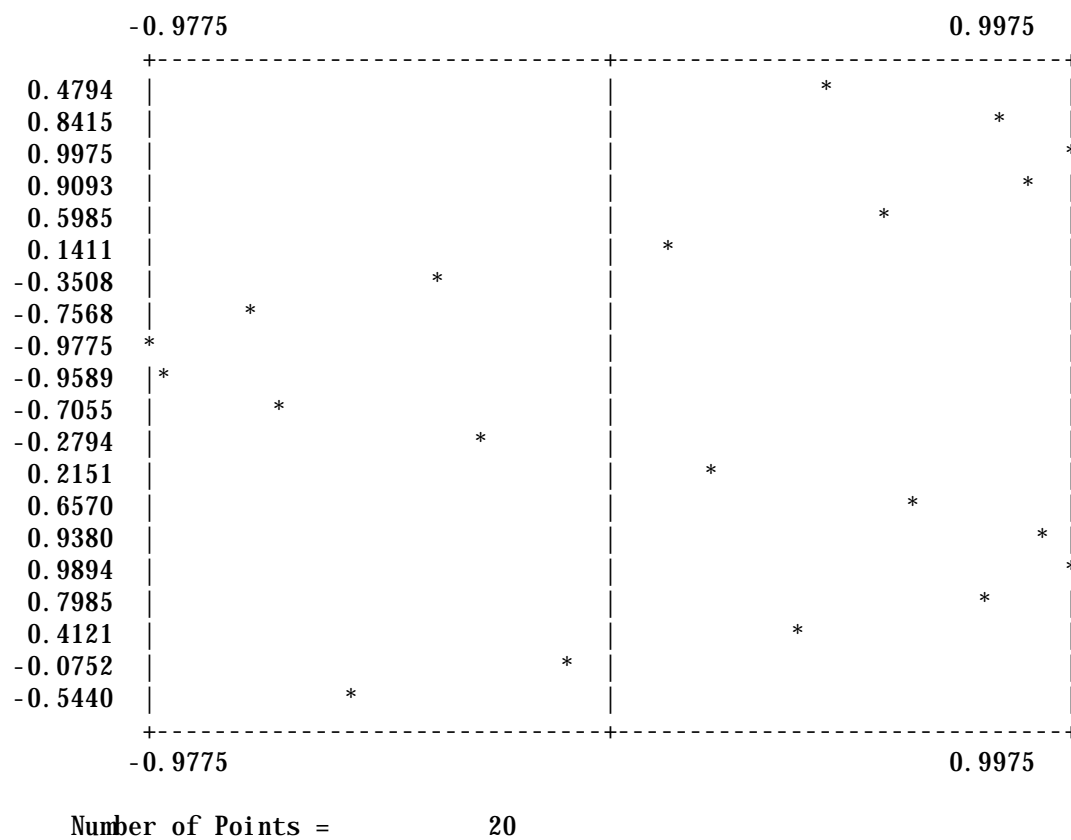
This subroutine makes a line printer plot of an input data set on the device specified by i/o unit LU.

Example:

This example plots the function $\sin(x)$ for 0 to 10 in steps of 0.5. Note that the values of *MNAMP* and *MAXAMP* are defaulted.

```
USE booklib
REAL, DIMENSION(20) :: data1
DO i = 1, 20
    data1(i) = SIN(REAL(i)/2.)
END DO
CALL plot (data1, 20, 6 )
```

Result:



plotxy (Single Prec. Real/Double Prec. Real)

Purpose: Subroutine to print a line printer cross-plot a set of (x,y) data points.

