A `FORTRAN 90` numerical library

Alberto Ramos. Madrid, November 2006.

Contents

Contents i

List of Tables x

Listings xi

Generalities xiii
Installing . . . xiii

1 MODULE `NumTypes` 1
1.1 Description . . . 1
1.2 Examples . . . 1

2 MODULE `Constants` 3
2.1 Name conventions . . . 3
2.2 π-related constants . . . 3
2.2.1 Real . . . 3
2.2.2 Complex . . . 3
2.3 Square roots and log related constants . . . 3
2.4 Other mathematical constants . . . 4

3 MODULE `Error` 5
3.1 Defined variables . . . 5
3.1.1 `stderr` . . . 5
3.2 Subroutine `perror([routine], msg)` . . . 5
3.2.1 Description . . . 5
3.2.2 Arguments . . . 5
3.2.3 Examples . . . 6
3.3 Subroutine `abort([routine], msg)` . . . 6
3.3.1 Description . . . 6
3.3.2 Arguments . . . 6
3.3.3 Examples . . . 6

4 MODULE `Integration` 9
4.1 Function `Trapecio(a, b, Func, [Tol])` . . . 9
4.1.1 Description . . . 9
4.1.2 Arguments . . . 9

4.1.3 Output 10
4.1.4 Examples 10
4.2 Function `Simpson(a, b, Func, [Tol])` 11
4.2.1 Description 11
4.2.2 Arguments 11
4.2.3 Output 11
4.2.4 Examples 11
4.3 Function `TrapecioAb(a, b, Func, [Tol])` 12
4.3.1 Description 12
4.3.2 Arguments 12
4.3.3 Output 13
4.3.4 Examples 13
4.4 Function `SimpsonAb(a, b, Func, [Tol])` 14
4.4.1 Description 14
4.4.2 Arguments 14
4.4.3 Output 14
4.4.4 Examples 14
4.5 Function `SimpsonInfUp(a, Func, [Tol])` 15
4.5.1 Description 15
4.5.2 Arguments 15
4.5.3 Output 16
4.5.4 examples 16
4.6 Function `SimpsonInfDw(a, Func, [Tol])` 17
4.6.1 Description 17
4.6.2 Arguments 17
4.6.3 Output 17
4.6.4 examples 17
4.7 Function `SimpsonSingUp(a, b, Func, [Tol], gamma)` 18
4.7.1 Description 18
4.7.2 Arguments 18
4.7.3 Output 19
4.7.4 Examples 19
4.8 Function `SimpsonSingDw(a, b, Func, [Tol], gamma)` 20
4.8.1 Description 20
4.8.2 Arguments 20
4.8.3 Output 20
4.8.4 Examples 20
4.9 Function `Euler(Init, Xo, Xfin, Feuler, [Tol])` 21
4.9.1 Description 21
4.9.2 Arguments 22
4.9.3 Output 22
4.9.4 Examples 22
4.10 Function `Rgnkta(Init, Xo, Xfin, Feuler, [Tol])` 24
4.10.1 Description 24
4.10.2 Arguments 24
4.10.3 Output 24
4.10.4 Examples 25

5 MODULE `Optimization` **27**

5.1 Function `Step(X, FStep[, Tol])` 27

5.1.1 Description 27

5.1.2 Arguments 27

5.1.3 Output 28

5.1.4 Example 28

5.2 Function `MaxPosition(FVal, IpX, IpY)` 29

5.2.1 Description 29

5.2.2 Arguments 29

5.2.3 Output 29

5.2.4 Example 30

6 MODULE `Linear` **31**

6.1 Subroutine `Pivoting(M,Ipiv,Idet)` 31

6.1.1 Description 31

6.1.2 Arguments 31

6.1.3 Examples 31

6.2 Subroutine `LU(M, Ipiv, Idet)` 32

6.2.1 Description 32

6.2.2 Arguments 32

6.2.3 Examples 33

6.3 Subroutine `LUsolve(M, b)` 34

6.3.1 Description 34

6.3.2 Arguments 34

6.3.3 Examples 34

6.4 Function `Det(M)` 35

6.4.1 Description 35

6.4.2 Arguments 35

6.4.3 Output 35

6.4.4 Examples 35

7 MODULE `NonNum` **37**

7.1 Subroutine `Swap(X,Ind1,Ind2)` 37

7.1.1 Description 37

7.1.2 Arguments 37

7.1.3 Examples 37

7.2 Subroutine `Insrt(X[, Ipt])` 38

7.2.1 Description 38

7.2.2 Arguments 38

7.2.3 Examples 38

7.3 Subroutine `Qsort(X[, Ipt])` 39

7.3.1 Description 39

7.3.2 Arguments 39

7.3.3 Examples 39

7.4 Function Locate(X, X_0[, Iin]) 40

7.4.1 Description 40

7.4.2 Arguments 40

7.4.3 Output . . . 40
7.4.4 Examples . . . 40

8 MODULE `SpecialFunc` **43**
8.1 Function `GammaLn(X)` . . . 43
8.1.1 Description . . . 43
8.1.2 Arguments . . . 43
8.1.3 Output . . . 43
8.1.4 Examples . . . 43
8.2 Function `Theta(i, z, tau[, Prec])` . . . 44
8.2.1 Description . . . 44
8.2.2 Arguments . . . 44
8.2.3 Output . . . 44
8.2.4 Examples . . . 44
8.3 Function `ThetaChar(a, b, z, tau[, Prec])` . . . 45
8.3.1 Description . . . 45
8.3.2 Arguments . . . 45
8.3.3 Output . . . 45
8.3.4 Examples . . . 45
8.4 Function `Hermite(n,x[, Dval])` . . . 46
8.4.1 Description . . . 46
8.4.2 Arguments . . . 46
8.4.3 Output . . . 46
8.4.4 Examples . . . 46
8.5 Function `HermiteFunc(n, x[, Dval])` . . . 47
8.5.1 Description . . . 47
8.5.2 Arguments . . . 47
8.5.3 Output . . . 47
8.5.4 Examples . . . 47
8.6 Function `Basis(X1, X2, n, s, q, itau[, Prec])` . . . 48
8.6.1 Description . . . 48
8.6.2 Arguments . . . 48
8.6.3 Output . . . 48
8.6.4 Examples . . . 48
8.7 Function `Factorial(N)` . . . 49
8.7.1 Description . . . 49
8.7.2 Arguments . . . 49
8.7.3 Output . . . 49
8.7.4 Examples . . . 49

9 MODULE `Statistics` **51**
9.1 Function `Mean(X)` . . . 51
9.1.1 Description . . . 51
9.1.2 Arguments . . . 51
9.1.3 Output . . . 51
9.1.4 Examples . . . 51
9.2 Function `Var(X)` . . . 52

9.2.1 Description . . . 52
9.2.2 Arguments . . . 52
9.2.3 Output . . . 52
9.2.4 Examples . . . 52
9.3 Function `Stddev(X)` . . . 52
9.3.1 Description . . . 52
9.3.2 Arguments . . . 53
9.3.3 Output . . . 53
9.3.4 Examples . . . 53
9.4 Function `Moment(X, k)` . . . 53
9.4.1 Description . . . 53
9.4.2 Arguments . . . 53
9.4.3 Output . . . 53
9.4.4 Examples . . . 54
9.5 Subroutine `Normal(X, [Rm], [Rsig])` . . . 54
9.5.1 Description . . . 54
9.5.2 Arguments . . . 54
9.5.3 Examples . . . 54
9.6 Subroutine `Laplace(X, Rm, Rb)` . . . 55
9.6.1 Description . . . 55
9.6.2 Arguments . . . 55
9.6.3 Examples . . . 55
9.7 Subroutine `Histogram(Val, Ndiv, Ntics, Vmin, Vmax, h)` . . . 56
9.7.1 Description . . . 56
9.7.2 Arguments . . . 56
9.7.3 Examples . . . 56
9.8 Subroutine `LinearReg(X, Y, Yerr, [Func], Coef, Cerr, ChisqrV)` . . . 57
9.8.1 Description . . . 57
9.8.2 Arguments . . . 57
9.8.3 Examples . . . 58

10 MODULE `Polynomial` **61**
10.1 `Type Pol` . . . 61
10.1.1 Description . . . 61
10.1.2 Components . . . 61
10.1.3 Examples . . . 61
10.2 `Type CmplxPol` . . . 62
10.2.1 Description . . . 62
10.2.2 Components . . . 62
10.2.3 Examples . . . 62
10.3 Assignment . . . 62
10.3.1 Description . . . 62
10.3.2 Examples . . . 62
10.4 Operator `+` . . . 63
10.4.1 Description . . . 63
10.4.2 Examples . . . 63
10.5 Operator `-` . . . 64

