D I S G C L 3.2

A Graphics

Command Language

by

Helmut Michels

Chapter 1

Overview

1.1 Introduction

This manual describes the graphics command language DISGCL which is an interpreter based on the graphics software DISLIN. All DISLIN statements can be written to a script file and then be executed with DISGCL, or can be entered in an interactive mode.

Similar to programming languages such as Fortran and C, high-level language elements can be used within DISGCL. These are variables, operators, expressions, array operations, loops, if and switch statements, user-defined subroutines and functions, and file I/O routines.

An easy to use interface for data input is given to include data into DISGCL jobs. The format of data files is very simple and useful for most DISLIN plotting routines.

Several quickplots are offered by DISGCL which are collections of DISLIN statements to display data with one command.

1.2 Syntax of the DISGCL Command

The DISGCL command has the following syntax:

Command:	DISGCL [filename[.gcl]] [args] [options]
filename	is the name of a DISGCL script file. The extension '.gcl' is optional.
args	are optional arguments that can be passed to DISGCL scripts. The arguments are stored in the system variables %ARG1, %ARG2, ..., %ARGn, or can be requested with the function GETARG (i), 1 <= i <= n. The number of passed arguments is stored in the system variable %NARGS.
options	is an optional field of keywords separated by blanks:
-D=device	defines the format of the metafile created by DISLIN. This parameter will overwrite the keyword in the DISLIN routine METAFL and can have the same values as the parameter in METAFL.
-F=file	defines the file used for data input. This parameter will overwrite the file parameter in the routine DATFIL.

-I=file	replaces the file parameter of the first INCLUDE statement in an DISGCL script file. This option can be used to initialize variables with different values.
-f	means that the extension '.gcl' is not added to the filename.
-v	prints program version and author.

Notes:

- If no parameters are specified, DISGCL runs in interactive mode.
- DISGCL searches the current working directory for the DISGCL script file. If the search fails, DISGCL searches the directory defined by the environment variable GCL_PATH.
- On UNIX systems, an DISGCL script file can be executed directly if the following line is included at the beginning of the script file:

  #! /path/disgcl -f

  where path is the directory containing the disgcl executable.

1.3 Syntax of DISGCL Script Files

DISGCL script files must have the following syntax:

- A DISGCL script file must begin with the indentifier '%GCL'.
- Each line may contain up to 132 characters.
- The current statement can be continued on the next line if a masterspace (@) is used at the end of the line.
- Lines are allowed to carry trailing comment fields, following a double slash (//) or the '#' character. Empty lines are also be interpreted as comment lines.
- Keywords and routine names can be in upper and lowercase letters.
- String constants must be enclosed in a pair of either apostrophes or quotation marks.

Example:

```
%GCL
SUM = 0
DO I = 1, 10
   SUM = SUM + I
END DO
PRINT SUM
```

1.4 Syntax of Statements

The following statements can be used in DISGCL script files, or can be typed directly at the DISGCL prompt.

Command	Description
%GCL	Identifier for DISGCL script files.
// Comment or # Comment	Comment line and inline comments.
routine (parameter list)	Call of a DISLIN or DISGCL routine.
CALL routine (parameter list)	Call of a user-defined subroutine.
v = function (parameter list)	Call of a DISLIN, DISGCL or user-defined function.
v = expression	Assigns the value of the expression to the variable v.
command [parameter list]	DISGCL command.
vray = { constant list }	Creates and initializes an integer or floatingpoint array.
IF (expression) statement	IF statement (conditional statement).

IF (expression)
  statements
ELSE IF (expression)
  statements
ELSE
  statements
END IF
	IF construct.
	Up to 8 IF constructs can be nested.
	The ELSE IF and the ELSE parts are optional.

DO v = expr1, expr2 [,expr3] statements END DO	DO loop. Up to 8 loops can be nested.

WHILE (expr)
  statements
END WHILE
	WHILE loop.
	Up to 8 loops can be nested.

SWITCH (iexpr)
  CASE n1:
  statements
  CASE n2:
  statements
.....
  DEFAULT:
  statements
END SWITCH
	SWITCH statement where iexpr
	must be an integer expression and
	n1, n2, ... integer constants.
	Up to 8 SWITCH statements can be nested.

label:	Label statement.
GOTO label	GOTO statement.
$command	Executes a system command.

Figure 1.1: DISGCL Statements

1.5 Data Types

Variables in DISGCL are dynamic. They don't have to be declared, and they can change their types during the lifetime of a DISGCL session. The following data types are known by DISGCL:

CHAR	an 8-bit integer in the range -128 to 127.
BYTE	an 8-bit integer in the range 0 to 255.
SHORT	an 16-bit integer in the range -32768 to 32767.
INT	an 32-bit integer in the range -2147483648 to 2147483647.
FLOAT	an 32-bit floatingpoint number in the range 1.2E-38 to 3.4E+38 and with 7-digit precision.
DOUBLE	an 64-bit floatingpoint number in the range 2.2E-308 to 1.8E+308 and with 15-digit precision.
COMPLEX	a pair of 32-bit floatingpoint numbers in the range 1.2E-38 to 3.4E+38.
STRING	a sequence of characters. Strings are stored as CHAR arrays terminated with the ASCII value zero.

1.6 Expressions

An expression is an combination of operands and operators. The operands can be constants, variables and functions, and may be scalars or arrays. Expressions can be assigned to variables or can be passed as parameters to subroutines and functions.

Example:

```
a = 60
x = exp (sin (a * 3.14159))
```

1.7 Quickplots

DISGCL offers several quickplots which are collections of DISLIN routines that can display data with one command. For example, the DISGCL command PLOT displays two-dimensional curves.

Example:

```
x = falloc (100)
plot x, sin (x/5)
```

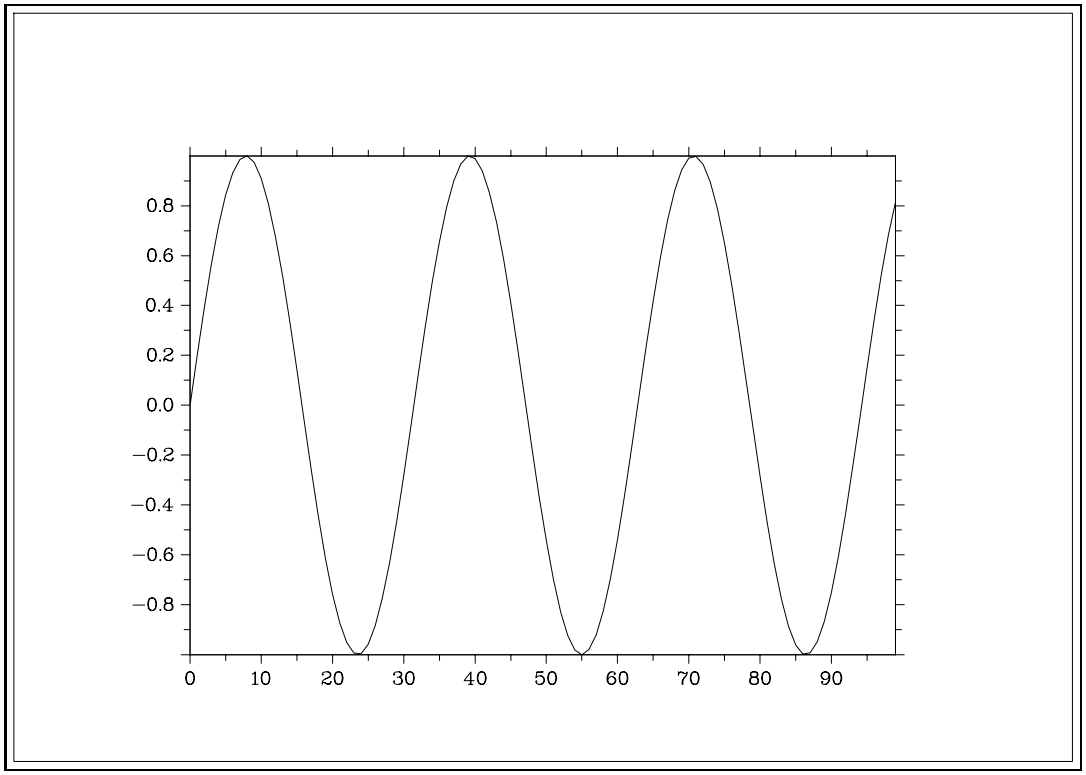

Figure 2.1: Example of the PLOT Command

Note:

All quickplots have corresponding widget interfaces that can be executed with the command

disgcl quickplot

where quickplot is the name of a quickplot. The widget interfaces for quickplots expect data in the form of data files described in chapter 8, 'Data Files'.

1.8 FTP Sites, WWW Homepage

DISGCL and DISLIN are available via ftp anonymous from the following sites:

ftp://ftp.gwdg.de/pub/grafik/dislin
ftp://linhmi.mpae.gwdg.de/pub/dislin

The DISLIN Homepage is:

http://www.linmpi.mpg.de/dislin

1.9 Reporting Bugs

DISGCL bugs can be reported to the author:

Helmut Michels
Max-Planck-Institut fuer Aeronomie
D-37191 Katlenburg-Lindau, Max-Planck-Str. 2, Germany
E-Mail: michels@linmpi.mpg.de
Tel.: +49 5556 979 334
Fax: +49 5556 979 240

Chapter 2

Data Types, Variables

This chapter explains the DISGCL data types and shows how to specify constants and variables.

2.1 Data Types

As described in the last chapter, DISGCL data can have the following types:

Type	Number of Bytes	Range
CHAR	1	-128 - 127
BYTE	1	0 - 255
SHORT	2	-32768 - 32767
INT	4	-2147483648 - 2147483647
FLOAT	4	1.2E-38 - 3.4E+38
DOUBLE	8	2.2E-308 - 1.8E+308
COMPLEX	8	1.2E-38 - 3.4E+38
STRING	n + 1	

Figure 2.1: Data Types

2.2 Variables

All data in DISGCL are variables or constants. As in other programming languages, variables can change their values during the lifetime of a DISGCL session. But in DISGCL, variables can also change their types, and they don't have to be declared.

The following rules are applied to variables:

- The first character must be a letter. Other characters can be letters, digits, or underscores.
- The first 16 characters of variable names are significant.
- DISGCL is not case-sensitive. Variable names, keywords, functions, routines and parameters can be specified in uppercase and lowercase letters.

2.3 System Variables

System variables are special variables with a predefined meaning. For example, system variables can be used to set options for quickplots. System variables begin with the '%' character and are available to all DISGCL units such as subroutines and functions.

2.4 Specifying Constants

Constants are data that cannot change their values during the life of a DISGCL session. Constants can be integers, floatingpoint numbers and strings.

Integer constants can be specified in decimal or hexadecimal notation. Floatingpoint constants contain a decimal point and can have an exponential part preceded by e or E. String constants must be enclosed in a pair of either apostrophes or quotation marks. Complex constants contain the keyword 'COMPLEX' and a real and imaginary part. The imaginary part is optional and assumed to be zero if it is omitted.

Examples:

Constant	Type
120	Decimal
0xFF	Hexadecimal
0.56	Floatingpoint
3.6E2	Floatingpoint
"ABC"	String
'abc'	String
COMPLEX (3,4)	COMPLEX
COMPLEX (7)	COMPLEX

Figure 2.2: Constants

2.5 Arrays

An array is a collection of data that share the same type and a common name. Array elements can be accessed by specifying subscripts in square brackets. Multidimensional arrays can be defined and accessed by specifying subscripts separated by commas.

Examples:

A[i] = 5
B[i, j] = 8
C[0, 10, 5] = 3.5

There are several ways to create arrays in DISGCL:

- If an expression is assigned to a variable and the value of the expression is an array, the variable will also be an array.

- The DISGCL commands CHAR, BYTE, SHORT, INT, FLOAT, DOUBLE and COMPLEX create corresponding arrays and initialize them with zeros.

  Example:

  INT A[10], B[20,10]

- Integer and floatingpoint arrays can be created and initialized with the statement:

  vray = { list }

  where list is a constant list of integers or floatingpoint numbers separated by commas.

Notes:

- Array elements begin with the number 0.
- If a subscript of an array is out of range, DISGCL prints a warning and cancels the calculation.
- Multidimensional arrays are stored by rows.
- If a subscript appears in a string or a CHAR array, the corresponding element is handled as an integer where the value of the integer is the ASCII code of the element.

2.6 Subscripts

Subscripts can be used to access single array elements or sections of arrays. The following are examples of array subscripts:

A[i]	Element i of array A
A[i:j]	Array section of size j - i + 1
B[i1:i2, j]	Elements i1 to i2 of column j
B[:i2, j]	Elements 0 to i2 of column j
B[i1:, j]	Elements i1 to m - 1 of column j
B[*, j]	The whole column j.

2.7 Character Arrays and Strings

Strings in DISGCL are stored as character arrays terminated with ASCII value zero. Normally, strings and character arrays can be used in the same way if character arrays contain a string terminator. Some I/O functions require character arrays instead of strings to store characters. Character arrays can be defined with the CHAR command.

Chapter 3

Expressions and Operators

An expression is an combination of operands and operators. The operands can be constants, variables and functions, and may be scalars or arrays. The operators are displayed in the following paragraph.

3.1 Operators

Figure 3.1 shows all DISGCL operators and summarizes the rules for precedence and associativity of operators.

Priority	Operator	Meaning	Associativity
1	**	Exponentiation	Right to left
2	-	Unary minus	Right to left
	+	Unary plus	Right to left
	!	Logical NOT	Right to left
3	*	Multiplication	Left to right
	/	Division	Left to right
	%	Modulus	Left to right
4	+	Addition	Left to right
	-	Subtraction	Left to right
5	<	Less than	Left to right
	<=	Less than or equal	Left to right
	>	Greater than	Left to right
	>=	Greater than or equal	Left to right
	==	Equal	Left to right
	!=	Not equal	Left to tight
6	&&	Logical AND	Left to right
	||	Logical OR	Left to right

Figure 3.1: Operators

The first column in figure 3.1 gives the precedence of an operator. This means that when two operators have different precedence, the operator with the higher precedence is evaluated first. The highest precedence is 1.

When two operators have the same precedence, they are evaluated in the direction specified in the last column 'Associativity'.

The order of normal precedence can be changed by enclosing expressions in parenthesis.

Examples:

3 * 2 + 1	has the value 7 since the operator '*' has a higher precedence than the operator '+'.
2 * 3 / 2	has the value 3 since '*' and '/' are evaluated from left to right.
2 ** 2 ** 3	has the value 256 since '**' is evaluated from right to left.

3.2 Array Operations

In DISGCL, the operands in an expression can be scalars and arrays. Figure 3.2 shows the allowed array operations:

Operation	Value
array + array	array
array - array	array
array / array	array
array * array	array
array ** array	array
array + scalar	array
scalar + array	array
array - scalar	array
scalar -array	array
array * scalar	array
scalar * array	array
array / scalar	array
scalar / array	array
array ** scalar	array
scalar ** array	array

Figure 3.2: Array Operations

3.3 Type Conversions

When an operator has operands of different types, they are converted to a common type. Figure 3.3 shows the rules for type conversions if both operands are scalars or both are arrays:

	BYTE	**SHORT**	**INT**	**FLOAT**	**DOUBLE**
BYTE	BYTE	SHORT	INT	FLOAT	DOUBLE
SHORT	SHORT	SHORT	INT	FLOAT	DOUBLE
INT	INT	INT	INT	FLOAT	DOUBLE
FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	DOUBLE
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE

Figure 3.3: Type Conversions

Notes:

- The only allowed operation between strings is the concatination with the '+' operator.
- Complex operands can be used with other operands. The result is complex.

When one operand is a scalar and the other operand an array, the value of the expression is also an array. The following figure shows the rules for type conversions between scalars and arrays:

Scalar/Array	**BYTE**	**SHORT**	**INT**	**FLOAT**	**DOUBLE**
BYTE	BYTE	SHORT	INT	FLOAT	DOUBLE
SHORT	BYTE	SHORT	INT	FLOAT	DOUBLE
INT	BYTE	SHORT	INT	FLOAT	DOUBLE
FLOAT	*	*	FLOAT	FLOAT	DOUBLE
DOUBLE	*	*	FLOAT	FLOAT	DOUBLE

Figure 3.4: Type Conversions between Scalars and Arrays

Note:

A '*' means that this operation is not allowed. A warning will be displayed by DISGCL if this operation appears in an expression.