Usage: `USE booklib`
`CALL plotxy (X, Y, NPTS, LU, MINX, MAXX, MINY, MAXY, &`
`NBINX, NBINY)`

Arguments:

Name	Type	Dim	I/O	Description
X	R/D	NPTS	I	X values of points to plot.
Y	Same as X	NPTS	I	Y values of points to plot.
NPTS	I		I	Number of points in input data set
LU	I		I	I/o unit to print plot on.
<i>MINX</i>	Same as X		I	Smallest X value to plot. The default value is the smallest number in the data set.
<i>MAXX</i>	Same as X		I	Largest X value to plot. The default value is the largest number in the data set.
<i>MINY</i>	Same as X		I	Smallest Y value to plot. The default value is the smallest number in the data set.
<i>MAXY</i>	Same as X		I	Largest Y value to plot. The default value is the largest number in the data set.
<i>NBINX</i>	I		I	Number of x bins to plot. This value is in the range $1 \leq \textit{NBINX} \leq 65$, and the default is 65.
<i>NBINY</i>	I		I	Number of y bins to plot. This value is in the range $1 \leq \textit{NBINY} \leq 65$, and the default is 65.

Algorithm:

This subroutine makes a line printer plot of an input data set consisting of **NPTS** pairs of (x,y) values on the device specified by i/o unit **LU**.

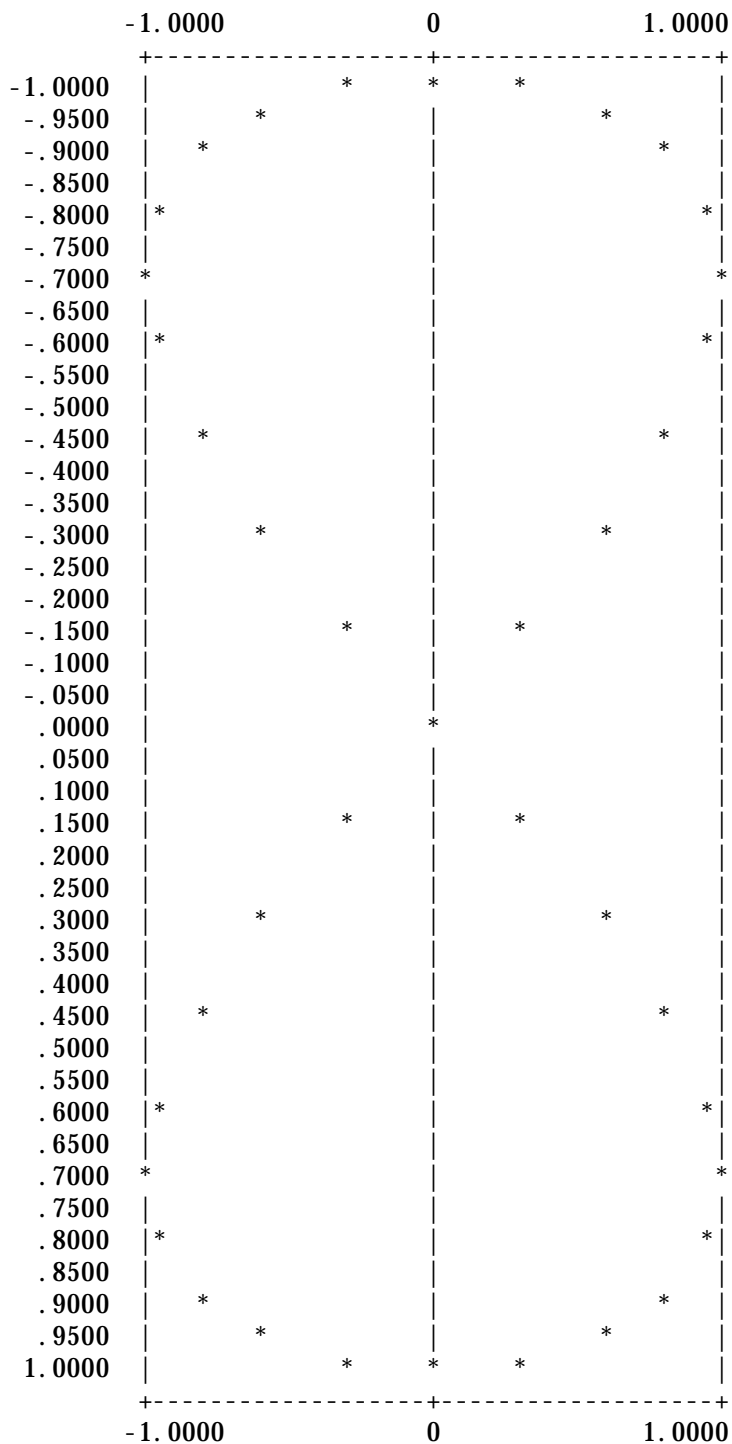
Example:

This example plots the (x,y) pairs formed by the functions $x(t) = \sin(t)$ and $y(t) = \sin(2t)$ for $t = 0$ to 2π in steps of $\pi/20$. Note that the values of **MINX**, **MAXX**, **MINY**, and **MAXY** are defaulted.

```
USE booklib
REAL, PARAMETER :: pi = 3.141593
REAL, DIMENSION(40) :: x, y
INTEGER :: npts = 40
DO i = 1, NPTS
    x(i) = SIN(REAL(i)*(pi/20.))
    y(i) = SIN(2.*REAL(i)*(pi/20.))
END DO
```

CALL plotxy (x, y, npts, 6, NBINX = 41, NBINY = 41)

Result:



Number of Points = 40

random_u (Single prec. real function)
random_n (Single prec. real function)
random_r (Single prec. real function)

Purpose: These procedures generate a sequence of pseudorandom numbers. Function **random_u** generates numbers uniformly distributed in the range [0,1). Function **random_n** generates a normal or Gaussian distribution with zero mean and a standard deviation of 1.0. Function **random_r** generates a Rayleigh distribution with a mean of 1.25 and a standard deviation equal to $\sqrt{\frac{4}{p}-1}$ times the mean.

Usage: `USE booklib`
`value = random_u()`
`value = random_n()`
`value = random_r()`

Arguments:

Name	Type	Dim	I/O	Description
none				

Algorithm:

Functions **random_u**, **random_n**, and **random_r** generate pseudorandom number sequences of the specified distributions. These functions are convenient if a random number is needed as a part of a larger calculation.

Example:

This example uses function **random_u** to generate 20 random numbers between 0 and 1.

```
USE booklib
WRITE (*,*) 'Uniform random number sequence:'
DO i = 1, 4
  WRITE (*, '(1X, 5F10.6)') ( random_u(), j=1, 5 )
END DO
```

Result:

```
Uniform random number sequence:
0.434307 0.454378 0.914314 0.482636 0.719896
0.731396 0.896297 0.470225 0.098918 0.326672
0.703519 0.228887 0.617632 0.079673 0.671855
0.257803 0.743262 0.741481 0.954373 0.729823
```


sgesv (Single Prec. Real)
dgesv (Double Prec. Real)
cgesv (Single Prec. Complex)
zgesv (Double Prec. Complex)

Purpose: To solve a system of N simultaneous equations in N unknowns of the form $A \cdot X = B$, where A is an $N \times N$ matrix, and B is an N -dimensional column vector. These subroutines can solve for multiple right hand sides simultaneously, so B can have multiple columns. These subroutines are from the LAPACK library.

Usage: `USE lapack_s`
`CALL sgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)`
`CALL dgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)`
`CALL cgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)`
`CALL zgesv(N, NRHS, A, LDA, IPIV, B, LDB, INFO)`

Arguments:

Name	Type	Dim	I/O	Description
N	I		I	Order of the system of equations.
NRHS	I		I	Number of right hand sides, that is the number of columns in matrix B . $NRHS \geq 1$.
A	R/D/C/Z	LDA x N	I/O	Coefficients of X
LDA	I		I	Leading dimension of array A . $LDA \geq \text{MAX}(1, N)$
IPIV	I	N	0	The pivot indices that define the permutation matrix P ; row i of the matrix was interchanged with $IPIV(i)$.
B	Same as A	LDB x NRHS	I/O	Input: the right hand side matrix B . Output: the solution matrix X .
LDB	I		I	Leading dimension of array B . $LDB \geq \text{MAX}(1, N)$
INFO	I		0	Error flag: = 0: Success < 0: if $INFO = -i$, the i^{th} argument had an illegal value > 0: if $INFO = i$, $U(i, i) = 0$, and the matrix is singular.