10.5.1 Description 64
10.5.2 Examples 64
10.6 Operator `*` 65
10.6.1 Description 65
10.6.2 Examples 65
10.7 Subroutine `Init(P, Dgr)` 66
10.7.1 Description 66
10.7.2 Arguments 66
10.7.3 Examples 66
10.8 Function `Degree(P)` 66
10.8.1 Description 66
10.8.2 Arguments 67
10.8.3 Output 67
10.8.4 Examples 67
10.9 Function `Value(P, X)` 68
10.9.1 Description 68
10.9.2 Arguments 68
10.9.3 Output 68
10.9.4 Examples 68
10.10 Function `Deriv(P)` 69
10.10.1 Description 69
10.10.2 Arguments 69
10.10.3 Output 69
10.10.4 Examples 69
10.11 Function `Integra(P[, Cte])` 70
10.11.1 Description 70
10.11.2 Arguments 70
10.11.3 Output 70
10.11.4 Examples 70
10.12 Function `InterpolValue(X, Y, Xo)` 71
10.12.1 Description 71
10.12.2 Arguments 71
10.12.3 Output 71
10.12.4 Examples 71
10.13 Function `Interpol(X, Y)` 72
10.13.1 Arguments 72
10.13.2 Output 72
10.13.3 Examples 72
10.14 Subroutine `Spline(X, Y, Ypp0, YppN, Pols)` 73
10.14.1 Description 73
10.14.2 Arguments 73
10.14.3 Examples 73

11 MODULE Root **75**
11.1 Subroutine RootPol(a, b, [c, d], z1, z2, [z3, z4]) 75
11.1.1 Description 75
11.1.2 Arguments 75
11.1.3 Examples 75
11.2 Function Newton(Xo, Fnew, [Tol]) 76
11.2.1 Description 76
11.2.2 Arguments 76
11.2.3 Output 77
11.2.4 Examples 77
11.3 Function Bisec(a, b, Fbis, [Tol]) 78
11.3.1 Description 78
11.3.2 Arguments 78
11.3.3 Output 79
11.3.4 Examples 79

12 MODULE Fourier **81**
12.1 Type Fourier_Serie 81
12.1.1 Description 81
12.1.2 Components 81
12.1.3 Examples 81
12.2 Type Fourier_Serie_2D 82
12.2.1 Description 82
12.2.2 Components 82
12.2.3 Examples 82
12.3 Assignment 82
12.3.1 Description 82
12.3.2 Examples 82
12.4 Operator + ... 83
12.4.1 Description 83
12.4.2 Examples 83
12.5 Operator - 84
12.5.1 Description 84
12.5.2 Examples 84
12.6 Operator * 84
12.6.1 Description 84
12.6.2 Examples 84
12.7 Operator ** 85
12.7.1 Description 85
12.7.2 Examples 85
12.8 Subroutine Init_Serie(FS,Ns) 86
12.8.1 Description 86
12.8.2 Arguments 86
12.8.3 Examples 86
12.9 Function Eval_Serie(FS, X, [Y], Tx, [Ty]) 87
12.9.1 Description 87
12.9.2 Arguments 87

12.9.3 Output . . . 87
12.9.4 Examples . . . 87
12.10 Function `Unit(FS, Ns)` . . . 88
12.10.1 Description . . . 88
12.10.2 Arguments . . . 88
12.10.3 Examples . . . 88
12.11 Function `DFT(Data, Is)` . . . 89
12.11.1 Description . . . 89
12.11.2 Arguments . . . 89
12.11.3 Output . . . 89
12.11.4 Examples . . . 89
12.12 Function `Conjg(FS)` . . . 90
12.12.1 Description . . . 90
12.12.2 Arguments . . . 90
12.12.3 Output . . . 90
12.12.4 Examples . . . 90
12.13 Subroutine `Save_Serie(FS, File)` . . . 91
12.13.1 Description . . . 91
12.13.2 Arguments . . . 91
12.13.3 Examples . . . 91
12.14 Subroutine `Read_Serie(FS, File)` . . . 91
12.14.1 Description . . . 91
12.14.2 Arguments . . . 92
12.14.3 Examples . . . 92

13 MODULE `Time` **93**
13.1 `Type tm` . . . 93
13.1.1 Description . . . 93
13.1.2 Components . . . 93
13.1.3 Example . . . 93
13.2 Function `gettime()` . . . 94
13.2.1 Description . . . 94
13.2.2 Arguments . . . 94
13.2.3 Output . . . 94
13.2.4 Example . . . 94
13.3 Function `isleap(Nyr)` . . . 95
13.3.1 Description . . . 95
13.3.2 Arguments . . . 95
13.3.3 Output . . . 95
13.3.4 Example . . . 95
13.4 Function `asctime(t)` . . . 95
13.4.1 Description . . . 95
13.4.2 Arguments . . . 96
13.4.3 Output . . . 96
13.4.4 Example . . . 96
13.5 Function `Day_of_Week(Day, Month, Year)` . . . 96
13.5.1 Description . . . 96

13.5.2 Arguments 96
13.5.3 Output 96
13.5.4 Example 97

GNU Free Documentation License **99**
1. APPLICABILITY AND DEFINITIONS 99
2. VERBATIM COPYING 101
3. COPYING IN QUANTITY 101
4. MODIFICATIONS 101
5. COMBINING DOCUMENTS 103
6. COLLECTIONS OF DOCUMENTS 103
7. AGGREGATION WITH INDEPENDENT WORKS 104
8. TRANSLATION 104
9. TERMINATION 104
10. FUTURE REVISIONS OF THIS LICENSE 104
ADDENDUM: How to use this License for your documents 105

Bibliography **107**

Index **109**

List of Tables

2.1 π-related real constants defined in the `MODULE constants`. 3
2.2 π-related complex constants defined in the `MODULE constants`. 4
2.3 Square roots and log related constants defined in the `MODULE constants`. 4
2.4 Other mathematical constants defined in the `MODULE constants`. 4

Listings

1.1 Definition of data types. 1
3.1 Standard error unit. 5
3.2 Print error message. 6
3.3 Print error message and stop a program. 6
4.1 Example of integration of a function using `Trpecio`. 10
4.2 Exmaple of integration of a function using `Simpson`. 11
4.3 Integrating a function using the open trapezoid rule. 13
4.4 Exmaple of integration using the open Simpson rule. 14
4.5 Integration of a function between 0 and ∞. 16
4.6 Integrating a function between $-\infty$ and 0. 17
4.7 Integrating functions with singularities in the upper limit. 19
4.8 Integrating functions with singularities in the lower limit. 21
4.9 Integrating differential equations with Euler. 23
4.10 Integrating differential equations with the Runge-Kutta method 25
5.1 Minimising a function. 28
5.2 Example of the usage of the routine `MaxPosition`. 30
6.1 Pivoting data of a matrix label 31
6.2 Making the LU decomposition. 33
6.3 Solving systems of linear equations. 34
6.4 Computing the determinant of a matrix. 35
7.1 Sorting data. 37
7.2 Sorting data. 38
7.3 Sorting data. 39
7.4 Searching data position in an ordered list. 40
8.1 Computing the Gamma Function. 43
8.2 Computing the Jacobi Theta functions. 44
8.3 Computing the Jacobi Theta functions with characteristics. 45
8.4 Computing the first 31 Hermite numbers. 46
8.5 Compute the Hermite functions. 47
8.6 Computing the bassi of a special Hilbert space (details in [2]). 48
8.7 Computing the factorial. 49
9.1 Computing the Mean of a vector of numbers. 51
9.2 Computing the Variance of a set of numbers. 52
9.3 Compputing the standard deviation. 53
9.4 Computing the $k^{\underline{th}}$ moment of a data set. 54
9.5 Obtaining numbers with a normal distribution. 54

9.6 Obtaining numbers with a Laplace distribution. 55
9.7 Making Histograms. 56
9.8 Doing linear regressions. 58
10.1 Defining a polynomial. 61
10.2 Defining a polynomial. 62
10.3 Assigning polynomials. 62
10.4 Adding polynomials. 63
10.5 Substracting polynomials. 64
10.6 Computing the product of two polynomials. 65
10.7 Initialising a polynomial data type. 66
10.8 Returns the degree of a polynomial. 67
10.9 Computes the values of a polynomial at some points. 68
10.10 Computing the derivative of a polynomial. 69
10.11 Computing the integral of a polynomial. 70
10.12 Compute values of the Interpolation polynomial. 72
10.13 Computes the interpolation polynomial. 72
10.14 Computes the cubic spline interpolation polynomial. 73
11.1 Computing roots of polynomials. 75
11.2 Computing roots of non-linear functions with the Newton method. 77
11.3 Computing roots with the bisection method. 79
12.1 Defining a Fourier serie. 81
12.2 Defining a two-dimensional Fourier serie. 82
12.3 Assigning Fourier series. 82
12.4 Adding Fourier series. 83
12.5 Subtracting Fourier series. 84
12.6 Computing the convolution of Fourier series. 84
12.7 ”Exponentiating” Fourier series. 85
12.8 Initialising a Fourier series. 86
12.9 Evaluating a Fourier series at a point. 87
12.10 Obtaining a constant Fourier series. 88
12.11 Computing the Discrete Fourier Transform. 89
12.12 Computing the Conjugate Fourier Series. 90
12.13 Saving a Fourier Serie in a file. 91
12.14 Reading a Fourier serie from a file. 92
13.1 Defining a Time data type. 93
13.2 Obtaining the current date and time. 94
13.3 Are we in a leap year?. 95
13.4 Printing current date/time. 96
13.5 Day of week of the first of January 1900. 97

Generalities

This is the documentation of a total of thirteen `FORTRAN 90` modules with different utilities. This code is well documented, and can be useful for several people, although the idea is *not* to produce fast, high performance code, but to have nice data structures and `INTERFACE` definitions so that complex problems can be solved fast, writing only a couple of lines of code.