3.4 Type Conversion Functions

DISGCL provides a set of functions that convert types of variables and expressions. The conversion functions are as follows:

Function	Meaning
BYTE (x)	converts x to BYTE.
CHAR (x)	converts x to CHAR.
SHORT (x)	converts x to SHORT.
INT (x)	converts x to INT.
FLOAT (x)	converts x to FLOAT. If x is complex, FLOAT returns the real part of x.
DOUBLE (x)	converts x to DOUBLE.
COMPLEX (x, y)	converts to COMPLEX. x is converted to the real, y to the imaginary part. The second parameter y is optional.
STRING (x)	converts x to STRING.

Figure 3.5: Type Conversion Functions

Note:

Normally, type conversion functions check operands for overflow and print warnings.

Examples:

Example	Meaning
INT A[100] A = FLOAT (A)	converts A to a floatingpoint array.
Z = COMPLEX (3, 4)	creates the complex number Z with the real part 3 and the imaginary part 4.
Z = COMPLEX (3)	creates the complex number Z with the real part 3 and the imaginary part 0.
IR = {65, 66, 67, 68} S = STRING (IR)	creates the string S = 'ABCD'.

Figure 3.6: Type Conversion Examples

Chapter 4

Statements

This chapter describes the statements that can appear in DISGCL script files, or can be entered at the DISGCL prompt.

4.1 Identification of Script Files

DISGCL script files must begin with the string '%GCL' where the string can appear in upper or lowercase letters.

Example:

```
%GCL
PRINT  3 + 4
```

4.2 Comment Lines

Empty lines and lines beginning with a double slash (//) or a '#' character are interpreted as commend lines. Lines are also allowed to carry trailing comment fields, following a double slash.

Example:

```
%GCL
// This is a comment
PRINT  3 + 4          // This is also a comment
```

4.3 Calling DISLIN Routines

About 400 DISLIN routines for plotting and parameter setting can be executed from DISGCL. DISLIN routines can either be subroutines that return no value, or functions that return a value.

4.3.1 Calling DISLIN Subroutines

DISLIN subroutines can be executed with the statement:

routine (list)

where routine is the name of a DISLIN routine and list the parameters of the routine separated by commas.

Example:

```
%GCL
METAFL   ('CONS')
DISINI   ()
GRAF     (0., 10., 0., 2., 0., 5., 0., 1.)
DISFIN   ()
```

4.3.2 Calling DISLIN Functions

DISLIN functions can be executed with the statement:

v = function (list)

where function is the name of a DISLIN function and list the parameters of the function separated by commas.

4.3.3 Passing Parameters to DISLIN Routines

Actual parameters in DISLIN routines can be constants, variables and expressions. Integer constants must be specified without decimal points, floatingpoint constants can be passed without decimal points. Normally, arrays must be passed to DISLIN routines as integer or floatingpoint arrays.

DISGCL checks the number and types of parameters passed to DISLIN routines. If an error occurs, a warning is printed on the screen and the call of the routine will be ignored.

Example:

N = TRMLEN (1)

DISGCL will print the warning:

```
>>>> Paramter mismatch
     User    : TRMLEN  (I)
     Correct : TRMLEN  (S)
```

The abbreviations in parameter mismatch warnings have the meaning:

I	denotes an integer parameter.
X	denotes a floatingpoint, integer or double parameter.
D	denotes a double parameter.
S	denotes a string.
BR	denotes a byte array.
IR	denotes an integer array.
FR	denotes a floatingpoint array.
DR	denotes a double array.

4.4 DISGCL Commands

A DISGCL command can be executed with the statement

command [list]

where command is the name of a DISGCL command and list the parameters of the command separated by commas. Several DISGCL commands are explained in the next chapter.

Example:

PRINT FALLOC(10)

4.5 Initializing Arrays with { }

Integer and floatingpoint arrays can be created and initialized with the statement

vray = { list }

where list is a constant list of integers or floatingpoint numbers separated by commas.

Note: Arrays can also be created and initialized with the functions FALLOC, DALLOC and IALLOC, and with the DISGCL commands BYTE, CHAR, SHORT, INT, FLOAT, DOUBLE and COMPLEX.

4.6 The IF Statement

The IF statement executes a statement if a logical expression is true. The syntax is:

IF (expr) statement

Example:

IF (I < 10) A = 1

4.7 IF Constructs

IF constructs are statements for decision making. The syntax is:

```
IF (expression)
    statements
ELSE IF (expression)
    statements
ELSE
    statements
END IF
```

If expression is true, the statements in the IF block will be executed; if expression is false, control is transferred to the next ELSE IF, ELSE or END IF statement.

Example:

```
IF (A > 0)
    ISIGN = 1
ELSE IF (A < 0)
```

```
    ISIGN = -1
ELSE
    ISIGN = 0
END IF
```

Notes:

- Up to 8 IF constructs can be nested.
- The ELSE IF and ELSE blocks are optional.
- Multiple ELSE IF blocks can be specified.

4.8 SWITCH Statements

The SWITCH statement is a multi-way decision that tests whether an expression matches one of the number of constant integer values. The syntax is:

```
SWITCH (iexpr)
  CASE n1:
   statements
  CASE n2:
   statements
  .........
  DEFAULT:
   statements
END SWITCH
```

Each case must be labeled by an INT constant while iexpr can be an integer expression. If a case matches iexpr, execution starts at that case. The case labeled DEFAULT is executed if none of the other cases are satisfied. The DEFAULT case is optional.

The following example counts the number of characters and the number of blanks in a string .

```
%GCL
S = 'This is a test'
I = 0
NCHAR = 0
NBLANK = 0
WHILE (S[I] != 0)
  SWITCH (S[I])
   CASE 32:
     NBLANK = NBLANK + 1
   DEFAULT:
     NCHAR = NCHAR + 1
  END SWITCH
  I = I + 1
END WHILE
PRINT  NCHAR,  NBLANK
```

Notes:

- Up to 8 SWITCH statements can be nested.
- The BREAK statement causes an immediate exit from a SWITCH statement.

4.9 The DO Statement

The DO statement can be used to repeat statements a set number of times. The syntax is:

```
DO  v = expr1,  expr2  [,expr3]
    statements
END DO
```

where

v	is a loop counter.
expr1	is an expression that initializes v.
expr2	is an expression that defines the end of the loop range.
expr3	is an optional expression that will be used as an increment for the loop counter. The value cannot equal zero. The default value is 1.

The following restrictions apply to DO loops:

- Up to 8 DO loops can be nested.
- Jumping into a DO loop from outside its range is not allowed.
- Overlapping of DO and IF constructs are not allowed.

Notes:

- The loop counter of a DO loop can be modified by statements within the loop.
- Whenever possible, array data should be processed by array operations instead of operations in a loop. Array operations are much more faster than loop operations.

4.10 The WHILE Statement

A WHILE loop repeats as long as a given condition remains true. The syntax is:

```
WHILE (expr)
    statements
END WHILE
```

Example:

```
I = 0
SUM = 0
WHILE (I < 10)
    I = I + 1
    SUM = SUM + I
END WHILE
```

The following restrictions apply to WHILE loops:

- Up to 8 WHILE loops can be nested.
- Jumping into a WHILE loop from outside its range is not allowed.
- Overlapping of WHILE and IF constructs are not allowed.

4.11 The BREAK Statement

The BREAK statement causes an immediate exit from a SWITCH, DO and WHILE construct.

Example:

```
I = 0
WHILE (I < 10)
    I = I + 1
    PRINT I
    IF ( I == 3) BREAK
END WHILE
```

The output of the example is:

```
1
2
3
```

Note:

The BREAK statement ends only the loop in which it appears.

4.12 The CONTINUE Statement

The CONTINUE statement can be used to skip all following statements in a loop and to execute the loop with the next iteration.

Example:

```
DO I = 1, 5
    IF ( I == 3) CONTINUE
    PRINT I
END DO
```

The output of the example is:

```
1
2
4
5
```

4.13 The GOTO Statement

The GOTO statement makes a jump to another part of the DISGCL file. The syntax is:

GOTO Label

where Label is the name of a label. For label names, the same rules as for variable names are applied.

The target of the GOTO statement must be a label statement which has the syntax:

Label:

Example:

```
I = 0
L1:
```

```
I = I + 1
PRINT  I
IF (I < 5) GOTO L1
```

The output of the example is:

```
1
2
3
4
```

4.14 Executing System Commands

System commands can be executed with the statement

$Command

where Command is a system command.

Example:

$DIR

displays the files in the current directory if the operating system is MS-DOS or VMS (use $ls for UNIX).

Chapter 5

DISGCL Commands

This chapter describes several DISGCL commands.

5.1 Termination of DISGCL

The command EXIT terminates a DISGCL session.

The call is: EXIT

5.2 Getting Help

The HELP command activates a menu for getting help.

The call is: HELP

5.3 Including DISGCL Files

A DISGCL file can be included into another DISGCL file or DISGCL session with the statement

INCLUDE 'file[.gcl]'

Notes:

- Up to 8 include files can be nested.
- DISGCL searches the current working directory for the filename. If the search fails, DISGCL searches the directory defined by the environment variable GCL_PATH.

5.4 Listing Variables

The command LIST list all defined variables.

The call is: LIST [option] [v1, v2, ... vn]

option — can have the values -s and -u. If -s is used, only system variables are listed, if -u is used, only user-defined variables are listed.

v1, v2, .., vn — are the names of variables. If no variables are specified, all defined variables are listed.

5.5 Freeing Variables

The command FREE deallocates the space allocated by arrays and strings.

The call is: FREE v1, v2, ..., vn

v1, v2, .., vn are the names of variables.

5.6 The PRINT Command

The command PRINT prints the value of an expression on the screen.

The call is: PRINT expr1, expr2, ..., exprn

expr1, expr2, .., exprn are expressions.

5.7 Logging Commands

The command LOGON defines a file where interactive commands will be logged.

The call is: LOGON cfil

cfil is a string containing a filename.

Notes:

- Logging can be terminated with the command LOGOFF.
- A log file can be executed with the DISGCL command, or can be included into a DISGCL session with the INCLUDE statement.
- If the file cfil already exists, a new file version will be created.

5.8 Creating Arrays

The commands BYTE, CHAR, INT, SHORT, FLOAT, DOUBLE and COMPLEX create corresponding arrays and initialize them with zeros.

The call is: FLOAT list

list is a list of arrays with dimension specifications separated by commas.

Example: FLOAT A[10], B[5, 10] creates an array A with 10 elements and a matrix B with 5 rows and 10 columns.

Chapter 6

User-defined Subroutines and Functions

User-defined subroutines and functions can be used to break large DISGCL tasks into several parts.
Subroutines and functions offer some advantages:

- They use local variables and labels that are not visible to other parts of the DISGCL task.
- They make debugging easier. Once the subroutine or function is tested, it can be used in many other DISGCL runs.
- They eleminate repetition of code.

Each user-defined subroutine or function must be stored in a file with the extension '.gcl' where the name of the file and the name of the routine must be identical. The first statement in the file must be the identifier '%GCL'. The next non-comment statement in the file should be either the SUBROUTINE or the FUNCTION statement. Functions and subroutines are terminated with the first END statement that should be the last statement in the file.

6.1 Calling User-defined Subroutines

The CALL statement executes a user-defined subroutine. The syntax is:

CALL routine (list)

where

routine	is the name of the subroutine.
list	are the actual parameters of the subroutine separated by commas.

DISGCL adds the extension '.gcl' to the name of the subroutine and searches the current directory for the filename. If the search fails, DISGCL searches the directory defined by the environment variable GCL_PATH.

6.2 Calling User-defined Functions

A user-defined function is called in the same way as a DISLIN and DISGCL function. The syntax is:

v = function (list)

where

function is the name of the function.

list are the parameters of the function separated by commas.

The search order for functions is:

1. DISLIN functions.
2. DISGCL functions.
3. User-defined functions.

If the search for a DISLIN and DISGCL function fails, DISGCL adds the extension '.gcl' to the name of the function and searches the current directory for the filename. If the search also fails, DISGCL searches the directory defined by the environment variable GCL_PATH.

6.3 The SUBROUTINE Statement

The SUBROUTINE statement must be the first non-comment statement in a user-defined subroutine (after the identifier '%GCL'). It has the syntax:

SUBROUTINE routine (list)

where

routine is the name of the subroutine.

list are the formal parameters of the subroutine separated by commas.

6.4 The FUNCTION Statement

The FUNCTION statement must be the first non-comment statement in a user-defined function (after the identifier '%GCL'). It has the syntax:

FUNCTION function (list)

where

function is the name of the function.

list are the formal parameters of the function separated by commas.

6.5 The EXTERN Statement

The EXTERN statement can be used to access variables of the main DISGCL unit. These are variables that are defined outside of functions and subroutines. The syntax is:

EXTERN v1, v2, ..., vn

where v1, v2, ..., vn are the names of variables.

6.6 The RETURN Statement

The RETURN statement can be used to exit a subroutine and must be used in a function to return a value. The syntax is:

RETURN	for subroutines.
RETURN expr	for functions.

6.7 Parameters

Variables and expressions can be passed as parameters to user-defined functions and subroutines. Parameters in the CALL statement of a subroutine or in the function reference are called actual paramters. Formal parameters are the parameters declared in the SUBROUTINE or FUNCTION statement.

A formal parameter has the syntax:

[type:][key=]name
where

type	is an optional type declaration of the parameter. It can have the keywords BYTE, CHAR, SHORT, INT, FLOAT, DOUBLE, COMPLEX and STRING. If a type declaration is specified, DISGCL compares automatically the type of actual and formal parameters.
key	is an optional keyword for the parameter that identifies which parameter is being passed.
name	is the formal name of the parameter. The name of the parameter can have a dimension specification that means that the actual parameter must be an array.

Example:

SUBROUTINE MYSUB (FLOAT: A, INT: N, FLOAT: DATA=X[])

An actual parameter has the syntax:

[key=]expr
where

key	is an optional keyword for the parameter.
expr	is an expression.

Example:

CALL MYSUB (3., 10, DATA=XRAY)

Notes:

- The number of actual parameters can be lower than the number of formal parameters. The function ARGCNT() returns the total number of parameters passed to a subroutine or function. The function KEYCNT () returns the number of keyword parameters passed to a subroutine or function. The functions VARDEF (x), VARTYP (x), VARCNT (x) and VARDIM (x, n) can be used to analyse passed parameters (see Appendix A).
- If the number of actual parameters is greater than the number of formal parameters, or a keyword is used that is not defined in the formal parameter list, DISGCL displays a warning and ignores the routine call.

Chapter 7

Quickplots

This chapter presents quickplots that are collections of DISLIN routines to display data with one command.

The following rules are applied to quickplots:

- Quickplots call DISINI automatically if it is not called before. METAFL ('XWIN') will be used in quickplots if METAFL is not used before.

- On X Window terminals, there are no calls to ENDGRF and DISFIN in quickplots, they let DISLIN in level 2 or 3. If the variable %ERASE is set to 0, following quickplots will overwrite the graphics window without erasing the window.

- On other terminals such as VGA screens, quickplots are terminated with DISFIN to get back to the text mode.

Note:

All quickplots have corresponding widget interfaces that can be executed with the command

gcl quickplot

where quickplot is the name of a quickplot. The widget interfaces for quickplots expect data in the form of data files described in chapter 8, 'Data Files'.

7.1 The PLOT Command

The command PLOT makes a quickplot of two or more floatingpoint arrays.

The call is: PLOT xray1, yray1 [,xray2, yray2, ... , xrayn, yrayn]

xray1, yray1 are floatingpoint arrays.

Example:
```
x = falloc (100)
plot x, sin (x), x, cos (x)
```

7.2 The SCATTR Command

The command SCATTR makes a quickplot of two or more floatingpoint arrays where the points are marked with symbols.

The call is:	SCATTR xray1, yray1 [,xray2, yray2, ... , xrayn, yrayn]
xray1, yray1	are floatingpoint arrays.

7.3 The PLOT3 Command

The command PLOT3 makes a 3-D colour plot.

The call is:	PLOT3 xray, yray, zray
xray, yray, zray	are floatingpoint arrays containing X-, Y- and Z-coordinates.

7.4 The PLOT3R Command

The command PLOT3R makes a 3-D colour plot where the data are specified as rectangles.

The call is:	PLOT3R x1ray, y1ray, x2ray, y2ray, zray
x1ray, y1ray	are floatingpoint arrays containing X- and Y- coordinates of rectangle corners.
x2ray, y2ray	are floatingpoint arrays containing the opposite rectangle corners.
zray	is a floatingpoint array containing Z-coordinates.

7.5 The SURF3 Command

The command SURF3 makes a 3-D colour plot of a matrix. The columns of the matrix will be plotted as rows.