Algorithm:

This subroutine uses LU decomposition with partial pivoting and row interchanges to factor A as $A = P \cdot L \cdot U$, where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations $A \cdot X = B$.

Example:

This example calculates the solution to a 2 x 2 set of equations, and prints the results. The arrays are declared large enough for a 4 x 4 set of equations, but only a part of each array is used in this problem.

```
USE lapack_s
INTEGER :: info, lda = 4, ldb = 4, n = 2, nrhs = 1
INTEGER, DIMENSION(4) :: ipiv
REAL, DIMENSION(4,4) :: a
REAL, DIMENSION(4) :: b = (/ 5., 2., 0., 0. /)
a(1,1) = 1.; a(1,2) = 4.; a(2,1) = 2.; a(2,2) = -3.
CALL sgesv( n, nrhs, a, lda, ipiv, b, ldb, info )
WRITE (*,1000) b(1), b(2)
1000 FORMAT ( ' The solution is X(1) = ', F10.4, ' and X(2) = ', F10.4)
```

Result:

The solution is X(1) = 2.0909 and X(2) = .7273

simul (single prec. real/double prec. real/single complex/double complex)

Purpose: To solve a system of N simultaneous equations in N unknowns of the form $AX = B$, where A is an $N \times N$ matrix, and B is an N -dimensional column vector.

Usage: `USE booklib`
`CALL simul (A, B, SOLN, NDIM N, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
A	R/D/C/Z	ndim x ndim	I	Coefficients of X
B	Same as A	ndim	I	Vector of constant terms
SOLN	Same as A	ndim	O	Solution to the system of equations
NDIM	I		I	Declared size of arrays.
N	I		I	Order of the system of equations.
ERROR	I		O	Error flag: 0 = No error 1 = Singular equations

Algorithm:

This subroutine uses the Gauss-Jordan method with maximum pivots for finding the solution to a system of simultaneous equations.

Example:

This example calculates the solution to a 2×2 set of equations, and prints the results. The arrays are declared large enough for a 4×4 set of equations, but only a part of each array is used in this problem.

```
USE booklib
INTEGER :: error
REAL, DIMENSION(4,4) :: a
REAL, DIMENSION(4) :: b = (/ 5., 2., 0., 0. /), soln
a(1,1) = 1.; a(1,2) = 4.; a(2,1) = 2.; a(2,2) = -3.
CALL simul ( a, b, soln, 4, 2, error )
WRITE (*,1000) soln(1), soln(2)
1000 FORMAT (' The solution is X(1) = ', F10.4, ' and X(2) = ', F10.4)
```

Result:

The solution is $X(1) = 2.0909$ and $X(2) = .7273$

sinc (Single Prec. Real / Double Prec. Real function)

Purpose: To calculate the sinc function: $\text{sinc}(x) = \sin(x) / x$.

Usage: `USE booklib`
`result = sinc(X)`

Arguments:

Name	Type	Dim	I/O	Description
X	R/D		I	Value for which to calculate the sinc function

Algorithm:

This function calculates the function $\text{sinc}(x) = \frac{\sin(x)}{x}$, with special handling of the computation near $x = 0$. The result is of the same kind as the input argument (single or double precision real).

Example:

This example calculates $\text{sinc}(x)$ for an arbitrary value of x , and prints the results.

```
REAL :: x
WRITE (*,*) 'Enter value of X: '
READ (*,*) x
WRITE (*,1000) x, sinc(x)
1000 FORMAT (1X, ' sinc(', F10.4, ') = ', F10.4)
```

Result:

```
Enter value of X:
1.0
SINC(    1.0000) =      .8415
```

spline_fit (single prec. real/double prec. real)

Purpose: To perform a cubic spline fit to an input data set.