The code of all these modules is *free software*, this means that you can redistribute and/or modify all the code under the terms of the GNU General Public License[1] as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. Note that the code is distributed in the hope that it will be useful, but **without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose**. See the GNU General Public License for more details.

The code has been written using standard `FORTRAN 90`, this means that it should run on any machine and with any compiler. In particular the code of all these modules has been compiled using `GNU gfortran`, `INTEL ifort` and `DIGITAL f90` for PC.

This manual is distributed under the GNU Free Documentation License. This means that you can copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

The source code of all the modules as well as the last version of this document should always be available (in it's last version) at:

`http://lattice.ft.uam.es/perpag/alberto/codigo_en.php`

there is also a `sourceforge.net` project, where the last version of both the source code and the documentation should be available:

`http://sourceforge.net/projects/afnl`

Enjoy programming.

Installation

To install this library in a Unix/Linux environment, simply edit the `Makefile` file, and set the `F90` and `F90OPT` variables to whatever your compiler and your favourite optimisation flags are. After running `make` you should obtain a file called `libf90.a`, and probably (that depends on

[1] `http://www.gnu.org/copyleft/gpl.html`

the particular compiler) some `.mod` files. Copy the `libf90.a` library and the `.mod` files to any place you like, and compile and link your program to that files. With `GNU gfortran` this is done using the flags `-I<path> -L<path> -lf90`, where `<path>` has to be substituted by the path you have chosen.

In other environments, you should ask the local guru/administrator about how to generate a library. In particular in a Windows environment the best option is to repartition you hard drive, eliminate Windows and install any Unix like free operating system, like Linux or FreeBSD.

One

MODULE NumTypes

This is the documentation of the MODULE NumTypes, that contaions the definition of Single Precision, and Double Precision data. All the other numerical modules uses this data type definitions.

1.1 Description

The MODULE NumTypes provides the definition of the Single Precision and Double Precision real and complex data in a potable way. When we want to define a single precision real we *will* do it with a statement like Real (kind=DP), instead of Real (kind=4). What we mean with DP is defined in this module. The different data types are:

SP: Single precision real.

DP: Double precision real.

SPC: Single precision complex.

DPC: Double precision complex.

To make all the code as portable as possible, all the data definitions should make use of this module.

1.2 Examples

Here we will define A as a single precision real, D as a double precision real, Ac as a single precision complex, and Dc as a double precision complex.

Listing 1.1: Definition of data types.

```
Program Types_of_Data
USE NumTypes


Real (kind=SP) :: A
Real (kind=DP) :: D
```

```
Complex (kind=SPC) :: Ac
Complex (kind=DPC) :: Dc

Write(*,*)Kind(A), Kind(Aa)


End Program Types_of_Data
```

Two

MODULE Constants

This is the documentation of the MODULE Constants, that contains the definition of the most used mathematical constants. This module uses numerical types defined in the MODULE NumTypes.

2.1 Name conventions

All the real simple precision constants ends with _SP, the real double precision constants with _DP, the complex simple precision with _SPC and the complex double precision with _DPC.

If a there exist a real or complex constant of simple precison defined, then it exist other with the same name (except for the sufix) of double precision and viceversa.

2.2 π-related constants

2.2.1 Real

The complex π-related defined in this module and its values can be seen in the table (2.1)

SP Name	DP Name	Value
PI_SP	PI_DP	π
TWOPI_SP	TWOPI_DP	2π
HALFPI_SP	HALFPI_DP	$\frac{\pi}{2}$

Table 2.1: π-related real constants defined in the MODULE constants.

2.2.2 Complex

The complex π-related defined in this module and its values can be seen in the table (2.2)

2.3 Square roots and $\log$ related constants

We have only real constants defined here. We can see a list of names-vlues in the table (2.3)

SPC Name	**DPC Name**	**Value**
UNITIMAG_SPC	UNITIMAG_DPC	ι
PI_IMAG_SPC	PI_IMAG_DPC	$\pi\iota$
TWOPI_IMAG_SPC	TWOPI_IMAG_DPC	$2\pi\iota$
HALFPI_IMAG_SPC	HALFPI_IMAG_SDC	$\frac{\pi}{2}\iota$

Table 2.2: π-related complex constants defined in the MODULE constants.

SP Name	**DP Name**	**Value**
SR2_SP	SR2_DP	$\sqrt{2}$
SR3_SP	SR3_DP	$\sqrt{3}$
SRe_SP	SRe_DP	$\sqrt{e}$
SRpi_SP	SRpi_DP	$\sqrt{\pi}$
LG102_SP	LG102_DP	$\log_{10} 2$
LG103_SP	LG103_DP	$\log_{10} 3$
LG10e_SP	LG10e_DP	$\log_{10} e$
LG10pi_SP	LG10pi_DP	$\log_{10} \pi$
LGe2_SP	LGe2_DP	$\log_e 2$
LGe3_SP	LGe3_DP	$\log_e 3$
LGe10_SP	LGe10_DP	$\log_e 10$

Table 2.3: Square roots and log related constants defined in the MODULE constants.

2.4 Other mathematical constants

In this section we have only the Euler γ constant. We can see the name-value pair in the table (2.4)

SP Name	**DP Name**	**Value**
GEULER_SP	GEULER_DP	$\gamma (= 0.5772\dots)$

Table 2.4: Other mathematical constants defined in the MODULE constants.

Three

MODULE Error

This is the documentation of the MODULE Error, a set of FORTRAN 90 routines that allow to write errors.

3.1 Defined variables

3.1.1 stderr

Description

This variable has the unit number of standard error.

Examples

Listing 3.1: Standard error unit.

```
Program Test
  USE Error

  Write(stderr,*)''This is printed in standard error.''

  Stop
End Program Test
```

3.2 Subroutine perror([routine], msg)

3.2.1 Description

Prints the error message msg in standard error. If the optional argument routine is given, it is used as the routine where the program has crashed.

3.2.2 Arguments

routine: Character string with arbitrary length. It should be the routine or program name where the error has ocurred. It is an optional argument.

msg: Character string with arbitrary length. It should be the message that you want to print.

3.2.3 Examples

Listing 3.2: Print error message.

```
Program Test
  USE Error

  Integer :: N1, N2

  Write(*,*)'Two integer numbers:'
  Read(*,*)N1,N2

  If (N2 == 0) Then
    CALL Perror('Test', 'Division by cero. See the product: ')
    Write(*,*)N1*N2
  Else
    Write(*,*)N1/N2
  End If

  Stop
End Program Test
```

3.3 Subroutine `abort([routine], msg)`

3.3.1 Description

Prints the error message `msg` in standard error, and stops the program. If the optional argument `routine` is given, it is used as the routine where the program has crashed.

3.3.2 Arguments

routine: Character string with arbitrary length. It should be the routine or program name where the error has ocurred. It is an optional argument.

msg: Character string with arbitrary length. It should be the message that you want to print.

3.3.3 Examples

Listing 3.3: Print error message and stop a program.

```
Program Test
  USE Error

  Integer :: N1, N2

  Write(*,*)'Two integer numbers:'
  Read(*,*)N1,N2

  If (N2 == 0) Then
```

```
    CALL abort('Test', 'Division by cero')
  Else
    Write(*,*)N1/N2
  End If

  Stop
End Program Test
```

Four

MODULE Integration

This is the documentation of the `MODULE Integration`, a set of `FORTRAN 90` routines that performs numerical integration and solves the initial value problem for a specified system of first-order ordinary differential equations. This module make use of the `MODULE NumTypes`, so please read the documentation of this module *before* reading this.

4.1 Function `Trapecio(a, b, Func, [Tol])`

4.1.1 Description

Calculates the integral of the function `Func` between `a` and `b` with precision `Tol` (optional) using the trapezoid rule.

4.1.2 Arguments

`a, b`: Real single or double precision. The limits of the integral.

`Func`: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
   End Function Fint
End Interface
```

`Tol`: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is `Tol` = 0.01.

4.1.3 Output

If the arguments are real of single (double) precision, the result will also be a real of single (double) precision. The value of the integral.

4.1.4 Examples

Listing 4.1: Example of integration of a function using Trpecio.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fint
     End Function Fint
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of x**2 between 0 and 1:'
  Write(*,*)Trapecio(0.0_DP, 1.0_DP, Fint, Tol)

  Stop
End Program Test

! ***********************************************
! *
Function Fint(X)
! *
! ***********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

  Fint = X**2

  Return
End Function Fint
```

4.2 Function Simpson(a, b, Func, [Tol])

4.2.1 Description

Calculates the integral of the function Func between a and b with precision Tol (optional) using the Simpson's rule.