The call is:	SURF3 zmat [,xray [, yray]]
zmat	is a two-dimensional floatingpoint array with m rows and n columns.
xray	is a floatingpoint array with the dimension m. It will be used to position the rows of zmat. If xray is missing, an array with the values { 0.5, 1.5, ..., m - 0.5} will be used.
yray	is a floatingpoint array with the dimension n. It will be used to position the columns of zmat. If yray is missing, an array with the values { 0.5, 1.5, ..., n - 0.5} will be used.

7.6 The SURFACE Command

The command SURFACE makes a surface plot of a matrix.

The call is:	SURFACE zmat [,xray, yray]
zmat	is a two-dimensional floatingpoint array with nx rows and ny columns.

xray — is a floatingpoint array with the dimension nx. It will be used to position the rows of zmat. If xray is missing, an array with the values { 0., 1., ..., nx - 1.} will be used.

yray — is a floatingpoint array with the dimension ny. It will be used to position the columns of zmat. If yray is missing, an array with the values { 0., 1., ..., ny - 1.} will be used.

7.7 The SURSHADE Command

The command SURSHADE makes a shaded surface plot of a matrix.

The call is: SURSHADE zmat [,xray, yray]

zmat — is a two-dimensional floatingpoint array with nx rows and ny columns.

xray — is a floatingpoint array with the dimension nx. It will be used to position the rows of zmat. If xray is missing, an array with the values { 0., 1., ..., nx - 1.} will be used.

yray — is a floatingpoint array with the dimension ny. It will be used to position the columns of zmat. If yray is missing, an array with the values { 0., 1., ..., ny - 1.} will be used.

7.8 The CONTOUR Command

The command CONTOUR makes a contour plot of a matrix.

The call is: CONTOUR zmat [,xray, yray, zlvray]

or: CONTOUR zmat, zlvray

zmat — is a two-dimensional floatingpoint array with nx rows and ny columns.

xray — is a floatingpoint array with the dimension nx. It will be used to position the rows of zmat. If xray is missing, an array with the values { 0., 1., ..., nx - 1.} will be used.

yray — is a floatingpoint array with the dimension ny. It will be used to position the columns of zmat. If yray is missing, an array with the values { 0., 1., ..., ny - 1.} will be used.

zlvray — is a floatingpoint array containing the levels. If zlvray is missing, 10 levels between the minimum and maximum of zmat will be generated.

7.9 The CONSHADE Command

The command CONSHADE makes a shaded contour plot of a matrix.

The call is: CONSHADE zmat [,xray, yray, zlvray]

or: CONSHADE zmat, zlvray

zmat — is a two-dimensional floatingpoint array with nx rows and ny columns.

xray — is a floatingpoint array with the dimension nx. It will be used to position the rows of zmat. If xray is missing, an array with the values { 0., 1., ..., nx - 1.} will be used.

yray — is a floatingpoint array with the dimension ny. It will be used to position the columns of zmat. If yray is missing, an array with the values { 0., 1., ..., ny - 1.} will be used.

zlvray — is a floatingpoint array containing the levels. If zlvray is missing, 10 levels between the minimum and maximum of zmat will be generated.

7.10 Scaling of Quickplots

Normally, quickplots are scaled automatically in the range of the data. This behaviour can be changed if certain variables are defined.

The variables for the X-axis are:

a) If the system variables %XMIN and %XMAX are defined, the X-axis will be scaled automatically in the range %XMIN, %XMAX.

b) If the system variables %XMIN, %XMAX, %XOR and %XSTEP are defined, the scaling and labeling of the X-axis is completly defined by the user.

c) If the system variable %XAUTO is defined and set to 1, the variables %XMIN, %XMAX, %XOR and %XSTEP will be ignored and scaling will be done automatically in the range of the data.

Analog: Y-axis, Z-axis.

Note: For logarithmic scaling, the parameters must be exponents of base 10.

7.11 Quickplot Variables

There is a set of variables that can modify the appearance of quickplots. The corresponding DISLIN routines are given in parenthesis.

%X — defines the X-axis title (NAME).

%Y — defines the Y-axis title (NAME).

%Z — defines the Z-axis title (NAME).

%T1 — defines line 1 of the axis system title (TITLIN).

%T2 — defines line 2 of the axis system title (TITLIN).

%T3 — defines line 3 of the axis system title (TITLIN).

%T4 — defines line 4 of the axis system title (TITLIN).

%XTIC — sets the number of ticks for the X-axis (TICKS).

%YTIC — sets the number of ticks for the Y-axis (TICKS).

%ZTIC — sets the number of ticks for the Z-axis (TICKS).

%XDIG	sets the number of digits for the X-axis (LABDIG).
%YDIG	sets the number of digits for the Y-axis (LABDIG).
%ZDIG	sets the number of digits for the Z-axis (LABDIG).
%XSCL	defines the scaling of the X-axis (AXSSCL).
%YSCL	defines the scaling of the Y-axis (AXSSCL).
%ZSCL	defines the scaling of the Z-axis (AXSSCL).
%XLAB	defines the labels of the X-axis (LABELS).
%YLAB	defines the labels of the Y-axis (LABELS).
%ZLAB	defines the labels of the Z-axis (LABELS).
%H	defines the character size (HEIGHT).
%HNAME	defines the size of axis titles (HNAME).
%HTITLE	defines the size of the axis sytem title (HTITLE).
%XPOS	defines the X-Position of the axis system (AXSPOS).
%YPOS	defines the Y-Position of the axis system (AXSPOS).
%XLEN	defines the size of an axis system in X-direction (AXSLEN).
%YLEN	defines the size of an axis system in Y-direction (AXSLEN).
%ZLEN	defines the size of an axis system in Z-direction (AX3LEN).
%POLCRV	defines an interpolation method used by CURVE (POLCRV).
%INCMRK	defines line or symbol mode for CURVE (INCMRK).
%MARKER	selctes a symbol for CURVE (MARKER).
%HSYMBL	defines the size of symbols (HSYMBL).
%XRES	sets the width of points plotted by PLOT3 (SETRES).
%YRES	sets the height of points plotted by PLOT3 (SETRES).
%X3VIEW	sets the X-position of the viewpoint in absolut 3-D coordinates (VIEW3D).
%Y3VIEW	sets the Y-position of the viewpoint in absolut 3-D coordinates (VIEW3D).
%Z3VIEW	sets the Z-position of the viewpoint in absolut 3-D coordinates (VIEW3D).
%X3LEN	defines the X-axis length of the 3-D box (AXIS3D).
%Y3LEN	defines the Y-axis length of the 3-D box (AXIS3D).
%Z3LEN	defines the Z-axis length of the 3-D box (AXIS3D).
%VTITLE	defines vertical shifting for the axis system title (VKYTIT).
%CONSHD	selects an algorithm used for contour filling (SHDMOD).

Note: The variables can also be used, to initalize plotting parameters in DISINI.

Example:

```
%X = 'X-axis'
%Y = 'Y-axis'
xray = falloc (10)
plot xray, xray
```

Chapter 8

Data Files

This chapter describes data files that can be used to include data into DISGCL jobs. The format of data files is very simple and useful for most DISLIN plotting routines.

8.1 Syntax of Data Files

- A data file must begin with the keyword '%GCL-ASC'.
- Each line may contain up to 512 characters.
- Lines are allowed to carry trailing comment fields, following a double slash (//). Empty lines are also be interpreted as comment lines.
- A data file can contain an optional header beginning with the keyword %HEADER. Variables can be defined in the header in the form name = value, where value is an integer, a floatingpoint number or a string. Strings must be enclosed in a pair of either apostrophes or quotation marks.
- A data field begins with the keyword %DATA. The first non comment line after %DATA must contain the number of columns of the data field. The following lines give the data separated by blanks or commas. Data can be specified as integer or floatingpoint numbers where floatingpoint numbers can contain an exponent part.
- Multiple data blocks can be given in one data file.

8.2 Data File Routines

The following routines handle data files.

D A T F I L

The routine DATFIL opens a file for data input.

The call is:	ISTAT = DATFIL (CFIL)
CFIL	is the name of the data file. The default extension is '.gdf' (Graphics Data File). DATFIL returns -1 if it fails, or zero.
Note:	DISGCL searches the current working directory for the data file. If the search fails, DISGCL searches the directory defined by the environment variable GCL_DATA.

D A T C L S

The routine DATCLS closes the current data file. It returns -1 if it fails, or zero.

The call is: ISTAT = DATCLS ()

D A T H D R

The routine DATHDR prints the header of a data file on the screen.

The call is: ISTAT = DATHDR ()

ISTAT is the returned status of DATHDR and can have the values 0 and -1. On error, DATHDR returns -1.

D A T V A R

The routine DATVAR returns the value of a variable defined in the header of a data file.

The call is: V = DATVAR (CNAME)

CNAME is the name of the variable.

V is the returned value of the variable. The first occurrence of the variable is returned.

D A T C N T

The routine DATCNT returns the number of data of the current data block in a data file.

The call is: N = DATCNT (COPT)

COPT is a string with the values

- = 'ROWS' means the number of rows in the current data block.
- = 'COLUMNS' means the number of columns in the current data block.
- = 'FULL' means the number of data in the current data block.
- = 'BLOCKS' means the number of blocks in the data file.

D A T R A Y

The routine DATRAY creates an array containing a column of the current data block in a data file.

The call is: XRAY = DATRAY (NCOLUMN)

NCOLUMN defines the column of the data file.

XRAY is a returned floatingpopint array.

D A T M A T

The routine DATMAT creates an array containing the whole current data block of the data file.

The call is: XMAT = DATMAT ()

XMAT is a returned twodimensional floatingpopint array.

D A T B L K

The routine DATBLK sets the current data block of a data file.

The call is: ISTAT = DATBLK (NBLOCK)

NBLOCK specifies the current data block. DATBLK returns 0 if the data block NBLOCK is defined, and -1 if NBLOCK is not defined in the current data file.

8.3 Example

```
%GCL-ASC
%DATA
3          // Number of columns
//   x      sin(x)   cos(x)
0.000000 0.000000 1.000000
14.545455 0.251148 0.967949
29.090910 0.486197 0.873849
43.636364 0.690079 0.723734
58.181820 0.849725 0.527225
72.727272 0.954902 0.296920
87.272728 0.998867 0.047582
101.818184 0.978802 -0.204807
116.363640 0.895994 -0.444067
130.909088 0.755750 -0.654861
145.454544 0.567060 -0.823677
160.000000 0.342020 -0.939693
174.545456 0.095056 -0.995472
189.090912 -0.158001 -0.987439
203.636368 -0.400931 -0.916108
218.181824 -0.618159 -0.786053
232.727280 -0.795762 -0.605610
247.272720 -0.922354 -0.386345
261.818176 -0.989821 -0.142315
276.363647 -0.993838 0.110838
290.909088 -0.934148 0.356886
305.454559 -0.814576 0.580057
320.000000 -0.642788 0.766044
334.545441 -0.429795 0.902926
349.090912 -0.189251 0.981929
360.000000 -0.000000 1.000000
```

Chapter 9

Input and Output

This chapter describes functions that write formatted data to the screen and access disk files.

9.1 Formatted Output with PRINTF

The function PRINTF translates internal values and prints them on the screen.

The call is:	ISTAT = PRINTF (CFMT, ARG1, ARG2, ..., ARGn)
CFMT	is a format string.
ARG1,..,ARGn	are optional arguments.
ISTAT	is the returned status of PRINTF and can have the values 0 and -1. On error, PRINTF returns -1.

The format string can contain ordinary characters and format specifications. Ordinary characters are printed as text while format specifications define how the arguments of PRINTF are formatted. A format specification begins with a % and ends with a conversion character. Between the % and the conversion character there may be the following parts:

- A minus sign, which specifies left adjustment.
- A number that specifies the minimum field length.
- A perid, which separates the field width from the precision.
- A number that specifies the precision. It defines the number of printed charcters from a string, or the number of digits after the decimal point of floating point values.

Conversion characters are shown in the following table:

Character	Meaning
%c	prints a character.
%d	prints an integer.
%i	prints an integer (same as %d).
%s	prints a string.
%f	prints a floatingpoint number.
%e	prints a floatingpoint number in e format.
%x	prints an integer in hexadecimal format.
%o	prints an integer in octal format.
\n	inserts a newline.

Figure 9.1: Conversion Characters

Examples:

The first example shows the effect of some format specifications for printing the string 'This is a string.' (17 characters).

```
%GCL

s = 'This is a string.'
printf (':%s:      \n', s)
printf (':%15s:    \n', s)
printf (':%17s:    \n', s)
printf (':%20s:    \n', s)
printf (':%-20s:   \n', s)
printf (':%17.13s: \n', s)
printf (':%-17.13s:\n', s)
```

The output of the DISGCL script is:

```
:This is a string.:
:This is a string.:
:This is a string.:
:   This is a string.:
:This is a string.   :
:    This is a str:
:This is a str    :
```

The next example shows the effect of some format specifications for printing the integer 254:

```
%GCL

i = 254
printf (':%d:    \n', i)
printf (':%8d:   \n', i)
printf (':%08d:  \n', i)
printf (':%-8d:  \n', i)
printf (':%-8.2d:\n', i)
printf (':%x:    \n', i)
printf (':%08X:  \n', i)
printf (':%o:    \n', i)
```

The output is:

```
:254:
:     254:
:00000254:
:254     :
:254     :
:fe:
:000000FE:
:376:
```

The next example shows the effect of some format specifications for printing the floatingpoint number 123.456:

```
%GCL

x = 123.456
printf (':%f:      \n', x)
printf (':%15f:    \n', x)
printf (':%-15f:   \n', x)
printf (':%-15.2f: \n', x)
printf (':%e:      \n', x)
printf (':%-15E:   \n', x)
printf (':%-15.2E: \n', x)
```

The output is:

```
:123.456000:
:     123.456000:
:123.456000     :
:123.46         :
:1.234560e+02:
:1.234560E+02   :
:1.23E+02       :
```

9.2 Formatted Output with SPRINTF

The function SPRINTF does the same conversions as PRINTF does, but stores the output in a character array.

The call is:	ISTAT = SPRINTF (CRAY, CFMT, ARG1, ARG2, ..., ARGn)
CRAY	is a character array that must be created with the CHAR command. CRAY must be big enough to hold the result and the string terminator.
CFMT	is a format string.
ARG1,..,ARGn	are optional arguments.
ISTAT	is the returned status of SPRINTF and can have the values 0 and -1. On error, SPRINTF returns -1.

9.3 Formatted Input with SCANF

The function SCANF is an analog function to PRINTF for formatted input from the keyboard. It uses the format specifications listed in table 9.1.

The call is:	ISTAT = SCANF (NU, CFMT, ARG1, ARG2, ..., ARGn)
NU	is a file unit.
CFMT	is a format string.
ARG1,..,ARGn	are optional arguments where the arguments must be variables. If a string should be read, the corresponding variable must be created with the DISGCL command CHAR and big enough to hold the string and a trailing string terminator.
ISTAT	is the returned status of SCANF and can have the values 0 and -1. On error, SCANF returns -1.

Note:

SCANF cannot be used in the interactive mode of DISGCL.

9.4 Formatted Input with SSCANF

The function SSCANF does the same conversions as SCANF does, but reads the input from a string.

The call is:	ISTAT = SSCANF (CSTR, CFMT, ARG1, ARG2, ..., ARGn)
CSTR	is a string where the input characters are taken from.
CFMT	is a format string.
ARG1,..,ARGn	are optional arguments (see SCANF).
ISTAT	is the returned status of SSCANF and can have the values 0 and -1. On error, SSCANF returns -1.

9.5 File Access

The following functions deal with operations on files.

F O P E N

The routine FOPEN opens a file and returns a file unit.

The call is: NU = FOPEN (CFIL, CMOD)

CFIL is the name of a file. If CFIL = 'CON:', the console is opened for file I/O. If DISGCL is used in interactive mode, the console can only be opened for output (CMOD = 'w').

CMOD is a string that defines the mode for file access. The modes are listed in table 9.2.

NU is the returned file unit, or -1 if an error occurs.

The following table shows the allowed file modes for FOPEN:

Mode	Meaning
'r'	opens a text file for reading. The file must exist, or an error message is printed.
'w'	opens a text file for writing. The contents of an existing file will be overwritten.
'a'	opens a text file for appending.
'rb'	opens a file for binary reading.
'wb'	opens a file for binary writing.
'ab'	opens a file for binary appending.

Figure 9.2: File Modes for FOPEN

F C L O S E

The function FCLOSE closes a file.

The call is: ISTAT = FCLOSE (NU)

NU is a file unit.