Usage: `USE booklib`

`CALL spline_fit (X, Y, N, YPP, ERROR, YP1, YPN)`

Arguments:

Name	Type	Dim	I/O	Description
X	R/D	N	I	X coefficients of data to fit. <i>X values must be monotonically increasing.</i>
Y	Same as A	N	I	Y coefficients of data to fit
N	I		I	Number of data points
YPP	Same as A	N	O	Second derivatives of curves at each point
ERROR	I		O	Error flag: 0 = No error 1 = Insufficient data
YP1	Same as A		I	First derivative at the beginning of the data set (optional)
YPN	Same as A		I	First derivative at the end of the data set (optional)

Algorithm:

This subroutine calculates a set of polynomials that fit an input data set of (x,y) points with a curve which is smooth in the first derivative and continuous in the second derivative, both within an interval and at its boundaries. There are two possible boundary conditions at the edges of the data set. If the values of first derivatives are specified at those points, then the equations will be constrained to have those values at the boundaries. If not, then the subroutine will solve for the *natural cubic spline*, which satisfies the condition that the second derivatives at the beginning and end of the data set are 0.

Example:

This example calculates the spline fit coefficients for ten points from the function $f(x) = \sin x$, and compares the resulting derivatives values calculated analytically.

```

USE booklib
INTEGER, PARAMETER :: n = 10
REAL, PARAMETER :: pi = 3.141593
REAL, DIMENSION(n) :: x, y, ypp
INTEGER :: error
REAL :: yp1, ypn
DO i = 1, n                                ! Generate input pts
    x(i) = REAL(i-1) * pi / REAL(n-1)
    y(i) = sin(x(i))
END DO
yp1 = cos(x(1))                            ! Set deriv at start
ypn = cos(x(n))                            ! Set deriv at end

```

```
call spline_fit(x, y, n, ypp, error, yp1, ypn)  ! Fit curves

WRITE (*, '(18X, A, T35, A)') 'Spline', 'Actual'
WRITE (*, '(T6, A, T17, A, T33, A)') 'angle', '2nd deriv', '2nd deriv'
DO i= 1, n
    WRITE (*, '(1X, F8. 2, 2F16. 6)') x(i), ypp(i), -sin(x(i))
END DO
```

Result:

	Spline	Actual
angle	2nd deriv	2nd deriv
0.00	-0.000831	0.000000
0.35	-0.345284	-0.342020
0.70	-0.649401	-0.642788
1.05	-0.874837	-0.866026
1.40	-0.994851	-0.984808
1.75	-0.994850	-0.984808
2.09	-0.874840	-0.866025
2.44	-0.649396	-0.642787
2.79	-0.345289	-0.342020
3.14	-0.000825	0.000000

spline_int (single prec. real/double prec. real)

Purpose: To interpolate points using the cubic spline fit polynomials calculated by subroutine `spline_fit`.

Usage: `USE booklib`
`CALL spline_int (X, Y, N, YPP, X0, Y0, ERROR)`

Arguments:

Name	Type	Dim	I/O	Description
X	R/D	N	I	X coefficients of data to fit. <i>X values must be monotonically increasing.</i>
Y	Same as A	N	I	Y coefficients of data to fit
N	I		I	Number of data points
YPP	Same as A	N	I	Second derivatives of curve at each point, calculated by subroutine <code>spline_fit</code>
X0	Same as A		I	Point at which to interpolate value
Y0	Same as A		O	Interpolated value
ERROR	I		O	Error flag: 0 - No error 1 - $\Delta x = 0$

Algorithm:

This subroutine calculates an interpolated value `Y0` at position `X0` using the cubic spline coefficients calculated by subroutine `spline_fit`. It first determines the pair of points which `X0` lies between, and then uses the cubic equation for that particular interval to estimate `Y0`. (Also see procedure `interp`.)