In general this routine is better than Trapecio.

4.2.2 Arguments

a, b: Real single or double precision. The limits of the integral.

Func: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
   End Function Fint
End Interface
```

Tol: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is Tol $= 0.01$.

4.2.3 Output

If the arguments are reals of single (double) precision, the result will also be a real of single (double) precision. The value of the integral.

4.2.4 Examples

Listing 4.2: Exmaple of integration of a function using Simpson.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fint
     End Function Fint
```

```
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of x**2 between 0 and 1:'
  Write(*,*)Simpson(0.0_DP, 1.0_DP, Fint, Tol)

  Stop
End Program Test

! ********************************************
! *
Function Fint(X)
! *
! ********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

  Fint = X**2

  Return
End Function Fint
```

4.3 Function TrapecioAb(a, b, Func, [Tol])

4.3.1 Description

Calculates the integral of the function `Func` between `a` and `b` with precision `Tol` (optional) using the open trapezoid rule.

4.3.2 Arguments

`a, b`: Single (SP) or double (DP) precision. They are the limits of the integral.

`Func`: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
   End Function Fint
End Interface
```

Tol: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is Tol = 0.01.

4.3.3 Output

If the arguments are single (double) precision, the result will also be of single (double) precision. The value of the integral.

4.3.4 Examples

Listing 4.3: Integrating a function using the open trapezoid rule.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fint
     End Function Fint
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of x**2 between 0 and 1:'
  Write(*,*)TrapecioAb(0.0_DP, 1.0_DP, Fint, Tol)

  Stop
End Program Test

! ***********************************************
! *
Function Fint(X)
! *
! ***********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

  Fint = X**2

  Return
End Function Fint
```

4.4 Function SimpsonAb(a, b, Func, [Tol])

4.4.1 Description

Calculates the integral of the function `Func` between `a` and `b` with precision `Tol` (optional) using the open Simpson's rule.

In general better than `TrapecioAb`

4.4.2 Arguments

`a, b`: Single (SP) or double (DP) precision. They are the limits of the integral.

`Func`: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
   End Function Fint
End Interface
```

`Tol`: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is `Tol` $= 0.01$.

4.4.3 Output

If the arguments are single (double) precision, the result will also be of single (double) precision. The value of the integral.

4.4.4 Examples

Listing 4.4: Exmaple of integration using the open Simpson rule.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fint
     End Function Fint
```

```
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of x**2 between 0 and 1:'
  Write(*,*)SimpsonAb(0.0_DP, 1.0_DP, Fint, Tol)

  Stop
End Program Test

! ***********************************************
! *
Function Fint(X)
! *
! ***********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

  Fint = X**2

  Return
End Function Fint
```

4.5 Function SimpsonInfUp(a, Func, [Tol])

4.5.1 Description

Calculates the integral of the function Func between a and ∞ with precision Tol (optional) using the Simpson rule and a change of variables.

4.5.2 Arguments

a: Single (SP) or double (DP) precision. They are the limits of the integral.

Func: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
   End Function Fint
End Interface
```

This routine does not check if the integral exist, so the function must obviously decay fast for large x to obtain a finite value.

`Tol`: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is `Tol` = 0.01.

4.5.3 Output

If the arguments are single (double) precision, the result will also be of single (double) precision. The value of the integral.

4.5.4 examples

Listing 4.5: Integration of a function between 0 and ∞.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fint
     End Function Fint
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of e**(-x**2) between 0 and infinity:'
  Write(*,*)SimpsonInfUp(0.0_DP, Fint, Tol)

  Stop
End Program Test

! ***********************************************
! *
Function Fint(X)
! *
! ***********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

  Fint = exp(-X**2)

  Return
End Function Fint
```

4.6 Function SimpsonInfDw(a, Func, [Tol])

4.6.1 Description

Calculates the integral of the function Func between $-\infty$ and a with precision Tol (optional) using the Simpson rule and a change of variables.

4.6.2 Arguments

a: Single (SP) or double (DP) precision. They are the limits of the integral.

Func: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
   End Function Fint
End Interface
```

This routine does not check if the integral exist, so the function must obviously decay fast for large $-x$ to obtain a finite value.

Tol: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is Tol $= 0.01$.

4.6.3 Output

If the arguments are single (double) precision, the result will also be of single (double) precision. The value of the integral.

4.6.4 examples

Listing 4.6: Integrating a function between $-\infty$ and 0.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
```

```
      Real (kind=DP) :: Fint
    End Function Fint
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of e**(-x**2) between -infinity and 0:'
  Write(*,*)SimpsonInfDw(0.0_DP, Fint, Tol)

  Stop
End Program Test

! ***********************************************
! *
Function Fint(X)
! *
! ***********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

  Fint = exp(-X**2)

  Return
End Function Fint
```

4.7 Function `SimpsonSingUp(a, b, Func, [Tol], gamma)`

4.7.1 Description

Calculates the integral of the function `Func` between `a` and `b` with precision `Tol` (optional) using the Simpson's rule. The function may have an integrable singularity of the type:

$$f(x+b) \approx \frac{c}{(x-b)^\gamma} + \dots$$

with $0 < \gamma < 1$.

4.7.2 Arguments

`a, b`: Single (SP) or double (DP) precision. They are the limits of the integral.

`Func`: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes
```

```
          Real (kind=DP), Intent (in) :: X
          Real (kind=DP) :: Fint
        End Function Fint
     End Interface
```

Tol: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is `Tol` $= 0.01$.

gamma: The "degree of divergence" of the function in $x \approx b$.

4.7.3 Output

If the arguments are single (double) precision, the result will also be of single (double) precision. The value of the integral.

4.7.4 Examples

Listing 4.7: Integrating functions with singularities in the upper limit.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fint
     End Function Fint
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of 1/sqrt(-x) between -1 and 0:'
  Write(*,*)SimpsonSingUp(-1.0_DP, 0.0_DP, Fint, Tol, 0.5_DP)

  Stop
End Program Test

! **********************************************
! *
Function Fint(X)
! *
! **********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

```

```
  Fint = Sqrt(-X)

  Return
End Function Fint
```

4.8 Function SimpsonSingDw(a, b, Func, [Tol], gamma)

4.8.1 Description

Calculates the integral of the function Func between a and b with precision Tol (optional) using the Simpson's rule. The function may have an integrable singularity of the type:

$$f(x+a) \approx \frac{c}{(x-a)^\gamma} + \ldots$$

with $0 < \gamma < 1$.

4.8.2 Arguments

a, b: Single (SP) or double (DP) precision. They are the limits of the integral.

Func: The function to be integrated. It must be a function of only one argument of the same type as the function itself. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Fint(X)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
   End Function Fint
End Interface
```

Tol: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter, and the default is Tol $= 0.01$.

gamma: The "degree of divergence" of the function in $x \approx a$.

4.8.3 Output

If the arguments are single (double) precision, the result will also be of single (double) precision. The value of the integral.

4.8.4 Examples

Listing 4.8: Integrating functions with singularities in the lower limit.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol

  Interface
     Function Fint(X)
       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fint
     End Function Fint
  End Interface

  Tol = 1.0E-6_DP
  Write(*,*)'Integral of 1/sqrt(x) between 0 and 1:'
  Write(*,*)SimpsonSingDw(0.0_DP, 1.0_DP, Fint, Tol, 0.5_DP)

  Stop
End Program Test

! ***********************************************
! *
Function Fint(X)
! *
! ***********************************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fint

  Fint = Sqrt(X)

  Return
End Function Fint
```

4.9 Function Euler(Init, Xo, Xfin, Feuler, [Tol])

4.9.1 Description

Integrate the first order set of ODE defined by the function Feuler, with initial conditions given by the vector Init in Xo, until Xfin, with a precision given by Tol (optional).

A set of first order ODE's is given by the first derivatives of the variables involved:

$$\frac{\mathrm{d}y_i(x)}{\mathrm{d}x} = f_i(y_j, x)$$

and the initial conditions:

$$y_i(x_0)$$

After the integration we get:

$$y_i(x_{\rm fin})$$

So to define a set of first order ODE's we need the value of the derivative of the variable i in te point x (this is done by `Feuler`), a vector of initial conditions (`Init`) and the point where this initial conditions are defined (`Xo`), and finally the point where we want the solution (`Xfin`)

4.9.2 Arguments

`Init(:)`: Single (SP) or double (DP) precision vector of one dimension with the initial conditions.

`Xo`: Single (SP) or double (DP) precision. The point where the initial conditions are defined.

`Xfin`: Single (SP) or double (DP) precision. The point where we want the value of the functions.