ISTAT is the returned status of FCLOSE and can have the values 0 and -1. On error, FCLOSE returns -1.

R E M O V E

The function REMOVE deletes a file.

The call is: ISTAT = REMOVE (CFIL)

CFIL	is the name of a file.
ISTAT	is the returned status of REMOVE and can have the values 0 and -1. On error, REMOVE returns -1.

R E N A M E

The function RENAME changes the name of a file.

The call is:	ISTAT = RENAME (COLD, CNEW)
COLD	is the old name of the file.
CNEW	is the new name of the file.
ISTAT	is the returned status of RENAME and can have the values 0 and -1. On error, RENAME returns -1.

F T E L L

The function FTELL returns the current file position.

The call is:	NPOS = FTELL (NU)
NU	is a file unit.
NPOS	is the returned file position, or -1 on error.

F S E E K

The function FSEEK defines the current file position.

The call is:	ISTAT = FSEEK (NU, NPOS)
NU	is a file unit.
NPOS	is the new file position.
ISTAT	is the returned status of FSEEK and can have the values 0 and -1. On error, FSEEK returns -1.

F F L U S H

The function FFLUSH flushes any output buffers. For input, FFLUSH has no effect.

The call is:	ISTAT = FFLUSH (NU)
NU	is a file unit.
ISTAT	is the returned status of FFLUSH and can have the values 0 and -1. On error, FFLUSH returns -1.

R E W I N D

The function REWIND sets the current file position to the beginning of the file.

The call is:	ISTAT = REWIND (NU)
NU	is a file unit.
ISTAT	is the returned status of REWIND and can have the values 0 and -1. On error, REWIND returns -1.

9.6 Formatted Output to Files

FPRINTF

The function FPRINTF does the same conversions as PRINTF does, but writes the output to a file.

The call is:	ISTAT = FPRINTF (NU, CFMT, ARG1, ARG2, ..., ARGn)
NU	is a file unit.
CFMT	is a format string.
ARG1,..,ARGn	are optional arguments.
ISTAT	is the returned status of FPRINTF and can have the values 0 and -1. On error, FPRINTF returns -1.

9.7 Formatted Input from Files

FSCANF

The function FSCANF is an analog function to FPRINTF for formatted input. It uses the format specifications listed in table 9.1.

The call is:	ISTAT = FSCANF (NU, CFMT, ARG1, ARG2, ..., ARGn)
NU	is a file unit.
CFMT	is a format string.
ARG1,..,ARGn	are optional arguments where the arguments must be variables. If a string should be read from the file, the corresponding variable must be created with the DISGCL command CHAR and big enough to hold the string and the trailing string terminator.
ISTAT	is the returned status of FSCANF and can have the values 0 and -1. On error, FSCANF returns -1.

9.8 Text Input and Output Functions

FGETC

The function FGETC returns the next character from a file.

The call is:	N = FGETC (NU)
NU	is a file unit.
N	is the ASCII code of the returned character, or -1 if end of file or error occurs.

FGETS

The function FGETS reads at most N-1 characters into a character array, stopping if a newline is encountered. The newline character is included in the character array.

The call is:	ISTAT = FGETS (CRAY, N, NU)
CRAY	is a character array filled by FGETS. CRAY must be created with the DISGCL command CHAR (i.e. CHAR CRAY[N]).
N	defines the number of characters to be read. At most N-1 characters will be read.
NU	is a file unit.
ISTAT	is the returned status of FGETS and can have the values 0 and -1. FGETS returns -1 if end of file or error occurs.

Note:

The function GETS (CRAY) reads the next input line into the character array CRAY; it replaces the terminating newline with '\0'.

FPUTC

The function FPUTC writes a character to a file.

The call is:	ISTAT = FPUTC (N, NU)
N	is the ASCII code of a character that should be written to a file.
NU	is a file unit.
ISTAT	is the returned status of FPUTC and can have the values 0 and -1. On error, FPUTC returns -1.

FPUTS

The function FPUTS writes a string to a file.

The call is:	ISTAT = FPUTS (CSTR, NU)
CSTR	is a string that should be written to the file. A newline character is not inserted after the string.
NU	is a file unit.
ISTAT	is the returned status of FPUTS and can have the values 0 and -1. On error, FPUTS returns -1.

Note:

The function PUTS (CSTR) prints the string CSTR and a newline on the console.

9.9 Binary Input and Output Functions

FREAD

The function FREAD reads binary data from a file.

The call is:	NRET = FREAD (VRAY, N, NU)
VRAY	is an array where the binary data should be filled in.
N	is the number of array elements that should be filled with data.
NU	is a file unit.
NRET	is the number of elements read, or -1 if end of file or error occurs.

FWRITE

The function FWRITE writes binary data to a file.

The call is:	NRET = FWRITE (VRAY, N, NU)
VRAY	is an array containing the data that should be written to the file.
N	is the number of array elements that should be written to the file.
NU	is a file unit.
NRET	is the number of elements written, or -1 if error occurs.

9.10 Example

The following example copies a text files and converts it to uppercase letters.

```
%GCL
// Copies a file and converts it to uppercase letters

char cr[100], cinp[40], cout[40]

printf ('Inputfile: ')
scanf  ('%s', cinp)

printf ('Outputfile: ')
scanf  ('%s', cout)

inp =  fopen (cinp, 'r')
if (inp == -1) exit
out =  fopen (cout, 'w')
if (out == -1) exit

i = fgets (cr, 100, inp)
while (i != -1)
  s = strupr (cr)
```

```
    fputs (s, out)
    i = fgets (cr, 100, inp)
  end while

  fclose (inp)
  fclose (out)
```

The example can be stored in a file and executed with the command:

gcl filename

Appendix A

Intrinsic Functions

This appendix is a summary of the DISGCL intrinsic functions.

A.1 Mathematical Functions

Function	Meaning
ABS (x)	absolute value $\mid$ x $\mid$.
ACOS (x)	cos (x)$^{-1}$ in the range [0, π], -1 $\leq$ x $\leq$ 1.
ASIN (x)	sin (x)$^{-1}$ in the range [0, π], -1 $\leq$ x $\leq$ 1.
ATAN (x)	tan (x)$^{-1}$ in the range [-π/2, π/2].
ATAN2 (y, x)	tan (y/x)$^{-1}$ in the range [-π, π].
CEIL (x)	smallest integer not less than x, as a double.
COS (x)	cosine of x.
COSH (x)	hyperbolic cosine of x.
EXP (x)	exponential function e^{x}.
FLOOR (x)	largest integer not greater than x, as a double.
FMOD (x, y)	floating-point remainder of x/y, with the same sign as x. FMOD returns zero if y is zero.
LOG (x)	natural logarithm ln (x), x $>$ 0.
LOG10 (x)	base 10 logarithm, x $>$ 0.
SIN (x)	sine of x.
SINH (x)	hyperbolic sine of x.
SQRT (x)	$\sqrt{x}$, x $\geq$ 0.
TAN (x)	tangent of x.
TANH (x)	hyperbolic tangent of x.

Figure A.1: Mathematical Functions

Note:

Normally, the parameter x in a mathematical function can be a scalar or an array expression, and may be real or complex. If x is a scalar expression, the returned value of a mathematical function has the type DOUBLE or COMPLEX. If x is an array expression of types FLOAT, DOUBLE or COMPLEX, the returned value is also a FLOAT, DOUBLE or COMPLEX array. If x is an array expression of type INT, the returned value is a FLOAT array.

A.2 Type Conversion Functions

Function	Meaning
ATONUM (s)	converts a string to a number.
BYTE (x)	converts x to BYTE.
CHAR (x)	converts x to CHAR.
COMPLEX (x, y)	converts to COMPLEX. x is converted to the real, y is converted to the imaginary part.
DOUBLE (x)	converts x to DOUBLE.
FLOAT (x)	converts x to FLOAT. If x is a complex number, FLOAT returns the real part of x.
INT (x)	converts x to INT.
NUMTOA (x, ndig)	converts a number x to a string where ndig is the number of decimal places plotted after the decimal point.
SHORT (x)	converts x to SHORT.
STRING (x)	converts x to STRING.

Figure A.2: Type Conversion Functions

A.3 Complex Functions

Function	Meaning
ARG (z)	returns the phase of z in radians.
CONJG (z)	returns the conjugate of the complex value z.
IMAG (z)	returns the imaginary part of z.

Figure A.3: Complex Functions

A.4 Array Functions

Function	Meaning
CATARR (a, b)	Concatenates two arrays.
MAXARR (a)	Returns the maximum of an array a.
MINARR (a)	Returns the minimum of an array a.
SUBARR (a, i1, n)	Creates a subarry from the array a. i1 is the first index, n the number of elements of a.
SUBMAT (a, idx)	Creates a submatrix from certain columns of the two-dimensional array a. idx is an integer array containing column numbers in the range 0 to n-1 where n is the number of columns in a.
TRPMAT (a)	Exchanges rows and columns of a matrix a.

Figure A.4: Array Functions

A.5 Variable and Parameter Functions

Function	Meaning
ARGCNT ()	returns the total number of parameters passed to a user-defined subroutine or function.
KEYCNT ()	returns the number of keyword parameters passed to a user-defined subroutine or function.
VARCNT (v)	returns the number of elements of a variable v, or -1 if v is not defined.
VARDEF (v)	returns 1 if the variable v is defined, and 0 if v is not defined.
VARDIM (v, n)	returns the n-th dimension of a variable v. If the dimension is not defined, VARDIM returns zero.
VARTYP (v)	returns the data type of the variable v, or the string 'UNDEF' if v is not defined. The returned type can have the value 'BYTE', 'INT', 'FLOAT' or 'DOUBLE'.

Figure A.5: Variable and Parameter Functions

A.6 Data File Functions

Function	Meaning
DATBLK (n)	defines the current data block. It returns -1 if the data block n is not defined.
DATCNT (copt)	returns the number of data in the current data block. copt can have the values 'ROWS', 'COLUMNS', 'FULL' and 'BLOCKS'.
DATFIL (s)	defines the data file. It returns -1 if it fails.
DATHDR ()	prints the header of a data file on the screen. DATHDR returns -1 if it fails.
DATMAT ()	returns an array containing the whole current data block.
DATRAY (ncolumn)	returns an array containing a column of the current data block.
DATVAR (cname)	returns the value of a variable defined in the header of a data file.

Figure A.6: Data File Functions

A.7 Memory Allocating Functions

Function	Meaning
DALLOC (n)	creates a double array and initializes it with {0., 1., ..., n-1.}.
FALLOC (n)	creates a floatingpoint array and initializes it with {0., 1., ..., n-1.}.
IALLOC (n)	creates an integer array and initializes it with {0, 1, ..., n-1}.

Figure A.7: Memory Allocating Functions

A.8 String Functions

Function	Meaning
STRLEN (s)	returns the length of a string.
STRLWR (s)	returns a string converted to lowercase letters.
STRREP (s, n)	returns n copies of the string s.
STRSTR (s1, s2)	returns the first occurence of string s2 in string s1, or -1 if not present.
STRUPR (s)	returns a string converted to uppercase letters.
SUBSTR (s, i1, n)	returns a substring of s where i1 is the starting index and n the number of characters of s.
TRMLEN (s)	returns the length of a string without trailing blanks.

Figure A.8: String Functions

A.9 File Functions

Function	Meaning
FCLOSE (nu)	closes the file with the unit nu. It returns -1 on error.
FFLUSH (nu)	flushes any output buffers.
FOPEN (cfile, cmode)	opens a file and returns a file unit, or -1 if the open fails.
FSEEK (nu, npos)	sets the current file position. It returns -1 on error.
FTELL (nu)	returns the current file position, or -1 on error.
INQUIRE (cfile)	tests if a file exists. It returns 0 on error, otherwise 1.
REMOVE (cfile)	deletes a file. It returns -1 if the attempt fails.
RENAME (cold, cnew)	changes the name of a file. It returns -1 if the attempt fails.
REWIND (nu)	rewinds the file with the unit nu. It returns -1 if any errors occured.

Figure A.9: File Functions

A.10 Input and Output Functions

Function	Meaning
FGETC (nu)	returns the next character from a file. FGETC returns -1 if end of file or error occurs.
FGETS (cbuf, n, nu)	reads at most n-1 characters into the array cbuf, stopping if a newline is encountered; The newline is included in the array. FGETS returns -1 if end of file or error occurs.
FPRINTF (nu, s, vlist)	writes formatted output to a file connected to the unit nu.
FPUTC (i, nu)	writes the character with the ASCII value i to a file. FPUTC returns 0 if successful, or -1 if an error occurs.
FPUTS (cbuf, nu)	writes the string cbuf to a file. FPUTS returns -1 if an error occurs.
FREAD (a, n, nu)	reads from a file n elements into the array a. FREAD returns the number of elements read, or -1 if an error occurs.
FSCANF (nu, s, vlist)	reads formatted input from a file connected to the unit nu.
FWRITE (a, n, nu)	writes from the array a n elements to the file with the unit nu. FWRITE returns the number of elements written, or -1 if an error occurs.
GETS (cbuf)	reads the next input line into the character array cbuf. The newline not is included in the array. GETS returns -1 if an error occurs.
PUTS (cbuf)	prints the string cbuf on the screen. A newline is printed after the string. PUTS returns -1 if an error occurs.
PRINTF (s , vlist)	writes formatted output to the terminal.
SCANF (s, vlist)	reads formatted input from the console.
SPRINTF (cr, s, vlist)	writes formatted output to a character array.
SSCANF (s1, s2, vlist)	reads formatted input from a string.

Figure A.10: Input/Output Functions

A.11 System Functions

Function	Meaning
GETENV (cenv)	returns the environment string associated with cenv, or blank if no string exists.
SYSTEM (cmd)	executes the system command cmd where cmd is a character variable. SYSTEM returns -1 if it fails.

Figure A.11: System Functions

A.12 Time Functions

Function	Meaning
CLOCK ()	returns the processor time in seconds used by the DISGCL session since the last call of CLOCK. At the first time, CLOCK returns 0.
DATE ()	returns a string containing the date in the format dd.mm.yy
TIME ()	returns a string containing the time in the format hh:mm:ss.

Figure A.12: Time Functions

Appendix B

Short Description of DISLIN Routines

This appendix presents a short description of all DISLIN routines that can be called from DISGCL. A complete description of the routines can be found in the DISLIN manual or via the online help of DISGCL. For parameters, the following conventions are used:

- integer variables begin with the character N or I
- strings begin with the character C
- other variables can be integers or floatingpoint numbers.
- arrays end with the keyword 'ray'. Normally, INT and FLOAT arrays must be passed to DISLIN routines.

B.1 Initialization and Introductory Routines

Routine	Meaning
CGMBGD (xr, xg, xb)	defines the background colour for CGM files.
CGMPIC (cstr)	sets the picture ID for CGM files.
DISINI ()	initializes DISLIN.
ERASE ()	clears the screen.
ERRDEV (cdev)	defines the error device.
ERRFIL (cfil)	sets the name of the error file.
FILBOX (nx, ny, nw, nh)	defines the position and size of included metafiles.
HWORIG (nx, ny)	defines the origin of the PostScript hardware page.
HWPAGE (nw, nw)	defines the size of the PostScript hardware page.
INCFIL (cfil)	includes metafiles into a graphics.
METAFL (cfmt)	defines the plotfile format.
NEWPAG ()	creates a new page.
ORIGIN (nx, ny)	defines the origin.
PAGE (nw, nh)	sets the page size.
PAGFLL (iclr)	fills the page with a colour.
PAGERA ()	plots a page border.
PAGHDR (c1, c2, iopt, idir)	plots a page header.
PAGMOD (copt)	selects a page rotation.
SCLFAC (x)	defines a scaling factor for the entire plot.
SCLMOD (copt)	defines a scaling mode.

Routine	Meaning
SCRMOD (copt)	swaps back- and foreground colours.
SETFIL (cfil)	sets the plotfile name.
SETPAG (copt)	selects a predefined page format.
SETXID (id, copt)	defines an external X Window or pixmap.
SYMFIL (cdev, cstat)	sends a plotfile to a device.
UNIT (nu)	defines the logical unit for messages.

Figure B.1: Initialization and Introductory Routines

B.2 Termination and Parameter Resetting

Routine	Meaning
DISFIN ()	terminates DISLIN.
ENDGRF ()	terminates an axis system and sets the level to 1.
RESET (copt)	resets parameters to default values.