Example:

This example interpolates a value at $x_0 = \pi/2$ from the function $f(x) = \sin x$, using cubic spline coefficients calculated from subroutine `spline_fit`.

```
USE booklib
INTEGER, PARAMETER :: n = 10
REAL, PARAMETER :: pi = 3.141593
REAL, DIMENSION(n) :: x, y, ypp
REAL :: yp1, ypn, y0
INTEGER :: error
DO i = 1, n
    x(i) = REAL(i-1) * pi / REAL(n-1)
    y(i) = sin(x(i))
END DO
yp1 = cos(x(1))
ypn = cos(x(n))
call spline_fit(x, y, n, ypp, error, yp1, ypn)
call spline_int(x, y, n, ypp, pi/2, y0, error)
```

Supplemental Library Descriptions to accompany
Fortran 90/95 for Scientists and Engineers, by Stephen J. Chapman

```
WRITE (*, '(1X, 2(A, F16.6))') 'Actual = ', sin(pi/2.), &  
                                ' Interp = ', y0  
END PROGRAM
```

Result:

```
Actual =          1.000000 Interp =          0.999960
```


ssort (Integer/Single Prec. Real/Double Prec. Real/Character)

Purpose: To sort an array into ascending order using the selection sort algorithm.

Usage: `USE booklib`
`CALL ssort (ARRAY, N)`

Arguments:

Name	Type	Dim	I/O	Description
ARRAY	I/R/D/C	N	I/O	Array to sort
N	I		I	Number of elements in array

Algorithm:

This subroutine sorts arrays into ascending order using the selection sort algorithm. This algorithm is very inefficient for large data sets, and is included here only for comparison to the better heapsort algorithm. Use the heapsort algorithm instead of this one.

Example:

This example declares an integer array `iarray` and initializes it with 15 values. It uses subroutine `ssort` to sort the array into ascending order.

```
USE booklib
INTEGER, DIMENSION(15) :: iarray = &
    (/ -100, 0, -20, 1, -20, &
       90, -123, 602, 5, 17, &
       91, -4, 0, 37, -11 /)
INTEGER :: n = 15
WRITE (*,*) ' iarray before sorting: '
WRITE (*, '(3X,5I6)') iarray
CALL ssort ( iarray, n )
WRITE (*,*) ' iarray after sorting: '
WRITE (*, '(3X,5I6)') iarray
```

Result:

```
iarray before sorting:
-100    0   -20    1   -20
  90 -123  602    5    17
  91   -4    0   37   -11
iarray after sorting:
-123 -100  -20  -20  -11
  -4    0    0    1    5
  17   37   90   91  602
```

statistics (Single Prec. Real/Double Prec. Real)

Purpose: To calculate the user-requested statistical values associated with a data set.

Usage: `USE booklib`
`CALL statistics (A, N, ERROR, AVE, STD_DEV, MEDIAN)`

Arguments:

Name	Type	Dim	I/O	Description
A	R	N	I	Data set to analyze
N	I		I	Number of data points
ERROR	I		0	Error flag: 0 = No error 1 = SD invalid ($N = 1$) 2 = AVE and SD invalid ($N < 1$)
AVE	R		0	Average of data set
STD_DEV	R		0	Standard deviation of data set
MEDIAN	R		0	Standard deviation of data set

Algorithm:

This subroutine calculates the average, and standard deviation, and median of an input data set, of the corresponding optional arguments are included in the subroutine call. The formulas use are:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

$$\sigma = \sqrt{\frac{N \sum_{i=1}^N x_i^2 - \left(\sum_{i=1}^N x_i \right)^2}{N(N-1)}}$$

and

median = middle value of sorted data set

Example:

This example calculates average and median of a small data set.

```
USE booklib
REAL :: ave, med
INTEGER :: error
REAL, DIMENSION(7) :: a = (/ 1., 4., 1., -4., 0., 2., 6. /)
CALL statistics ( a, 7, error, AVE=ave, MEDIAN=med )
WRITE (*,1000) ave, med
1000 FORMAT (' AVE = ', F10.4, '      median = ', F10.4)
```

Supplemental Library Descriptions to accompany
Fortran 90/95 for Scientists and Engineers, by Stephen J. Chapman

Result:

AVE = 1.4286 median = 1.0000