`Feuler`: The function that defines the set of first order ODE's. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Feuler(X, Y) Result (Func)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X, Y(:)
     Real (kind=DP) :: Func(Size(Y))
   End Function Feuler
End Interface
```

The function must return a vector with the values of the first derivatives of the functions $y_i(x)$ in the point `X`.

`Tol`: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter.

4.9.3 Output

Real single or double precision (same as input) one dimensional array. The array contains the values of the functions y_i in the point `Xfin`.

4.9.4 Examples

This example below will integrate the set of first order ODE's defined by the equations:

$$\frac{{\rm d}y_1(x)}{{\rm d}x} = y_2(x); \qquad \frac{{\rm d}y_2(x)}{{\rm d}x} = -y_1(x)$$

whose solution is:

$$y_1(x) = A\cos(x) + B\sin(x)$$

With the initial conditions $y_1(0) = 0; y_2(0) = 1$, the solution is:

$$y_1(x) = \sin(x); \qquad y_2(x) = \cos(x)$$

so if we plot $y_1(1)$ and $y_2(1)$ we will obtain the values $\sin(1)$ and $y_2(1)$. In the following example, we will compare the result of integrating the differential equations with the exact values.

Listing 4.9: Integrating differential equations with Euler.

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol, In(2)

  Interface
    Function Feuler(X, Y) Result (Func)
      USE NumTypes

      Real (kind=DP), Intent (in) :: X, Y(:)
      Real (kind=DP) :: Func(Size(Y))
    End Function Feuler
  End Interface


  Tol = 1.0E-2_DP
  In(1) = 0.0_DP
  In(2) = 1.0_DP
  Write(*,*)'Values of sin(1) and cos(1): '
  Write(*,*)Euler(In, 0.0_DP, 1.0_DP, Feuler, Tol)
  Write(*,*)Sin(1.0_DP), Cos(1.0_DP)

  Stop
End Program Test

! **********************************************
! *                                            *
  Function FEuler(X, Y) Result (Func)
! *                                            *
! **********************************************

    Real (kind=8), Intent (in) :: X, Y(:)
    Real (kind=8) :: Func(Size(Y))

    Func(1) = Y(2)
    Func(2) = -Y(1)

    Return
  End Function FEuler
```

4.10 Function Rgnkta(Init, Xo, Xfin, Feuler, [Tol])

4.10.1 Description

Integrate the first order set of ODE defined by the function `Feuler`, with initial conditions given by the vector `Init` in `Xo`, until `Xfin`, with a precision given by `Tol` (optional). This method uses a Runge-Kutta algorithm and is much more exact than the previous `Euler` function.

A set of first order ODE's is given by the first derivatives of the variables involved:

$$\frac{\mathrm{d}y_i(x)}{\mathrm{d}x} = f_i(y_j, x)$$

and the initial conditions:

$$y_i(x_0)$$

After the integration we get:

$$y_i(x_{\mathrm{fin}})$$

So to define a set of first order ODE's we need the value of the derivative of the variable i in te point x (this is done by `Feuler`), a vector of initial conditions (`Init`) and the point where this initial conditions are defined (`Xo`), and finally the point where we want the solution (`Xfin`)

4.10.2 Arguments

`Init(:)`: Single (SP) or double (DP) precision vector of one dimension with the initial conditions.

`Xo`: Single (SP) or double (DP) precision. The point where the initial conditions are defined.

`Xfin`: Single (SP) or double (DP) precision. The point where we want the value of the functions.

`Feuler`: The function that defines the set of first order ODE's. If it is an external function an interface block like the following should be declared:

```
Interface
   Function Feuler(X, Y) Result (Func)
     USE NumTypes

     Real (kind=DP), Intent (in) :: X, Y(:)
     Real (kind=DP) :: Func(Size(Y))
   End Function Feuler
End Interface
```

The function is the same as in the previos function.

`Tol`: Single (SP) or double (DP) precision. An estimation of the desired accuracy of the result. It is an optional parameter.

4.10.3 Output

Real single or double precision (same as input) one dimensional array. The array contains the values of the functions y_i in the point `Xfin`.

4.10.4 Examples

This example below will integrate the set of first order ODE's defined by the equations:

$$\frac{\mathrm{d}y_1(x)}{\mathrm{d}x} = y_2(x); \qquad \frac{\mathrm{d}y_2(x)}{\mathrm{d}x} = -y_1(x)$$

whose solution is:

$$y_1(x) = A\cos(x) + B\sin(x)$$

With the initial conditions $y_1(0) = 0; y_2(0) = 1$, we have:

$$y_1(x) = \sin(x); \qquad y_2(x) = \cos(x)$$

so if we plot $y_1(1)$ and $y_2(1)$ we will obtain the values $\sin(1)$ and $y_2(1)$. In the following example, we will compare the values obtained with `Euler`, with `Rgnkta` and the exact ones.

Listing 4.10: Integrating differential equations with the Runge-Kutta method

```
Program Test
  USE NumTypes
  USE Integration

  Real (kind=DP) :: Tol, In(2)

  Interface
    Function Feuler(X, Y) Result (Func)
      USE NumTypes

      Real (kind=DP), Intent (in) :: X, Y(:)
      Real (kind=DP) :: Func(Size(Y))
    End Function Feuler
  End Interface


  Tol = 1.0E-3_DP
  In(1) = 0.0_DP
  In(2) = 1.0_DP
  Write(*,*)'Values of sin(1) and cos(1): '
  Write(*,*)'  Euler: '
  Write(*,*)Euler(In, 0.0_DP, 1.0_DP, Feuler, Tol)
  Write(*,*)'  Runge-Kutta: '
  Write(*,*)Rgnkta(In, 0.0_DP, 1.0_DP, Feuler, Tol)
  Write(*,*)'  Exact: '
  Write(*,*)Sin(1.0_DP), Cos(1.0_DP)

  Stop
End Program Test

! **********************************************
! *                                            *
  Function FEuler(X, Y) Result (Func)
! *                                            *
! **********************************************

```

```
    Real (kind=8), Intent (in) :: X, Y(:)
    Real (kind=8) :: Func(Size(Y))

    Func(1) = Y(2)
    Func(2) = -Y(1)

    Return
  End Function FEuler
```

Five

MODULE `Optimization`

This is the documentation of the MODULE `Optimization`, a set of routines to Optimise (maximise or minimise) functions of one or several variables. Lot of work is needed to improve this module (conjugate gradient, simplex, etc...).

5.1 Function `Step(X, FStep[, Tol])`

5.1.1 Description

The function `Step(X, FStep, Tol)` returns the position of the minimum of the Function `Fstep` with an optional precision `Tol`.

5.1.2 Arguments

`X`: Real single or double precision. An initial guess of the position of the minimum.

`Fstep`: The function that we want to minimise. It can be a function of one or several variables. In the case of one variable functions an interface like the following should be declared

```
Interface
   Function Fstep(Xo)
     USE NumTypes

     Real (kind=DP), Intent (in) :: Xo
     Real (kind=DP) :: Fstep
   End Function Fstep
End Interface
```

In the case of a function of several variables, the interface block should be like the following

```
Interface
   Function Fstep(Xo)
     USE NumTypes
```

```
        Real (kind=DP), Intent (in) :: Xo(:)
        Real (kind=DP) :: Fstep
      End Function Fstep
   End Interface
```

Tol: Real single or double precision. As estimation of the precision of the result. The dault value is 10^{-3}.

5.1.3 Output

Real Single or double precision (same as the output). The position of a minimum of `Fstep`.

5.1.4 Example

Listing 5.1: Minimising a function.

```fortran
Program TestMin

  USE NumTypes
  USE Optimization

  Integer, Parameter :: Ndim = 4
  Real (kind=DP) :: XoM(Ndim), Xmin(Ndim)

  Interface
     Function FstepM(Xo)
       USE NumTypes

       Real (kind=DP), Intent (in) :: Xo(:)
       Real (kind=DP) :: FstepM
     End Function FstepM
  End Interface


  ! Initial guess of the position of the minimum
  XoM(1) = 1.373_DP
  XoM(2) = 1.373_DP
  XoM(3) = 1.373_DP
  XoM(4) = 1.373_DP


  Write(*,*)'Initial Position: '
  Do I = 1, Ndim
     Write(*,'(1A,1I4,1A,1ES33.25)')'Variable ', I, " : ", XoM(I)
  End Do

  Xmin = Step(XoM, FstepM, 1.0E-7_DP)
  Write(*,*)
  Write(*,*)'Position of the minimum: '
  Do I = 1, Ndim
     Write(*,'(1A,1I4,1A,1ES33.25)')'Variable ', I, " : ", Xmin(I)
```

```
  End Do

  Stop
End Program TestMin

! *****************************
! *
Function FstepM(Xo)
! *
! *****************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: Xo(:)
  Real (kind=DP) :: FstepM


  FstepM = (Xo(1)-1.0_DP)**2 + &
       & (Xo(2)-2.0_DP)**2 + &
       & (Xo(3)+3.0_DP)**4 + &
       & (Xo(4)-4.0_DP)**8


  Return
End Function FstepM
```

5.2 Function `MaxPosition(FVal, IpX, IpY)`

5.2.1 Description

Given a two dimensional array of values `FVal(:,:)`, the function `MaxPosition(FVal, IpX, IpY)` returns the number of local maxima of `FVal(:,:)` and its positions in the one dimensional arrays `IpX(:)`and `IpY(:)`.