Figure B.2: Termination and Parameter Resetting

B.3 Plotting Text and Numbers

Routine	Meaning
ANGLE (n)	defines the character angle.
CHAANG (x)	defines an inclination angle for characters.
CHASPC (x)	affects character spacing.
CHAWTH (x)	affects the width of characters.
FIXSPC (x)	sets a constant character width.
FRMESS (nfrm)	defines the thickness of text frames.
HEIGHT (n)	defines the character height.
MESSAG (cstr, nx, ny)	plots text.
MIXALF ()	enables control signs in character strings for plotting indices and exponents.
NEWMIX ()	defines an alternate set of control characters for plotting indices and exponents.
n = NLMESS (cstr)	returns the length of character strings in plot coordinates.
NUMBER (x, ndig, nx, ny)	plots floating-point numbers.
NUMFMT (copt)	determines the format of numbers.
NUMODE (c1, c2, c3, c4)	modifies the appearance of numbers.
RLMESS (cstr, x, y)	plots text where the position is specified in user coordinates.

Routine	Meaning
RLNUMB (x, ndig, xp, yp)	plots numbers where the position is specified in user coordinates.
SETBAS (xfac)	determines the position of indices and exponents.
SETEXP (xfac)	determines the character height of indices and exponents.
SETMIX (char, cmix)	defines global control signs for plotting indices and exponents.
TXTJUS (copt)	defines the alignment of text and numbers.

Figure B.3: Plotting Text and Numbers

B.4 Fonts

Routine	Meaning
BASALF (calph)	defines the base alphabet.
COMPLX ()	sets a complex font.
DUPLX ()	sets a double-stroke font.
DISALF ()	sets the default font.
EUSHFT (cnat, char)	defines a shift character for special European characters.
GOTHIC ()	sets a gothic font.
HELVE ()	sets a shaded font.
HELVES ()	sets a shaded font with small characters.
HWFONT ()	sets a standard hardware font.
PSFONT (cfont)	sets a PostScript font.
SERIF ()	sets a complex shaded font.
SIMPLX ()	sets a single-stroke font.
SMXALF (calph, c1, c2, n)	defines shift characters for alternate alphabets.
TRIPLX ()	sets a triple-stroke font.
WINFNT (cfont)	sets a TruType font.

Figure B.4: Fonts

B.5 Symbols

Routine	Meaning
HSYMBL (n)	defines the height of symbols.
RLSYMB (nsym, x, y)	plots symbols for user coordinates.
SYMBOL (nsym, nx, ny)	plots symbols.
SYMROT (xang)	defines a rotation angle for symbols.

Figure B.5: Symbols

B.6 Axis Systems

Routine	Meaning
AX2GRF ()	suppresses the plotting of the upper X- and the left Y-axis.
AX3LEN (nxl, nyl, nzl)	defines axis lengths for a coloured 3-D axis system.
AXGIT ()	plots the lines X = 0 and Y = 0.
AXSBGD (iclr)	defines the background colour.
AXSLEN (nxl, nyl)	defines axis lengths for a 2-D axis system.
AXSORG (nx, ny)	determines the position of a crossed axis system.
AXSPOS (nxp, nyp)	determines the position of axis systems.
AXSTYP (ctype)	select rectangular or crossed axis systems.
BOX2D ()	plots a border around an axis system.
CENTER ()	centres axis systems.
CROSS ()	plots the lines X = 0 and Y = 0 and marks them with ticks.
ENDGRF ()	terminates an axis system.
FRAME (nfrm)	defines the frame thickness of axis systems.
GRACE (ngrace)	affects the clipping margin of axis systems.
GRAF (xa, xe, xor, xstp, ya, ye, yor, ystp)	plots a two-dimensional axis system.
GRAF3 (xa, xe, xor, xstp, ya, ye, yor, ystp, za, ze, zor, zstp)	plots an axis system for colour graphics.
GRDPOL (nx, ny)	plots a polar grid.
GRID (nx, ny)	overlays a grid on an axis system.
NOCLIP ()	suppresses clipping of user coordinates.
NOGRAF ()	suppresses the plotting of an axis system.
SETGRF (c1, c2, c3, c4)	suppresses parts of an axis system.
SETSCL (xray, n, cax)	sets automatic scaling.
TITLE ()	plots a title over an axis system.
XAXGIT ()	plots the line Y = 0.
XCROSS ()	plots the line Y = 0 and marks it with ticks.
YAXGIT ()	plots the line X = 0.
YCROSS ()	plots the line X = 0 and marks it with ticks.

Figure B.6: Axis Systems

B.7 Secondary Axes

Routine	Meaning
XAXIS (xa, xe, xor, xstp, nl, cstr, it, nx, ny)	plots a linear X-axis.
XAXLG (xa, xe, xor, xstp, nl, cstr, it, nx, ny)	plots a logarithmic X-axis.
YAXIS (ya, ye, yor, ystp, nl, cstr, it, nx, ny)	plots a linear Y-axis.
YAXLG (ya, ye, yor, ystp nl, cstr, it, nx, ny)	plots a logarithmic Y-axis.
ZAXIS (za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a linearly scaled colour bar.
ZAXLG (za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a logarithmically scaled colour bar.

Figure B.7: Secondary Axes

B.8 Modification of Axes

Routine	Meaning
AXCLRS (nclr, copt, cax)	defines colours for axis elements.
AXENDS (copt, cax)	suppresses certain labels.
AXSSCL (copt, cax)	defines the axis scaling.
HNAME (nh)	defines the character height of axis names.
INTAX ()	defines integer numbering for all axes.
LABDIG (ndig, cax)	sets the number of decimal places for labels.
LABDIS (ndis, cax)	sets the distance between labels and ticks.
LABELS (copt, cax)	selects labels.
LABJUS (copt, cax)	defines the alignment of axis labels.
LABMOD (ckey, cval, cax)	modifies date labels.
LABPOS (copt, cax)	determines the position of labels.
LABTYP (copt, cax)	defines vertical or horizontal labels.
LOGTIC (copt)	modifies the appearance of logarithmic ticks.
MYLAB (cstr, itic, cax)	sets user-defined labels.
NAMDIS (ndis, cax)	sets the distance between axis names and labels.
NAME (cstr, cax)	defines axis titles.
NAMJUS (copt, cax)	defines the alignment of axis titles.
NOLINE (cax)	suppresses the plotting of axis lines.
RGTLAB ()	right-justifies labels.
RVYNAM ()	defines an angle for Y-axis names.
TICKS (ntics, cax)	sets the number of ticks.

Routine	Meaning
TICLEN (nmaj, nmin)	sets the length of ticks.
TICMOD (copt, cax)	modifies the plotting of ticks at calendar axes.
TICPOS (copt, cax)	determines the position of ticks.
TIMOPT ()	modifies time labels.

Figure B.8: Modification of Axes

B.9 Axis System Titles

Routine	Meaning
HTITLE (nh)	defines the character height of titles.
LFTTIT ()	left-justifies title lines.
LINESP (xfac)	defines line spacing.
TITJUS (copt)	defines the alignment of titles.
TITLE ()	plots axis system titles.
TITLIN (cstr, ilin)	defines text lines for titles.
TITPOS (copt)	defines the position of titles.
VKYTIT (nshift)	shifts titles in the vertical direction.

Figure B.9: System Titles

B.10 Plotting Data Points

Routine	Meaning
BARS (xray, y1ray, y2ray, n)	plots a bar graph.
CHNATT ()	changes curve attributes.
CHNCRV (copt)	defines attributes changed automatically by CURVE.
COLOR (color)	defines the colour used for text and lines.
CRVMAT (zmat, n, m, ixpts, iypts)	plots a coloured surface.
CURVE (xray, yray, n)	plots curves.
CURVE3 (xray, yray, zray, n)	plots coloured rectangles.
CURVX3 (xray, y, zray, n)	plots rows of coloured rectangles.
CURVY3 (x, yray, zray, n)	plots columns of coloured rectangles.
ERRBAR (xray, yray, e1ray, e2ray, n)	plots error bars.
FIELD (x1ray, y1ray, x2ray, y2ray, n, ivec)	plots a vector field.
GAPCRV (xgap)	defines gaps plotted by CURVE.
INCCRV (ncrv)	defines the number of curves plotted with equal attributes.

Routine	Meaning
INCMRK (nmrk)	selects symbols or lines for CURVE.
MARKER (nsym)	sets the symbols plotted by CURVE.
NOCHEK ()	suppresses listing of data points that lie outside of the axis scaling.
PIEGRF (cbuf, nlin, xray, n)	plots a pie chart.
POLCRV (copt)	defines the interpolation method used by CURVE.
RESATT ()	resets curve attributes.
SETRES (nx, ny)	sets the size of coloured rectangles.
SHDCRV (x1ray, y1ray, n1, x2ray, y2ray, n2)	plots shaded areas between curves.
SPLMOD (ngrad, npts)	modifies spline interpolation.
THKCRV (nthk)	defines the thickness of curves.

Figure B.10: Plotting Data Points

B.11 Legends

Routine	Meaning
FRAME (nfrm)	sets the frame thickness of legends.
LEGEND (cbuf, ncor)	plots legends.
LEGINI (cbuf, nlin, nmaxln)	initializes legends.
LEGLIN (cbuf, cstr, ilin)	defines text for legend lines.
LEGOPT (xf1, xf2, xf3)	modifies the appearance of legends.
LEGPAT (ityp, ithk, isym, iclr, ipat, ilin)	stores curve attributes.
LEGPOS (nxp, nyp)	determines the position of legends.
LEGTIT (ctitle)	defines the legend title.
LINESP (xfac)	affects line spacing.
MIXLEG ()	enables multiple text lines in legends.
nxl = NXLEGN (cbuf)	returns the width of legends in plot coordinates.
nyl = NYLEGN (cbuf)	returns the height of legends in plot coordinates.

Figure B.11: Legends

B.12 Line Styles and Shading Patterns

Routine	Meaning
CHNDOT ()	sets a dotted-dashed line style.
CHNDSH ()	sets a dashed-dotted line style.
COLOR (color)	sets a colour.
DASH ()	sets a dashed line style.
DASHL ()	sets a long-dashed line style.
DASHM ()	sets a medium-dashed line style.
DOT ()	sets a dotted line style.
DOTL ()	sets a long-dotted line style.
LINTYP (itype)	defines a line style.
LINWID (nwidth)	sets the line width.
LNCAP (copt)	sets the line cap parameter.
LNJOIN (copt)	sets the line join parameter.
LNMLT (xfac)	sets the miter limit parameter.
MYLINE (nray, n)	sets a user-defined line style.
MYPAT (iangle, itype, idens, icross)	defines a global shading pattern.
PENWID (nwidth)	sets the pen width.
SHDPAT (ipat)	selects a shading pattern.
SOLID ()	sets a solid line style.

Figure B.12: Line Styles and Shading Patterns

B.13 Cycles

Routine	Meaning
CLRCYC (index, iclr)	modifies the colour cycle.
LINCYC (index, itype)	modifies the line style cycle.
PATCYC (index, ipat)	modifies the pattern cycle.

Figure B.13: Cycles

B.14 Base Transformations

Routine	Meaning
TRFRES ()	resets base transformations.
TRFROT (xang, nx, ny)	affects the rotation of plot vectors.
TRFSCL (xscl, yscl)	affects the scaling of plot vectors.
TRFSHF (nx, ny)	affects the shifting of plot vectors.

Figure B.14: Base Transformations

B.15 Shielding

Routine	Meaning
SHIELD (carea, cmode)	defines automatic shielding.
SHLCIR (nx, ny, nr)	defines circles as shielded areas.
SHLDEL (id)	deletes shielded areas.
SHLELL (nx, ny, na, nb, t)	defines ellipses as shielded areas.
id = SHLIND ()	returns the index of a shielded area.
SHLPIE (nx, ny, nr, a, b)	defines pie segments as shielded areas.
SHLPOL (nxray, nyray, n)	defines polygons as shielded areas.
SHLRCT (nx, ny, nw, nh, t)	defines rotated rectangles as shielded areas.
SHLREC (nx, ny, nw, nh)	defines rectangles as shielded areas.
SHLRES (n)	deletes shielded areas.
SHLVIS (id, cmode)	enables or disables shielded areas.

Figure B.15: Shielding

B.16 Parameter Requesting Routines

Routine	Meaning
calf = GETALF ()	returns the base alphabet.
n = GETANG ()	returns the current angle used for text and numbers.
GETCLP (nx, ny, nw, nh)	returns the currents clipping window.
n = GETCLR ()	returns the current colour number.
GETDIG (nx, ny, nz)	returns the number of decimal places used in labels.
cfil = GETFIL ()	returns the current plotfile name.
GETGRF (a, b, or, stp, cax)	returns the scaling of the current axis system.
n = GETHGT ()	returns the current character height.
GETIND (i, xr, xg, xb)	returns the RGB coordinates for a colour index.
GETLAB (cx, cy, cz)	returns the current labels.
GETLEN (nx, ny, nz)	returns the current axis lengths.
n = GETLEV ()	returns the current level.
n = GETLIN ()	returns the current line width.
cmfl = GETMFL ()	returns the current file format.
c = GETMIX (copt)	returns shift characters for indices and exponents.
GETOR (nx, ny)	returns the current origin.
GETPAG (nx, ny)	returns the current page size.
n = GETPAT ()	returns the current shading pattern.
GETPOS (nx, ny)	returns the position of the axis system.
GETRAN (nx, ny)	returns the range of colour bars.
GETRES (nx, ny)	returns the size of points used in 3-D colour graphics.

Routine	Meaning
GETRGB (xr, xg, xb)	returns the RGB coordinates of the current colour.
GETSCL (nx, ny, nz)	returns the current axis scaling.
c = GETSHF (copt)	returns shift characters for European characters.
GETSP1 (nx, ny, nz)	returns the distance between axis ticks and labels.
GETSP2 (nx, ny, nz)	returns the distance between axis labels and names.
GETSYM (nsym, nh)	returns the current symbol number and height.
GETTCL (nmaj, nmin)	returns the current tick lengths.
GETTIC (nx, ny, nz)	returns the number of ticks plotted between labels.
n = GETTYP ()	returns the current line style.
n = GETUNI ()	returns the current unit used for messages.
x = GETVER ()	returns the DISLIN version number.
GETVK (nytit, nxbar, nybar)	returns the current lengths used for shifting.
cvlt = GETVLT ()	returns the current colour table.
n = GETWID ()	returns the width of colour bars.
GETWIN (nx, ny, nw, nh)	returns the position and size of the graphics window.
id = GETXID ('WINDOW')	returns the X window ID.
n = GMXALF (copt, c1, c2)	returns shift characters for additional alphabets.

Figure B.16: Parameter Requesting Routines

B.17 Elementary Plot Routines

Routine	Meaning
ARCELL (nx, ny, na, nb, alpha, beta, theta)	plots elliptical arcs.
AREAF (nxray, nyray, n)	plots polygons.
CIRCLE (nx, ny, nr)	plots circles.
CONNPT (x, y)	plots a line to a point.
ELLIPS (nx, ny, nr1, nr2)	plots ellipses.
LINE (nx, ny, nu, nv)	plots lines.
NOARLN ()	suppresses the outline of geometric figures.
PIE (nx, ny, nr, a, b)	plots pie segments.
POINT (nx, ny, nb, nh, nc)	plots coloured rectangles where the position is defined by the centre point.
RECFLL (nx, ny, nw, nh, nc)	plots coloured rectangles.
RECTAN (nx, ny, nw, nh)	plots rectangles.
RNDREC (nx, ny, nw, nh, iopt)	plots a rectangle with rounded corners.
RLARC (x, y, r1, r2, a, b, t)	plots elliptical arcs for user coordinates.
RLAREA (xray, yray, n)	plots polygons for user coordinates.

Routine	Meaning
RLCIRC (x, y, r)	plots circles for user coordinates.
RLELL (x, y, r1, r2)	plots ellipses for user coordinates.
RLINE (x, y, u, v)	plots lines for user coordinates.
RLPIE (x, y, r, a, b)	plots pie segments for user coordinates.
RLPOIN (x, y, nw, nh, nc)	plots coloured rectangles for user coordinates.
RLREC (x, y, xw, xh)	plots rectangles for user coordinates.
RLRND (x, y, xw, xh, iopt)	plots for user coordinates a rectangle with rounded corners.
RLSEC (x, y, r1, r2, a, b, ncol)	plots coloured pie sectors for user coordinates.
RLVEC (x1, y1, x2, y2, ivec)	plots vectors for user coordinates.
SECTOR (nx, ny, nr1, nr2, a, b, ncol)	plots coloured pie sectors.
STRTPT (x, y)	moves the pen to a point.
VECTOR (nx, ny, nu, nv, ivec)	plots vectors.
XMOVE (x, y)	moves the pen to a point.
XDRAW (x, y)	plots a line to a point.

Figure B.17: Elementary Plot Routines

B.18 Conversion of Coordinates

Routine	Meaning
COLRAY (zray, nray, n)	converts Z-coordinates to colour numbers.
n = NXPOSN (x)	converts X-coordinates to plot coordinates.
n = NYPOSN (y)	converts Y-coordinates to plot coordinates.
n = NZPOSN (z)	converts Z-coordinates to colour numbers.
TRFCO1 (xray, n, cfrom, cto)	converts one-dimensional coordinates.
TRFCO2 (xray, yray, n, cfrom, cto)	converts two-dimensional coordinates.
TRFCO3 (xray, yray, zray, n, cfrom, cto)	converts three-dimensional coordinates.
TRFREL (xray, yray, n)	converts X- and Y-coordinates to plot coordinates.
x = XINVRS (nx)	converts X plot coordinates to user coordinates.
x = XPOSN (x)	converts X-coordinates to real plot coordinates.
y = YINVRS (ny)	converts Y plot coordinates to user coordinates.
y = YPOSN (y)	converts Y-coordinates to real plot coordinates.

Figure B.18: Conversion of Coordinates

B.19 Utility Routines

Routine	Meaning
BEZIER (xray, yray, n, xpray, ypray, np)	calculates a Bezier interpolation.
n = BITSI2 (nbits, ninp, iinp, nout, iout)	allows bit manipulation on 16 bit variables.
n = BITSI4 (nbits, ninp, iinp, nout, iout)	allows bit manipulation on 32 bit variables.
n = FCHA (x, ndig, cstr)	converts floating-point numbers to character strings.
n = FLEN (x, ndig)	calculates the number of digits for floating-point numbers.
HISTOG (xray, n,	calculates a histogram.
n = INTCHA (nx, cstr)	converts integers to character strings.
n = INTLEN (nx)	calculates the number of digits for integers.
n = NLMESS (cstr)	returns the length of character strings in plot coordinates.
n = NLNUMB (x, ndig)	returns the length of numbers in plot coordinates.
SORTR1 (xray, n, copt)	sorts floating-point numbers.
SORTR2 (xray, yray, n, copt)	sorts points in the X-direction.
SPLINE (xray, yray, n, xsray, ysray, nspl)	returns splined points as calculated in CURVE.
SWAPI2 (iray, n)	swaps the bytes of 16 bit integer variables.
SWAPI4 (iray, n)	swaps the bytes of 32 bit integer variables.
n = TRMLEN (cstr)	calculates the number of characters in character strings.
UPSTR (cstr)	converts a character string to uppercase letters.

Figure B.19: Utility Routines

B.20 Date Routines

Routine	Meaning
BASDAT (id, im, iy)	defines the base date.
n = INCDAT (id, im, iy)	returns incremented days.
n = NWKDAY (id, im, iy)	returns the weekday of a date.
TRFDAT (n, id, im, iy)	converts incremented days to a date.

Figure B.20: Date Routines

B.21 Cursor Routines

Routine	Meaning
CSRMOV (nxray, nyray, nmax, n, iret)	collects cursor movements.
CSRPT1 (nx, ny)	returns a pressed cursor position.
CSRPTS (nxray, nyray, nmax, n, iret)	collects cursor positions.
CSRUNI (copt)	selects the unit of returned cursor positions.

Figure B.21: Cursor Routines

B.22 Bar Graphs

Routine	Meaning
BARBOR (iclr)	defines the colour of bar borders.
BARCLR (ic1, ic2, ic3)	defines bar colours.
BARGRP (ngrp, gap)	affects clustered bars.
BAROPT (xf, ang)	modifies teh appearance of 3-D bars.
BARPOS (copt)	selects predefined positions for bars.
BARS (xray, y1ray, y2ray, n)	plots bar graphs.
BARTYP (copt)	selects vertical or horizontal bars.
LABCLR (nclr, 'BARS')	defines the colour of bar labels.
LABDIG (ndig, 'BARS')	defines the number of decimal places in bar labels.
LABELS (copt, 'BARS')	defines bar labels.
LABPOS (copt, 'BARS')	defines the position of bar labels.

Figure B.22: Bar Graphs

B.23 Pie Charts

Routine	Meaning
CHNPIE (copt)	defines colour and pattern attributes for pie segments.
LABCLR (nclr, 'PIE')	defines the colour of segment labels.
LABDIG (ndig, 'PIE')	defines the number of decimal places in segment labels.
LABELS (copt, 'PIE')	defines pie labels.
LABPOS (copt, 'PIE')	defines the position of segment labels.
LABTYP (copt, 'PIE')	modifies the appearance of segment labels.
PIEBOR (iclr)	defines the colour of pie borders.
PIECLR (ic1ray, ic2ray, n)	defines pie colours.
PIEEXP ()	defines exploded pie segments.
PIEGRF (cbuf, nlin, xray, n)	plots pie charts.
PIELAB (clab, cpos)	sets additional character strings plotted in segment labels.

Routine	Meaning
PIEOPT (xf, ang)	modifies teh appearance of 3-D pies.
PIETYP (copt)	selects 2-D of 3-D pie charts.
PIEVEC (ivec, copt)	modifies the arrow plotted between labels and segments.

Figure B.23: Pie Charts

B.24 Coloured 3-D Graphics

Routine	Meaning
AX3LEN (nx, ny, nz)	defines axis lengths.
COLOR (color)	defines colours.
COLRAN (nx, ny)	defines the range of colour bars.
CRVMAT (zmat, n, m, ixpts, iypts)	plots a coloured surface.
CURVE3 (xray, yray, zray, n)	plots coloured rectangles.
CURVX3 (xray, y, zray, n)	plots rows of coloured rectangles.
CURVY3 (x, yray, zray, n)	plots columns of coloured rectangles.
ERASE ()	erases the screen.
GRAF3 (xa, xe, xor, xstp, ya, ye, yor, ystp, za, ze, zor, zstp)	plots a coloured axis system.
HSVRGB (xh, xs, xv, xr, xg, xb)	converts HSV to RGB coordinates.
MYVLT (rray, gray, bray, n)	changes the current colour table.
NOBAR ()	suppresses the plotting of colour bars.
NOBGD ()	suppresses the plotting of points which have the same colour as the background.
n = NZPOSN (z)	converts a Z-coordinate to a colour number.
POINT (nx, ny, nb, nh, nc)	plots coloured rectangles where the position is defined by the centre point.
RECFLL (nx, ny, nw, nh, nc)	plots coloured rectangles.
REVSCR ()	exchanges the colours with the indices 0 and 255.
RGBHSV (xr, xg, xb, xh, xs, xv)	converts RGB to HSV coordinates.
RLPOIN (x, y, nw, nh, nc)	plots coloured rectangles for user coordinates.
RLSEC (x, y, r1, r2, a, b, ncol)	plots coloured pie sectors for user coordinates.
SECTOR (nx, ny, nr1, nr2, a, b, ncol)	plots coloured pie sectors.
SETCLR (nclr)	defines colours.

Routine	Meaning
SETIND (i, xr, xg, xb)	changes the current colour table.
SETRES (nx, ny)	defines the size of coloured rectangles.
SETRGB (xr, xg, xb)	defines colours.
SETVLT (cvlt)	selects a colour table.
VKXBAR (nshift)	shifts colour bars in the X-direction.
VKYBAR (nshift)	shifts colour bars in the Y-direction.
WIDBAR (nw)	defines the width of colour bars.
ZAXIS (za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a linearly scaled colour bar.
ZAXLG (za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a logarithmically scaled colour bar.

Figure B.24: Coloured 3-D Graphics

B.25 3-D Graphics

Routine	Meaning
ABS3PT (x, y, z, xp, yp)	converts absolute 3-D coordinates to plot coordinates.
AXIS3D (x, y, z)	defines the lengths of the 3-D box.
BOX3D ()	plots a border around the 3-D box.
CONN3D (x, y, z)	plots a line to a point in 3-D space.
CURV3D (xray, yray, zray, n)	plots curves or symbols.
FLAB3D ()	disables the suppression of axis labels.
GETMAT (xray, yray, zray, n, zmat, nx, ny, zv, imat, wmat)	calculates a function matrix from randomly distributed data points.
GRAF3D (xa, xe, xor, xstp, ya, ye, yor, ystp, za, ze, zor, zstp)	plots an axis system.
GRFFIN ()	terminates a projection into 3-D space.
GRFINI (x1, y1, z1, x2, y2, z2, x3, y3, z3)	initializes projections in 3-D space.
GRID3D (nx, ny, copt)	plots a grid.
MDFMAT (ix, iy, w)	modifies the algorithm used in GETMAT.
NOHIDE ()	disables the hidden-line algorithm.
POS3PT (x, y, z, xp, yp, zp)	converts user coordinates to absolute 3-D coordinates.
REL3PT (x, y, z, xp, yp)	converts user coordinates to plot coordinates.
SHLSUR ()	protects surfaces from overwriting.
STRT3D (x, y, z)	moves the pen to a point.
SURCLR (itop, ibot)	selects surface colours.

Routine	Meaning
SURFCE (xray, nx, yray, ny, zmat)	plots the surface of a function matrix.
SURFCP (cfun, a1, a2, astp, b1, b2, bstp)	plots the surface of a parametric function.
SURFUN (cfun, ixp, xdel, iyp, ydel)	plots the surface grid of a function.
SURMAT (zmat, nx, ny, ixpts, iypts)	plots the surface of a function matrix.
SURMSH (copt)	enables grid lines for surfcp and surshd.
SUROPT (copt)	suppresses surface lines for surfce.
SURSHD (xray, nx, yray, ny, zmat)	plots a coloured surface.
SURVIS (copt)	determines the visible part of surfaces.
VECTR3 (x1, y1, z1, x2, y2, z2, ivec)	plots vectors in 3-D space.
VANG3D (ang)	defines the field of view.
VIEW3D (x, y, z, copt)	defines the viewpoint.
VUP3D (ang)	defines the camera orientation.
ZBFFIN ()	terminates the Z-buffer.
iret = ZBFINI ()	allocates space for a Z-buffer.
ZBFLIN (x1, y1, z1, x2, y2, z2)	plots lines.
ZBFTRI (xray, yray, zray, iray)	plots triangles.
ZSCALE (zmin, zmax)	defines a Z-scaling for coloured surfaces.

Figure B.25: 3-D Graphics

B.26 Geographical Projections

Routine	Meaning
CURVMP (xray, yray, n)	plots curves or symbols.
GRAFMP (xa, xe, xor, xstp, ya, ye, yor, ystp)	plots a geographical axis system.
GRIDMP (nx, ny)	plots a grid.
MAPBAS (copt)	defines the base map.
MAPLEV (copt)	specifies land or lake plotting.
MAPMOD (copt)	modifies the connection of points used in CURVMP.
MAPPOL (xpol, ypol)	defines the map pole used for azimuthal projections.
MAPREF (ylw, yup)	defines two latitudes used for conical projections.
POS2PT (x, y, xp, yp)	converts user coordinates to plot coordinates.
PROJCT (copt)	selects a projection.

Routine	Meaning
SHDEUR (inray, ipray, icray, n)	shades European countries.
SHDMAP (copt)	shades continents.
WORLD ()	plots coastlines and lakes.
XAXMAP (xa, xe, xor, xstp, cstr, nt, ny)	plots a secondary X-axis.
YAXMAP (ya, ye, yor, ystp) cstr, nt, nx)	plots a secondary Y-axis.

Figure B.26: Geographical Projections

B.27 Contouring

Routine	Meaning
CONCRV (xray, yray, n, z)	plots generated contours.
CONGAP (xfac)	affects the spacing between contour lines and labels.
CONLAB (copt)	defines a character string used for contour labels.
CONMAT (zmat, nx, ny, z)	plots contours.
CONMOD (xfac, xquot)	affects the position of contour labels.
CONSHD (xray, nx, yray, ny, zmat, zlray, n)	plots shaded contours.
CONTUR (xray, nx, yray, ny, zmat, zlev)	plots contours.
LABCLR (nclr, 'CONT')	defines the colour of contour labels.
LABDIS (ndis, 'CONT')	defines the distance between labels.
LABELS (copt, 'CONT')	defines contour labels.
SHDMOD (copt, 'CONT')	sets the algorithm for shaded contours.

Figure B.27: Contouring

B.28 Image Routines

Routine	Meaning
IMGINI ()	initializes transfering of image data.
IMGFIN ()	terminates transfering of image data.
RIMAGE (cfil)	copies an image from memory to a file.
RPIXEL (ix, iy, iclr)	reads a pixel from memory.
RPIXLS (iray, ix, iy, nw, nh)	reads image data from memory.
RPNG (cfil)	stores an image as a PNG file.
RPXROW (iray, nx, ny, n)	reads a row of image data from memory.

Routine	Meaning
RTIFF (cfil)	stores an image as a TIFF file.
TIFORG (nx, ny)	defines the position of TIFF files copied with WTIFF.
TIFWIN (nx, ny, nw, nh)	defines a clipping window for TIFF files copied with WTIFF.
WIMAGE (cfil)	copies an image from file to memory.
WPIXEL (ix, iy, iclr)	writes a pixel to memory.
WPIXLS (iray, ix, iy, nw, nh)	writes image data to memory.
WPXROW (iray, nx, ny, n)	write a row of image data to memory.
WTIFF (cfil)	copies a TIFF file created by DISLIN to memory.

Figure B.28: Image Routines

B.29 Window Routines

Routine	Meaning
CLSWIN (id)	closes a window.
OPNWIN (id)	opens a window for graphics output.
SELWIN (id)	selects a window for graphics output.
WINDOW (nx, ny, nw, nh)	defines the position and size of windows.
id = WINID ()	returns the ID of the currently selected window.
WINKEY (ckey)	defines a key that can be used for program continuation in DISFIN.
WINMOD (copt)	affects the handling of windows in the termination routine DISFIN.
WINSIZ (nw, nh)	defines the size of windows.
WINTIT (cstr)	sets the title of the currently selected window.
X11MOD (copt)	enables backing store.
ival = DWGBUT (cstr, ival)	displays a message that can be answered with 'Yes' or 'No'.
cfil = DWGFIL (clab, cfil, cmask)	creates a file selection box.
isel = DWGLIS (clab, clis, isel)	gets a selection from a list of items.
DWGMSG (cstr)	displays a message.
cstr = DWGTXT (clab, cstr)	prompts an user for input.
cdsp = GETDSP ()	returns the terminal type.
n = GWGBOX (id)	requests the value of a box widget.
n = GWGBUT (id)	requests the status of a button widget.
cfil = GWGFIL (id)	requests the value of a file widget.
n = GWGLIS (id)	requests the value of a list widget.

Routine	Meaning
x = GWGSCL (id)	requests the value of a scale widget.
cstr = GWGTXT (id)	requests the value of a text widget.
ITMCAT (clis, citem)	concatenates an element to a list string.
n = ITMCNT (clis)	calculates the number of elements in a list string.
citem = ITMSTR (clis, n)	extracts an element from a list string.
MSGBOX (cstr)	displays a message.

Figure B.29: Window Routines

B.30 Widget Routines

Routine	Meaning
SWGBOX (id, isel)	changes the selection of a box widget.
SWGBUT (id, ival)	changes the status of a button widget.
SWGCB (id, routine, iray)	connects a widget with a callback routine.
SWGFIL (id, cfil)	changes the value of a file widget.
SWGHLP (cstr)	sets a character string for the Help menu.
SWGJUS (cjus, class)	defines the alignment of label widgets.
SWGLIS (id, isel)	changes the selection of a list widget.
SWGMIX (char, cmix)	defines control characters.
SWGMOD (copt)	defines ASCII or X Window mode for widgets.
SWGMRG (ival, cmrg)	defines widget margins.
SWGPOP (copt)	modifies the appearance of the popup menubar.
SWGPOS (nx, ny)	defines the position of widgets.
SWGSCL (id, xval)	changes the value of a scale widget.
SWGSIZ (nw, nh)	defines the size of widgets.
SWGTIT (cstr)	sets a title for the main widget.
SWGTXT (id, cval)	changes the value of a text widget.
SWGTYP (ctype, class)	modifies the appearance of widgets.
SWGWIN (nx, ny, nw, nh)	defines the position and size of widgets.
SWGWTH (nwth)	sets the default width of widgets.
id = WGAPP (ip, clab)	creates an entry in a popup menu.
id = WGBAS (ip, copt)	creates a container wdiget.
id = WGBOX (ip, clis, isel)	creates a list widget with toggle buttons.
id = WGBUT (ip, cval, ival)	creates a button widget.
id = WGCMD (ip, clab, cmd)	creates a push button widget. A system command will be executed if the button is pressed.
id = WGFIL (ip, clab, cfil, cmask)	creates a file widget.