5.2.2 Arguments

`Fval(:,:)`: Real single or double precision. The values of a function in a two dimensional grid of points.

`IpX(:)`: Integer. A one dimensional array that contains the value of X for the positions of the maxima.

`IpY(:)`: Integer. A one dimensional array that contains the value of Y for the positions of the maxima.

5.2.3 Output

Integer. The number of local maxima of the input. `FVal(:,:)`.

5.2.4 Example

Listing 5.2: Example of the usage of the routine MaxPosition.

```
Program MaxLoc

  USE NumTypes
  USE Constants
  USE Optimization
  USE Error

  IMPLICIT NONE

  Integer :: I, J, IsX, IsY, Nmax
  Character (len=200) :: Filename
  Real (kind=DP) :: DnullX, DnullY
  Real (kind=DP), Allocatable :: F(:,:)
  Integer ::  IpX(10), IpY(10)

  Write(stderr,*)"SizeX, SizeY, Filename"
  Read(*,*)IsX, IsY, Filename

  Allocate(F(IsX,IsY))
  Open (Unit=666, File=Trim(Filename), Action="READ")
  Do I = 1, IsX
     Do J = 1, IsY
        Read(666,*)DnullX, DnullY, F(I,J)
        Write(stderr,*)DnullX, DnullY, F(I,J)
     End Do
  End Do
  Close(666)

  Nmax = MaxPosition(F, IpX, IpY)
  Write(*,*)"# Number of maxima: ", Nmax
  Write(*,*)"# Positions of the maxima: "
  Do I = 1, Nmax
     Write(*,*)IpX(I), IpY(I)
  End Do

  Stop
End Program MaxLoc
```

Six

MODULE Linear

This is the documentation of the `MODULE Linear`, a set of `FORTRAN 90` routines to solve linear systems of equations. This module make use of the `MODULE NumTypes`, and `MODULE Error` so please read the documentation of these modules *before* reading this.

6.1 Subroutine `Pivoting(M,Ipiv,Idet)`

6.1.1 Description

Permute the rows of M so that the biggest elements (in absolute value) of M are in the diagonal.

6.1.2 Arguments

`M(:,:)`: Real or complex single or double precision two dimensional array. Initially it contains the matrix to permute, after calling the routine, it contains the permuted matrix. *Note that M is overwritten when calling this routine.*

`Ipiv(:)`: Integer one dimensional array. It returns the permutation of rows made to M.

`Idet`: Integer. If the number of permutations is odd, `Idet` $= 1$, if it is even `Idet` $= -1$

6.1.3 Examples

Listing 6.1: Pivoting data of a matrix label

```
Program TestLinear

  USE NumTypes
  USE Linear

  Integer, Parameter :: Nord = 4

  Real (kind=DP) :: M(Nord,Nord), L(Nord,Nord), U(Nord,Nord), &
       & Mcp(Nord,Nord)
  Integer :: Ipiv(Nord), Iperm

```

```
  ! Fill M of random numbers
  CALL Random_Number(M)

  Write(*,*)'Original M: '
  Do I = 1, Nord
     Write(*,'(100ES10.3)')(M(I,J), J = 1, Nord)
  End Do

  CALL Pivoting (M, Ipiv, Iperm)
  Write(*,*)'Permuted M: '
  Do I = 1, Nord
     Write(*,'(100ES10.3)')(M(I,J), J = 1, Nord)
  End Do

  Stop
End Program TestLinear
```

6.2 Subroutine `LU(M, Ipiv, Idet)`

6.2.1 Description

Make the LU decomposition of matrix M. That is to say, given a matrix M, this function returns two matrix L and U, such that

$$M = LU \tag{6.1}$$

where L is lower triangular, and U upper triangular.

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots \\ L_{21} & 1 & 0 & \dots \\ L_{31} & L_{32} & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}; \quad U = \begin{pmatrix} U_{11} & U_{12} & U_{13} & \dots \\ 0 & U_{22} & U_{23} & \dots \\ 0 & 0 & U_{33} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{6.2}$$

The rows of M are permuted so that the biggest possible elements are on the diagonal (this makes the problem more stable). The two matrices L and U are returned *overwriting* M.

6.2.2 Arguments

`M(:,:)`: Real or complex single or double precision two dimensional array. Initially it contains the matrix to decompose, after calling the routine, it contains L in its lower part, and U in its upper part. *Note that M is overwritten when calling this routine.*

`Ipiv(:)`: Integer one dimensional array. It returns the permutation of rows made to M.

`Idet`: Integer. If the number of permutations is odd, `Idet` $= 1$, if it is even `Idet` $= -1$

6.2.3 Examples

Listing 6.2: Making the LU decomposition.

```
Program TestLinear

  USE NumTypes
  USE Linear

  Integer, Parameter :: Nord = 4

  Real (kind=DP) :: M(Nord,Nord), L(Nord,Nord), U(Nord,Nord), &
       & Mcp(Nord,Nord)
  Integer :: Ipiv(Nord), Iperm


  ! Fill M of random numbers, and make a copy
  CALL Random_Number(M)
  Mcp = M
  L = 0.0_DP
  U = 0.0_DP

  ! Make the LU decomposition and fill the matrices
  ! L and U
  CALL Lu(M, Ipiv, Iperm)
  Do I = 1, Nord
     L(I,I) = 1.0_DP
     U(I,I) = M(I,I)
     Do J = I+1, Nord
        L(J,I) = M(J,I)
        U(I,J) = M(I,J)
     End Do
  End Do

  ! Now Make the product and see that it is the original matrix with
  ! some rows permuted
  Write(*,*)'M: '
  Do I = 1, Nord
     Write(*,'(100ES10.3)')(Mcp(I,J), J = 1, Nord)
  End Do

  Write(*,*)'L: '
  Do I = 1, Nord
     Write(*,'(100ES10.3)')(L(I,J), J = 1, Nord)
  End Do
  Write(*,*)'U: '
  Do I = 1, Nord
     Write(*,'(100ES10.3)')(U(I,J), J = 1, Nord)
  End Do

 M = MatMul(L,U)
  Write(*,*)'LU (Same as M with some rows permuted): '
  Do I = 1, Nord
```

```
      Write(*,'(100ES10.3)')(M(I,J), J = 1, Nord)
   End Do


   Stop
End Program TestLinear
```

6.3 Subroutine LUsolve(M, b)

6.3.1 Description

Solves the linear system of equations

$$\begin{aligned} M_{11}x_1 + M_{12}x_2 + M_{13}x_3 + M_{14}x_4 + \ldots &= b_1 \\ M_{21}x_1 + M_{22}x_2 + M_{23}x_3 + M_{24}x_4 + \ldots &= b_2 \\ &\vdots \end{aligned} \tag{6.3}$$

6.3.2 Arguments

`M(:,:)`: Real or complex single or double precision two dimensional array. The matrix of coefficients. *M is overwritten when solving the system.*

`b(:)`: Real or complex single or double precision one dimensional array. The independent terms before calling the routine, and the solution of the linear system of equations after calling it. *Note that b is overwritten when calling this routine.*

6.3.3 Examples

Listing 6.3: Solving systems of linear equations.

```
Program TestLinear

   USE NumTypes
   USE Linear

   Integer, Parameter :: Nord = 10

   Real (kind=DP) :: M(Nord,Nord), L(Nord,Nord), U(Nord,Nord), &
        & Mcp(Nord,Nord), b(Nord), bcp(Nord)
   Integer :: Ipiv(Nord), Iperm

   ! Fill M and b of random numbers, and make a copy of both
   CALL Random_Number(M)
   CALL Random_Number(b)
   Mcp = M
   bcp = b

   ! Solve the linear system
   CALL LUsolve(M,b)
```

```
  ! Check that it is a solution:
  b = MatMul(Mcp,b)
  Write(*,*)'b: '
  Write(*,'(100ES10.3)')(Abs(bcp(I)-b(I)), I = 1, Nord)


  Stop
End Program TestLinear
```

6.4 Function Det(M)

6.4.1 Description

Computes the determinant of the matrix M.

6.4.2 Arguments

`M(:,:)`: Real or complex, simple or double precision two dimensional array. The matrix whose determinant we want to know.

6.4.3 Output

The value of the determinant. Same precision as the input argument.