Routine	Meaning
WGFIN ()	terminates widget routines.
id = WGINI (copt)	creates a main widget and initalizes widget routines.
id = WGLAB (ip, cstr)	creates a label widget.
id = WGLIS (ip, clis, isel)	creates a list widget.
id = WGLTXT (ip, clab, cstr, nwth)	creates a labeled text widget.
id = WGOK (ip)	creates a OK push button widget.
id = WGPBUT (ip, clab)	creates a push button widget.
id = WGPOP (ip, cstr)	creates a popup menu.
id = WGQUIT (ip)	creates a Quit push button widget.
id = WGSCL (ip, clab, xmin, xmax, xval, ndez)	creates a scale widget.
id = WGTXT (ip, cstr)	creates a text widget.

Figure B.30: Widget Routines

B.31 DISLIN Quickplots

Routine	Meaning
QPLBAR (xray, yray, n)	plots a bar graph.
QPLCLR (zmat, n, m)	plots a coloured surface.
QPLCON (zmat, n, m, nlv)	makes a contour plot.
QPLOT (xray, yray, n)	plots a curve.
QPLPIE (xray, yray, n)	plots a pie chart.
QPLSCA (xray, yray, n)	makes a scatter plot.
QPLSUR (zmat, n, m)	plots a surface.

Figure B.31: DISLIN Quickplots

B.32 MPAe Emblem

Routine	Meaning
MPAEPL (iopt)	plots the MPAe emblem.
MPLANG (xang)	defines a rotation angle for the MPAe emblem.
MPLCLR (nbg, nfg)	defines the fore- and background colours of the MPAe emblem.
MPLPOS (nx, ny)	defines the position of the MPAe emblem.
MPLSIZ (nsize)	defines the size of the MPAe emblem.
NOFILL ()	suppresses the shading of the MPAe emblem.

Figure B.32: MPAE Emblem

Appendix C

Examples

This appendix presents some examples of the DISLIN manual in DISGCL coding. The examples can be found in the DISLIN subdirectory gcldir and can be executed with the command:

disgcl example

where example is one of the examples.

C.1 Demonstration of CURVE

```
%GCL
// Demonstration of CURVE

N=101
PI    = 3.1415926

XRAY  = FALLOC (N)
XRAY  = (XRAY - 1.) * 3.6
YRAY1 = SIN (XRAY * PI / 180.)
YRAY2 = COS (XRAY * PI / 180.)

METAFL ('CONS')
DISINI ()
COMPLX ()
PAGERA ()

NAME   ('X-axis', 'X')
NAME   ('Y-axis', 'Y')
TITLIN ('Demonstration of CURVE', 2)
TICKS  (10, 'X')
LABDIG (-1, 'X')

GRAF   (0.,360.,0.,90.,-1.,1.,-1.,0.5)
TITLE  ()

CURVE  (xray, yray1, n)
CURVE  (xray, yray2, n)

DASH   ()
XAXGIT ()
DISFIN ()
```

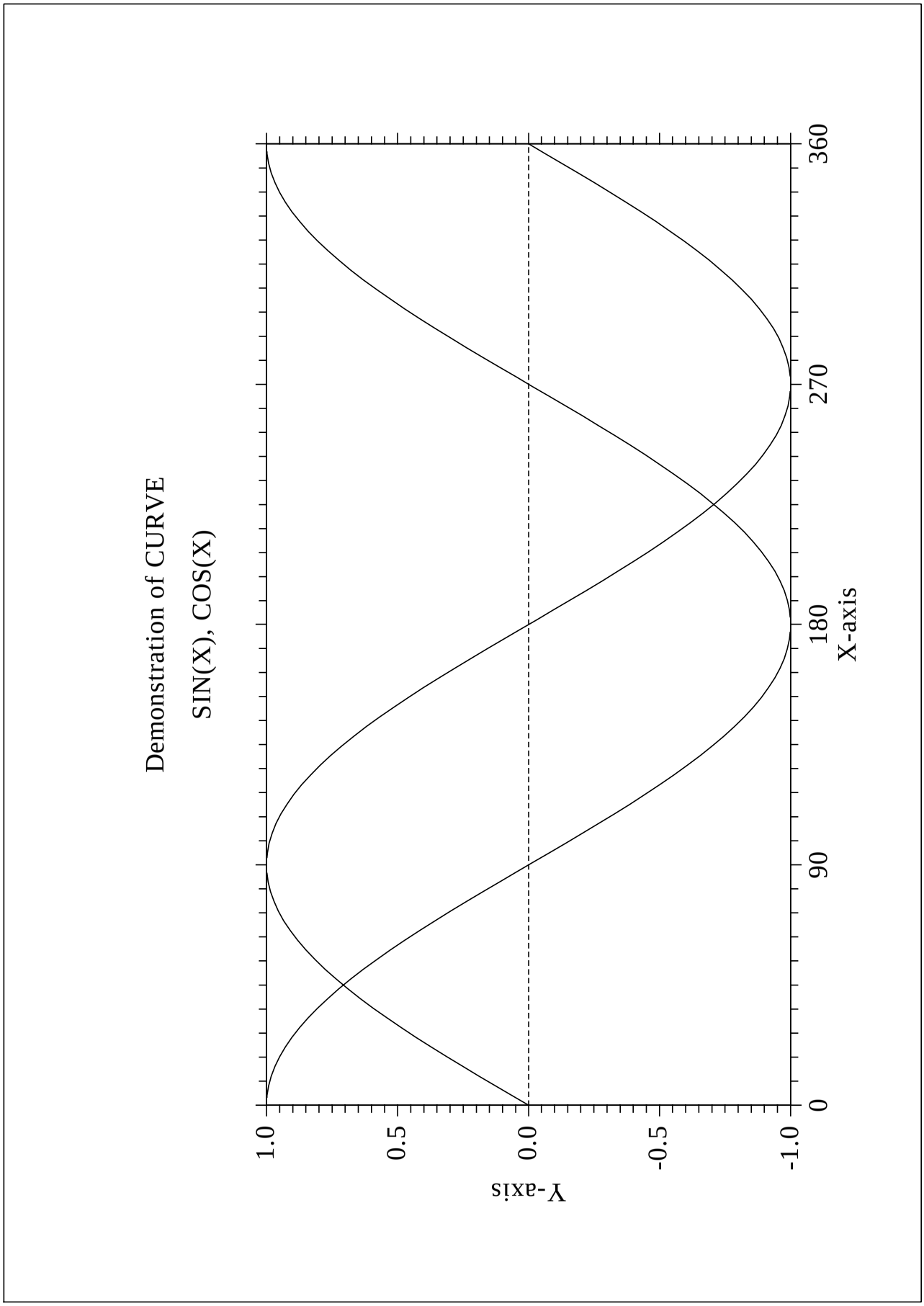

Figure C.1: Demonstration of CURVE

C.2 3-D Colour Plot

```
%GCL
// 3-D Colour Plot

N = 100
FLOAT ZMAT[N,N], XRAY[N]

PI = 3.1415927
STEP = 2. * PI / (N - 1)

DO I = 0, N - 1
  XRAY[I] = SIN (I * STEP)
END DO

DO J = 0, N - 1
  ZMAT[*,J] = 2 * XRAY * SIN (J * STEP)
END DO

METAFL ('CONS')
DISINI ()
COMPLX ()
PAGERA ()

LABDIG (-1, 'XYZ')
NAME   ('X-Achse', 'X')
NAME   ('Y-Achse', 'Y')
NAME   ('Z-Achse', 'Z')
TITLIN ('3-D Colour Plot of the Function', 1)
TITLIN ('F(X,Y) = 2 * SIN(X) * SIN(Y)', 3)

AUTRES (N, N)
GRAF3  (0.,360.,0.,90.,0.,360.,0.,90.,-2.,2.,-2.,1.0)
TITLE  ()

CRVMAT (ZMAT, N, N, 1, 1)
MPAEPL (3)
DISFIN ()
```

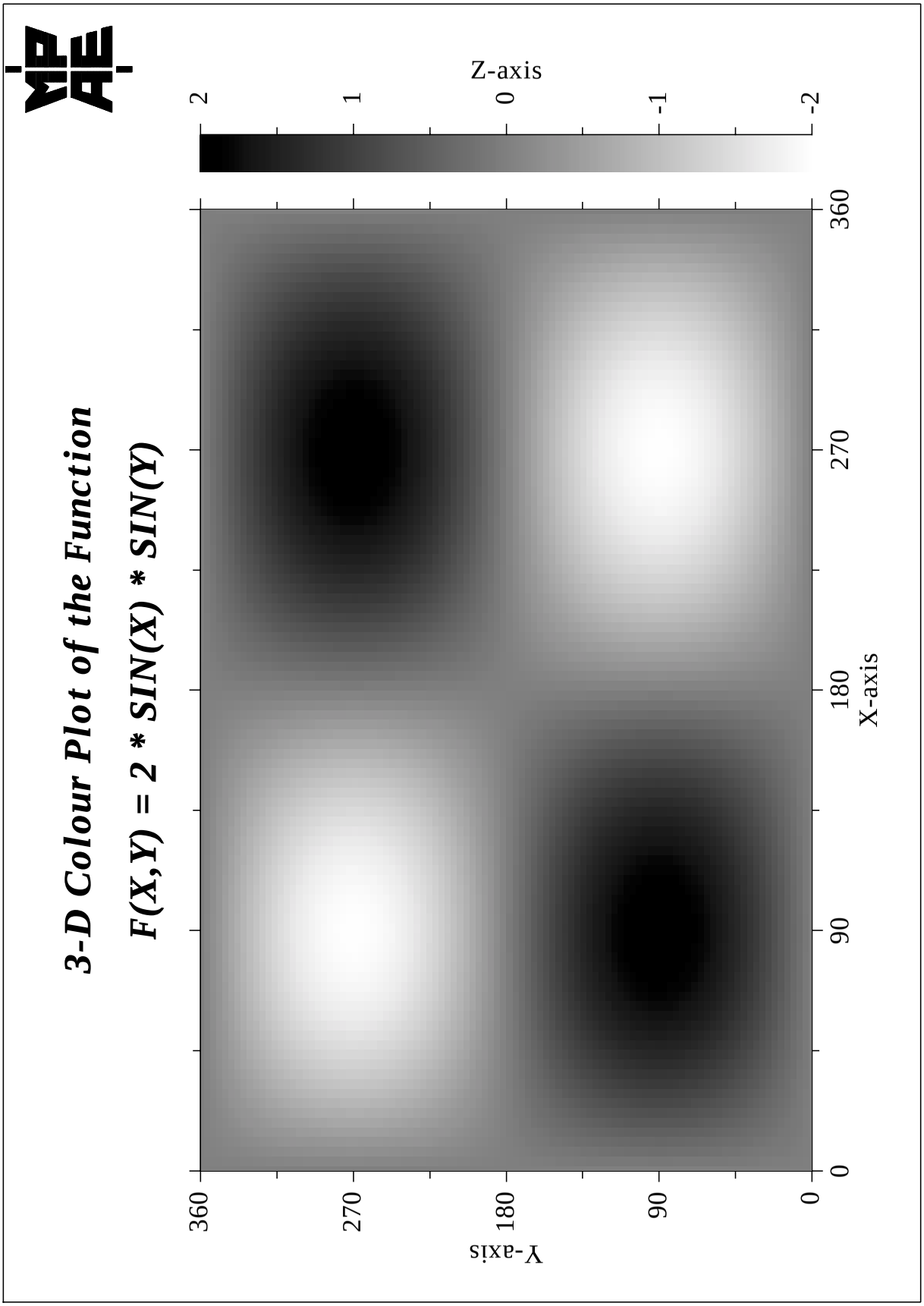

Figure C.2: 3-D Colour Plot

C.3 Surface Plot

```
%GCL
// Surface Plot
N = 100
FLOAT ZMAT[N,N], XRAY[N]

PI = 3.1415927
STEP = 2. * PI / (N - 1)

DO I = 0, N - 1
  XRAY[I] = SIN (I * STEP)
END DO

DO J = 0, N - 1
  ZMAT[*,J] = 2 * XRAY * SIN (J * STEP)
END DO

METAFL ('CONS')
SETPAG ('DA4P')
DISINI ()
COMPLX ()
PAGERA ()

AXSPOS (200, 2600)
AXSLEN (1800, 1800)

NAME   ('X-axis', 'X')
NAME   ('Y-axis', 'Y')
NAME   ('Z-axis', 'Z')

TITLIN ('Surface Plot (SURMAT)', 2)
TITLIN ('F(X,Y) = 2 * SIN(X) * SIN(Y)', 4)

VIEW3D (-5.,-5.,4.,'ABS')
GRAF3D (0.,360.,0.,90.,0.,360.,0.,90.,-3.,3.,-3.,1.)
HEIGHT (50)
TITLE  ()

COLOR  ('GREEN')
SHLSUR ()
SURMAT (ZMAT, N, N, 1, 1)

DISFIN ()
```

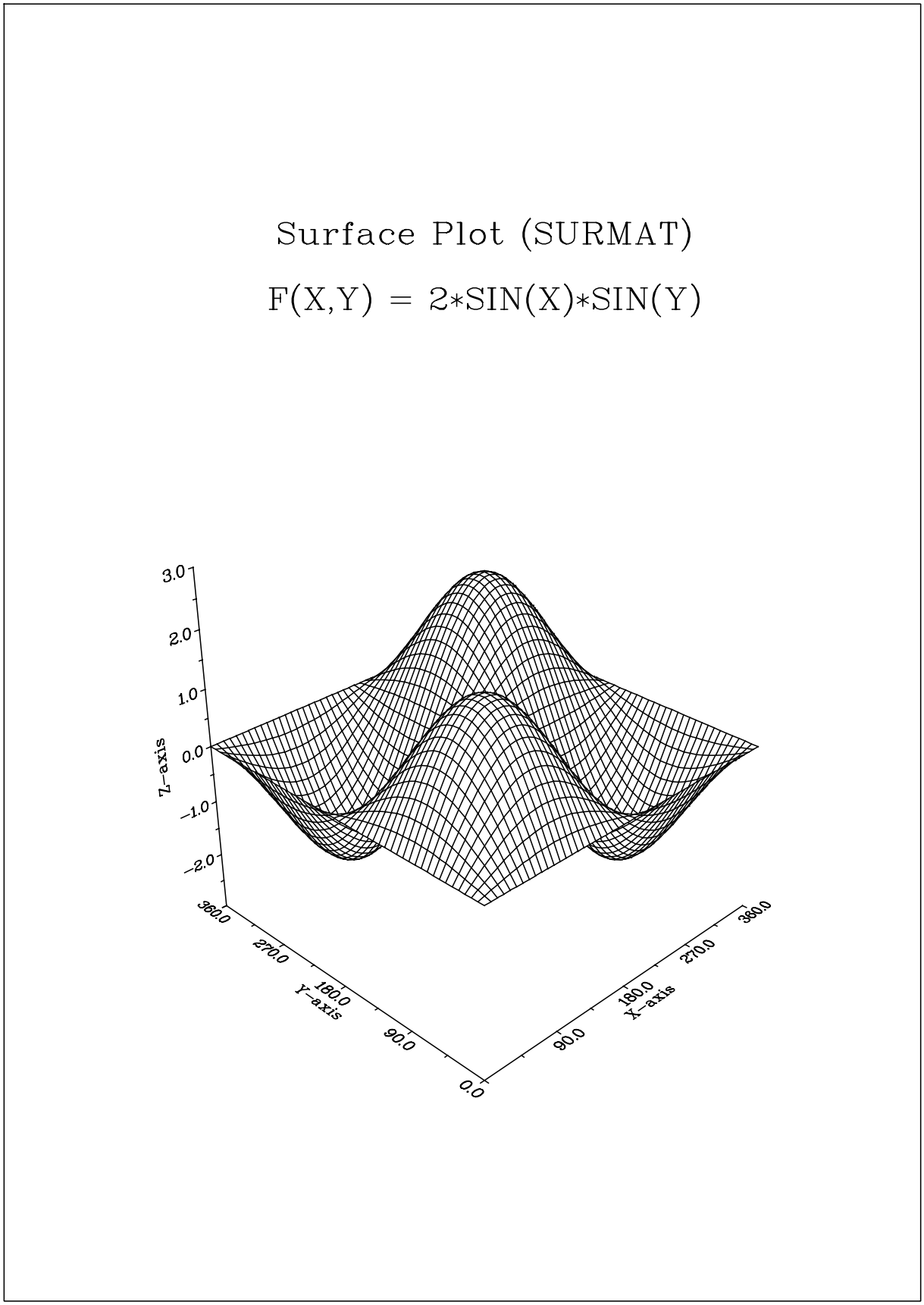

Figure C.3: Surface Plot

C.4 Contour Plot

```
%GCL
// Contour Plot
N = 100
FLOAT ZMAT[N,N], XRAY[N]

FPI = 3.1415927 / 180.
STEP = 360. / (N - 1)
X = FALLOC (N) * STEP
Y = FALLOC (N) * STEP

DO J = 0, N - 1
  ZMAT[*,J] = 2 * SIN (X * FPI) * SIN (Y[J] * FPI)
END DO

METAFL ('CONS')
SETPAG ('DA4P')
DISINI ()
COMPLX ()
PAGERA ()

INTAX  ()
AXSPOS (450, 2670)

NAME   ('X-axis', 'X')
NAME   ('Y-axis', 'Y')
TITLIN ('Contour Plot', 2)
TITLIN ('F(X,Y) = 2 * SIN(X) * SIN(Y)', 4)

GRAF (0.,360.,0.,90.,0.,360.,0.,90.)

HEIGHT (30)
DO I = 1,9
  ZLEV = -2.0 + (I - 1) * 0.5
   IF (I == 5)
    LABELS ('NONE', 'CONTUR')
   ELSE
    LABELS ('FLOAT', 'CONTUR')
  END IF
  CONTUR (X, N, Y, N, ZMAT, ZLEV)
END DO

HEIGHT (50)
TITLE  ()
DISFIN ()
```

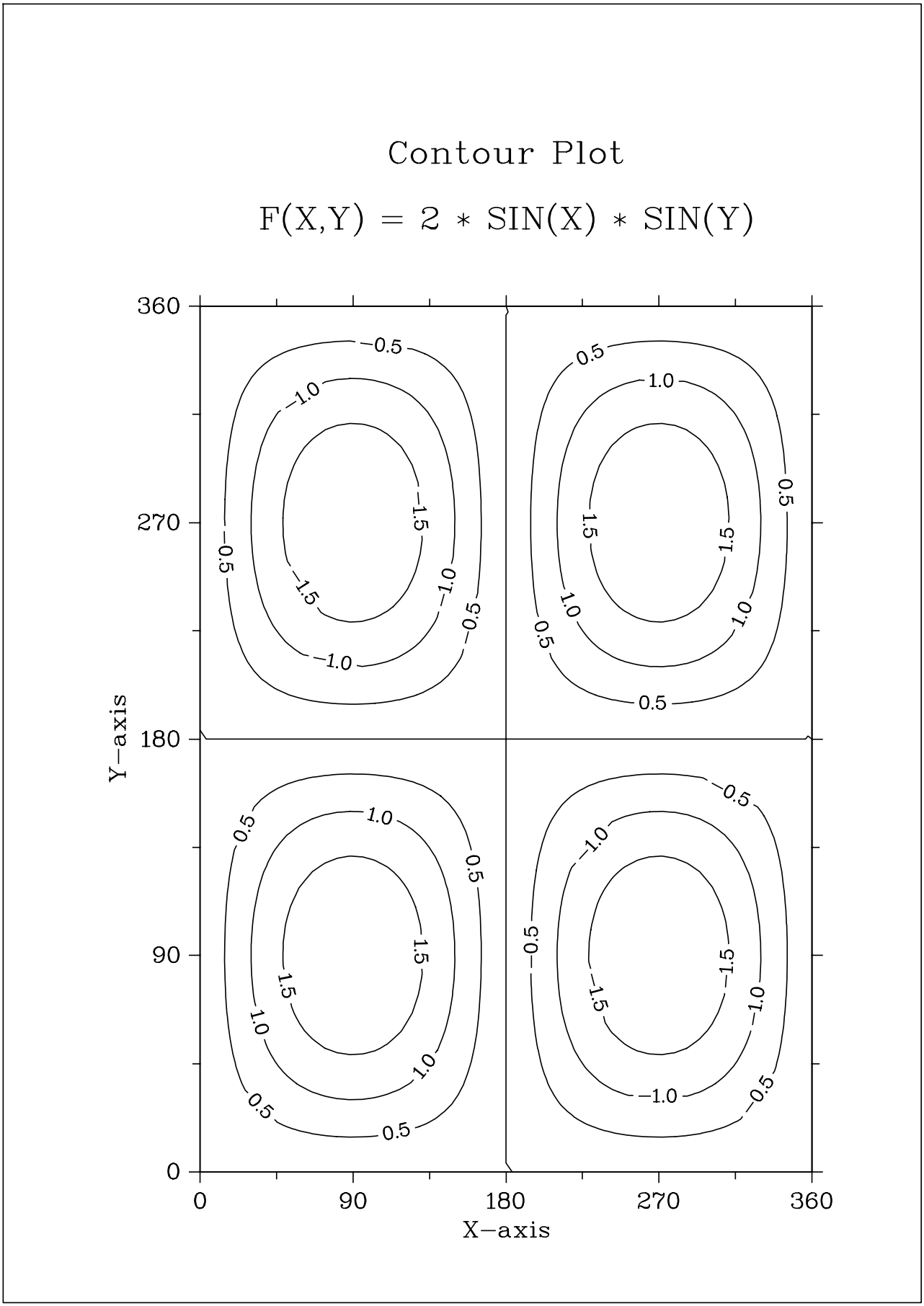

Figure C.4: Contour Plot

C.5 Shaded Contour Plot

```
%GCL
// Shaded Contour Plot
N = 100
FLOAT ZMAT[N,N], XRAY[N], YRAY[N], ZLEV[12]

STEP = 1.6 / (N - 1)
XRAY = FALLOC (N) * STEP
YRAY = FALLOC (N) * STEP
X    = XRAY * XRAY - 1

DO J = 0, N - 1
  Y=YRAY[J] * YRAY[J] - 1.
  ZMAT[*,J] = X * X + Y * Y
END DO

METAFL ('CONS')
SETPAG ('DA4P')
DISINI ()
COMPLX ()
PAGERA ()

NAME   ('X-axis', 'X')
NAME   ('Y-axis', 'Y')

MIXALF ()
TITLIN ('Shaded Contour Plot', 1)
TITLIN ('F(X,Y) = (X[2$ - 1)[2$ + (y[2$ - 1)[2$', 3)

SHDMOD ('POLY', 'CONTUR')
AXSPOS (450, 2670)
GRAF   (0., 1.6, 0., 0.2, 0., 1.6, 0., 0.2)

HEIGHT (30)
DO I = 0, 11
  ZLEV[11-I] = 0.1 + I * 0.1
END DO

CONSHD (XRAY, N, YRAY, N, ZMAT, ZLEV, 12)

HEIGHT (50)
TITLE  ()
DISFIN ()
```

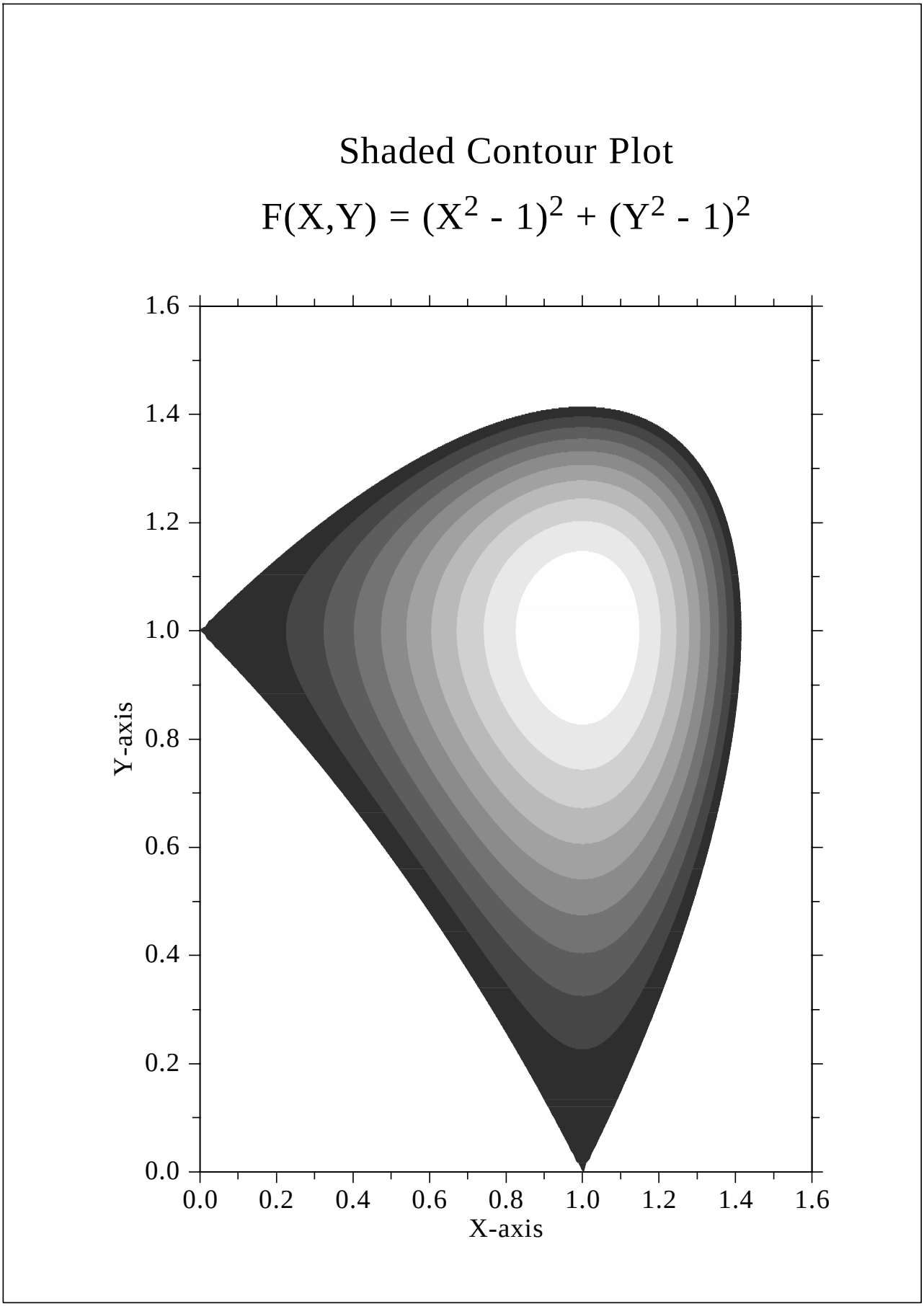

Figure C.5: Shaded Contour Plot

C.6 World Coastlines and Lakes

```
%GCL
// World Coastlines and Lakes

METAFL ('CONS')
DISINI ()
PAGERA ()
COMPLX ()

AXSPOS (400, 1850)
AXSLEN (2400, 1400)

NAME   ('Longitude', 'X')
NAME   ('Latitude', 'X')
TITLIN ('World Coastlines and Lakes', 3)

LABELS ('MAP', 'XY')
GRAFMP (-180.,180.,-180.,90.,-90.,90.,-90.,30.)

GRIDMP (1, 1)
COLOR  ('GREEN')
WORLD  ()
COLOR  ('FORE')

HEIGHT (50)
TITLE  ()
DISFIN ()
```

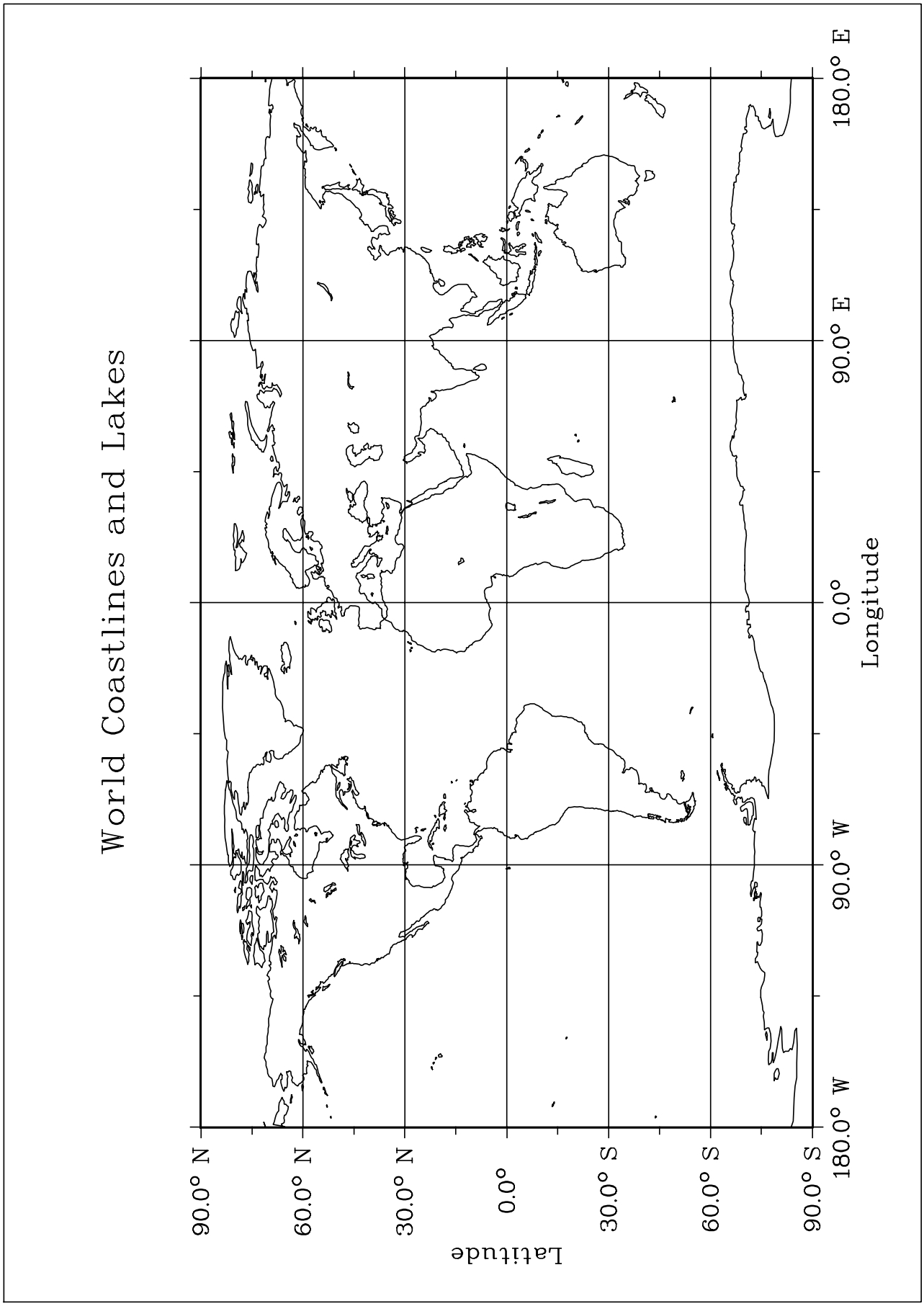

Figure C.6: World Coastlines and Lakes

C.7 Widgets

```
%GCL
// Widgets

CL1  = 'Item1|Item2|Item3|Item4|Item5'
CFIL = ' '

SWGTIT ('Example')
IP = WGINI ('VERT')
ID_LAB = WGLAB (IP, 'File Widget: ')
ID_FIL = WGFIL (IP, 'Open File', CFIL, '*.c')

ID_LAB = WGLAB (IP, 'List Widget: ')
ID_LIS = WGLIS (IP, CL1, 1)

ID_LAB = WGLAB (IP, 'Button Widgets')
ID_BT1 = WGBUT (IP, 'This is Button 1', 0)
ID_BT2 = WGBUT (IP, 'This is Button 2', 1)

ID_LAB = WGLAB (IP, 'Scale Widget')
ID_SCL = WGSCL (IP, ' ', 0., 10., 5., 1)

ID_OK  = WGOK  (IP)
WGFIN ()

CFIL = GWGFIL (ID_FIL)
ILIS = GWGLIS (ID_LIS)
IBT1 = GWGBUT (ID_BT1)
IBT2 = GWGBUT (ID_BT2)
XSCL = GWGSCL (ID_SCL)

FREE IP, CL1, ID_LAB, ID_OK, ID_FIL, ID_LIS, ID_BT1 @
     ID_BT2, ID_SCL
LIST
```

Figure C.7: Widgets

Index