6.4.4 Examples

Listing 6.4: Computing the determinant of a matrix.

```
Program TestLinear

  USE NumTypes
  USE Linear

  Integer, Parameter :: Nord = 10

  Real (kind=DP) :: M(Nord,Nord), L(Nord,Nord), U(Nord,Nord), &
       & Mcp(Nord,Nord), b(Nord), bcp(Nord)
  Integer :: Ipiv(Nord), Iperm


  ! Fill M of randoms numbers
  CALL Random_Number(M)

  ! Now compute the determinant.
  Write(*,'(ES15.8)')Det(M)


  Stop
End Program TestLinear
```

Seven

MODULE NonNum

This is the documentation of the `MODULE NonNum`, a set of `FORTRAN 90` routines to sort and search. This module make use of the `MODULE NumTypes`, and `MODULE Error` so please read the documentation of these modules *before* reading this.

7.1 Subroutine `Swap(X,Ind1,Ind2)`

7.1.1 Description

Swaps elements `Ind1` and `Ind2` of the array `X(:)`.

7.1.2 Arguments

`X(:)`: Integer, real single or real double precision one dimensional array. *Note that X is overwritten when calling this routine.*

`Ind1, Ind2`: Integer. The elements that we want to permute.

7.1.3 Examples

Listing 7.1: Sorting data.

```
Program TestNN

  USE NumTypes
  USE NonNumeric

  Integer, Parameter :: Nmax = 10
  Integer :: Ima(Nmax), I


  ! Fill Ima(:)
  Forall (I=1:Nmax) Ima(I) = I

  ! Plot the numbers
  Do I = 1, Nmax
     Write(*,'(1000I10)')Ima(I)
```

```
   End Do

   ! Swap first and last elemetns of Ima(:) and plot them.
   CALL Swap(Ima, 1, Nmax)
   Write(*,*)'# With first and last elements permuted: '
   Do I = 1, Nmax
      Write(*,'(1000I10)')Ima(I)
   End Do


   Stop
End Program TestNN
```

7.2 Subroutine Insrt(X[, Ipt])

7.2.1 Description

Sort the elements of `X(:)` in ascendant order[1].

7.2.2 Arguments

`X(:)`: Integer, real single or real double precision one dimensional array. Initially it contains unsorted numbers, and after calling the routine, it contains the sorted elements. *Note that X is overwritten when calling this routine.*

`Ipt(:)`: Integer vector, Optional. It returns the permutation made to `X(:)` to sort it.

7.2.3 Examples

Listing 7.2: Sorting data.

```
Program TestNN

   USE NumTypes
   USE NonNumeric

   Integer, Parameter :: Nmax = 10
   Integer :: Ima(Nmax)
   Real (kind=DP) :: X(Nmax), Y(Nmax)


   ! Fill X(:) with random data, and define Y(:)
   CALL Random_Number(X)
   Y = Sin(12.34_DP*(X-0.5_DP))

   ! Plot an unsorted data table
   Do I = 1, Nmax
      Write(*,'(1000ES13.5)')X(I), Y(I)
```

[1]This routine uses *insertion sort*, and is much slower than `Qsort`. To sort more than 10 elements, use `Qsort` unless you know what you are doing.

```fortran
  End Do

  ! Sort them, and plot the table again. Same points, but this time
  ! sorted
  CALL Insrt(X, Ima)
  Write(*,*)'# Again, this time sorted: '
  Do I = 1, Nmax
     Write(*,'(1000ES13.5)')X(I), Y(Ima(I))
  End Do


  Stop
End Program TestNN
```

7.3 Subroutine Qsort(X[, Ipt])

7.3.1 Description

Sort the elements of `X(:)` in ascendant order.

7.3.2 Arguments

`X(:)`: Integer, real single or real double precision one dimensional array. Initially it contains unsorted numbers, and after calling the routine, it contains the sorted elements. *Note that X is overwritten when calling this routine.*

`Ipt(:)`: Integer vector, Optional. It returns the permutation made to `X(:)` to sort it.

7.3.3 Examples

Listing 7.3: Sorting data.

```fortran
Program TestNN

  USE NumTypes
  USE NonNumeric

  Integer, Parameter :: Nmax = 10
  Integer :: Ima(Nmax)
  Real (kind=DP) :: X(Nmax), Y(Nmax)


  ! Fill X(:) with random data, and define Y(:)
  CALL Random_Number(X)
  Y = Sin(12.34_DP*(X-0.5_DP))

  ! Plot an unsorted data table
  Do I = 1, Nmax
     Write(*,'(1000ES13.5)')X(I), Y(I)
  End Do

```

```
  ! Sort them, and plot the table again. Same points, but this time
  ! sorted
  CALL Qsort(X, Ima)
  Write(*,*)'# Again, this time sorted: '
  Do I = 1, Nmax
     Write(*,'(1000ES13.5)')X(I), Y(Ima(I))
  End Do


  Stop
End Program TestNN
```

7.4 Function `Locate(X, X0[, Iin])`

7.4.1 Description

Given a *sorted* vector of elements `X(:)`, and a point X_0, `Locate` returns the position n such that $X(n) < X_0 < X(n+1)$. If X_0 is less than all the elements of `X(:)`, `Locate` returns 0, and if it is greater than all the elements of `X(:)`, it returns the number of elements of `X(:)`

7.4.2 Arguments

`X(:)`: Integer, real single or real double precision one dimensional *sorted* array.

X_0: Integer, real single or real double precision number, but the same type as `X(:)`. Point that we want to locate in the sorted vector `X(:)`.

`Iin`: Integer, Optional. Initial guess of the position.

7.4.3 Output

Integer. The position n such that

$$X(n) < X_0 < X(n+1)$$

7.4.4 Examples

Listing 7.4: Searching data position in an ordered list.

```
Program TestNN

  USE NumTypes
  USE NonNumeric

  Integer, Parameter :: Nmax = 100
  Integer :: Ima(Nmax), Idx
  Real (kind=DP) :: X(Nmax), Y(Nmax), X0


  ! Fill X(:) with random data, and set X0 to some arbitrary value.
  CALL Random_Number(X)
```

```
  X0 = 0.276546754_DP

  ! Sort X(:), find the position of X0, and plot the neighbtborr
  ! elements.
  CALL Qsort(X)
  Idx = Locate(X, X0)
  Write(*,'(1A,1ES33.25)')'Searched element: ', X0
  Write(*,'(1A,1ES33.25)')'Previous element in the list: ', X(Idx)
  Write(*,'(1A,1ES33.25)')'Next element in the list:     ', X(Idx+1)

  Stop
End Program TestNN
```

Eight

MODULE SpecialFunc

This is the documentation of the `MODULE SpecialFunc`, a set of `FORTRAN 90` routines to compute the value of some functions. This module make use of the `MODULE NumTypes`, `MODULE Constants`, `MODULE Error` so please read the documentation of these modules *before* reading this.

8.1 Function `GammaLn(X)`

8.1.1 Description

Compute $\log(\Gamma(X))$.

8.1.2 Arguments

`X`: Double (DP) precision. The point in which we want to know the value of $\Gamma(X)$.

8.1.3 Output

A real Double precision (DP).

8.1.4 Examples

This program should write the factorial of the first 100 numbers.

Listing 8.1: Computing the Gamma Function.

```
Program TestSpecialFunc

  USE NumTypes
  USE SpecialFunc

  Integer :: q


  Do q = 1, 100
     Write(*,'(1A13,1I4,1A3,1ES33.25)')'Factorial of:', q, ' = ',&
```

```
            & exp(GammaLn(Real(q+1,kind=DP)))
  End Do


  Stop
End Program TestSpecialFunc
```

8.2 Function Theta(i, z, tau[, Prec])

8.2.1 Description

Compute the value of the $i^{\underline{\text{th}}}$ Jacobi theta function ($i = 1, 2, 3, 4$) with nome $q = e^{i\pi\tau}$

$$\vartheta_i(z|\tau) \tag{8.1}$$

For a definition and properties of these functions take a look [1], here we will only say that following the conventions of the cited reference, our Theta functions have quasi-periods π and $\tau\pi$.

8.2.2 Arguments

i: Integer. Which theta function we want to compute. i must have one of the following values: $1, 2, 3, 4$.

z: Complex Double Precision (DPC) or Complex Single Precision (SPC). The point in which we want to compute the Theta function.

tau: Complex, with the same precision as `z`. is the quasi period of the Theta function. must be in the upper half plane ($\mathrm{I}m(\tau) > 0$).

Prec: Real, Optional. If `z` is DPC (SPC), `Prec` must be double precision (single precision). An estimation of the desired precision of the result. The default value is 1×10^{-3}

8.2.3 Output

If `z` is Double Precision Complex (SPC), the the result will be Double Precision Complex (SPC).

8.2.4 Examples

Listing 8.2: Computing the Jacobi Theta functions.

```
Program TestSpecialFunc

  USE NumTypes
  USE SpecialFunc

  Complex (DPC) :: Z, tau

```

```
  Z = Cmplx(0.546734, 2.76457643, kind=DPC)
  tau = Cmplx(0.0_DP, 3.76387540_DP)

  ! Check the quasi-periodicity of the Third
  ! Jacobi Theta function.
  Write(*,*)Theta(3, Z          , tau)
  Write(*,*)Theta(3, Z+Cmplx(PI_DP), tau)
  Write(*,*)Theta(3, Z+PI_DP*tau, tau) * &
       &exp(PI_IMAG_DPC*tau + 2.0_DP*UNITIMAG_DPC*Z)


  Stop
End Program TestSpecialFunc
```

8.3 Function ThetaChar(a, b, z, tau[, Prec])

8.3.1 Description

Computes the value of the Theta function with Characteristics (a, b) and quasi-periods $(\pi, \pi\tau)$ in the point z:

$$\vartheta \begin{bmatrix} a \\ b \end{bmatrix} (z|\tau) \tag{8.2}$$

8.3.2 Arguments

a, b: Complex or Real, Single or double precision. The two characteristics of the Theta function.

z: Complex (Single or Double precision). The point in the complex plane.

tau: Complex (Single or Double precision). The quasi-period of the theta function. Must have $(\mathrm{Im}(\tau) > 0)$.

Prec: Real (Single or Double precision). Optional. An estimation of the desired precision of the value of the theta function.

8.3.3 Output

Complex Single or Double precision, the same as the input values.

8.3.4 Examples

Listing 8.3: Computing the Jacobi Theta functions with characteristics.

```
Program TestSpecialFunc

  USE NumTypes
  USE SpecialFunc

  Real(kind=DP) :: Deriv, X1, X2
```

```
  Complex (DPC) :: Wmas, Wmenos, Z, tau
  Integer :: q, s


  Z = Cmplx(0.546734, 2.76457643, kind=DPC)
  tau = Cmplx(0.0_DP, 3.76387540_DP)


  Write(*,*)'Theta 1:'
  Write(*,*)Theta(1, Z, tau)
  Write(*,*)-ThetaChar(0.5_DP,0.5_DP, Z, tau)
  Write(*,*)'Theta 2:'
  Write(*,*)Theta(2, Z, tau)
  Write(*,*)ThetaChar(0.5_DP,0.0_DP, Z, tau)
  Write(*,*)'Theta 3:'
  Write(*,*)Theta(3, Z, tau)
  Write(*,*)ThetaChar(0.0_DP,0.0_DP, Z, tau)
  Write(*,*)'Theta 4:'
  Write(*,*)Theta(4, Z, tau)
  Write(*,*)ThetaChar(0.0_DP,0.5_DP, Z, tau)


  Stop
End Program TestSpecialFunc
```

8.4 Function `Hermite(n,x[, Dval])`

8.4.1 Description

Returns the value of the $n^{\underline{\text{th}}}$ Hermite polynomial in the point X. If `Dval` is specified, the value of the Derivative of the $n^{\underline{\text{th}}}$ Hermite polynomial in the point X is also returned.

8.4.2 Arguments

`n`: Integer. Which Hermite polynomial wants to compute.

`X`: Real (Single or Double precision). The point in which we want to compute the Polynomial.

`Dval`: Real (Single or Double precision). Optional. If specified, it stores the value of the Derivative of the Polynomials.

8.4.3 Output

Real single or double precision (same as input). The value of the $n^{\underline{\text{th}}}$ Hermite Polynomial in the point X.

8.4.4 Examples

Listing 8.4: Computing the first 31 Hermite numbers.

```
Program TestSpecialFunc
```

```

  USE NumTypes
  USE SpecialFunc

  Integer :: q


  Write(*,*)'The first 31 Hermite Numbers'
  Write(*,*)'http://www.research.att.com/~njas/sequences/A067994'
  Do q = 1, 31
     Write(*,'(1I4,1ES33.25)')q, Hermite(q, 0.0_DP)
  End Do

  Stop
End Program TestSpecialFunc
```

8.5 Function HermiteFunc(n, x[, Dval])

8.5.1 Description

Returns the value of the $n^{\underline{\text{th}}}$ Hermite function

$$\frac{1}{\sqrt{n!2^n\sqrt{\pi}}}e^{-x^2/2}H_n(x) \tag{8.3}$$

in the point X. If `Dval` is specified, the value of the Derivative of the $n^{\underline{\text{th}}}$ Hermite function in the point X is also returned.

8.5.2 Arguments

`n`: Integer. Which Hermite function wants to compute.

`X`: Real (Single or Double precision). The point in which we want to compute the Polynomial.

`Dval`: Real (Single or Double precision). Optional. If specified, it stores the value of the Derivative of the function.

8.5.3 Output

Real single or double precision (same as input). The value of the $n^{\underline{\text{th}}}$ Hermite function in the point X.

8.5.4 Examples

Listing 8.5: Compute the Hermite functions.

```
Program TestSpecialFunc

  USE NumTypes
  USE SpecialFunc

```

```
  Real(kind=DP) :: Deriv, X1, X2, Sum
  Complex (DPC) :: Wmas, Wmenos, Z, tau
  Integer :: q, s


  Write(*,*)'A (really bad) proof of orthonormality:'
  X1 = -10.0_DP
  Sum = 0.0_DP
  Do q = -1000, 1000
     Sum = Sum + HermiteFunc(6,X1)**2
     X1 = X1 + 0.01_DP
  End Do

  Write(*,'(1ES33.25)')Sum*0.01_DP

  Stop
End Program TestSpecialFunc
```

8.6 Function `Basis(X1, X2, n, s, q, itau[, Prec])`

8.6.1 Description

Return the value of the basis elements of the Hilbert space $\mathcal{H}_q$ of quasi-periodic functions

$$\mid n, s\rangle = e^{\imath \frac{f}{2} x_1 x_2} \sum_{k\in s+q\mathbb{Z}} e^{-u^2/2} H_n(u) e^{2\pi\imath k \frac{x_1}{l_1}} \qquad n = 0, \ldots \infty; s = 1, \ldots, q \tag{8.4}$$

defined in the appendix of [2] (look there for more details and properties).

8.6.2 Arguments

`X1,X2`: Real (Single or Double precision). The point in the Torus.

`n,s`: Integer. Specify which element of the basis.

`q`: Integer. Specify the Hilbert space $\mathcal{H}_q$.

`itau`: Real (Single or Double precision). Specify the ratio of quasi-periods: `itau` $= l_2/l_1$.

`Prec`: Real (Single or Double precision). Optional. An estimation of the desired precision.

8.6.3 Output

Complex single or double precision, depends of the input arguments.

8.6.4 Examples

Listing 8.6: Computing the bassi of a special Hilbert space (details in [2]).

```
Program TestSpecialFunc

```

```
  USE NumTypes
  USE SpecialFunc

  Real(kind=DP) :: X1, X2
  Complex (DPC) :: Wmas, Wmenos,
  Integer :: I, q, s


  Write(*,*)'Looking at the quasi-periodicity properties:'
  X1 = 0.97834D0
  X2 = 0.873873D0
  q = 4
  s = 3
  Do I = 0, 8
     Wmas   = Basis( X1,  X2+1.0_DP, I, s, q, 1.0_DP, 1.0D-15) * &
          & exp(PI_IMAG_DPC*X1*q)
     Wmenos = Basis( X1+1.0_DP,  X2, I, s, q, 1.0_DP, 1.0D-15) * &
          & exp(-PI_IMAG_DPC*X2*q)
     Write(*,'(1I3,2ES33.25)')I, Basis( X1,  X2, I, s, q, 1.0_DP, 1.0D-15)
     Write(*,'(1I3,2ES33.25)')I, Wmas
     Write(*,'(1I3,2ES33.25)')I, Wmenos
  End Do


  Stop
End Program TestSpecialFunc
```

8.7 Function Factorial(N)

8.7.1 Description

Compute $N!$. Better (faster and more accurate for small numbers) than the use of `GammaLn` to compute the factorial of a number.

8.7.2 Arguments

`N`: Integer. The number to compute the factorial.

8.7.3 Output

A real Double precision (DP).

8.7.4 Examples

This program should write the factorial of the first 100 numbers.

Listing 8.7: Computing the factorial.

```
Program TestSpecialFunc

  USE NumTypes
```

```
  USE SpecialFunc

  Integer :: q


  Do q = 1, 100
     Write(*,'(1A13,1I4,1A3,1ES33.25)')'Factorial of:', q, ' = ',&
          & Factorial(q)
  End Do


  Stop
End Program TestSpecialFunc
```

Nine

MODULE Statistics

This is the documentation of the `MODULE Statistics`, a set of `FORTRAN 90` routines to perform statistical description of data. This module make use of the `MODULE NumTypes`, `MODULE Constants`, `MODULE Error` and `MODULE Linear` so please read the documentation of these modules *before* reading this.

9.1 Function `Mean(X)`

9.1.1 Description

Compute the mean value of the numbers stored in `X(:)`.

9.1.2 Arguments

`X(:)`: Double (DP) or simple (SP) precision one dimensional array. The values whose mean we want to compute.

9.1.3 Output

A real double or simple precision (same type as the input). The mean of the values.

9.1.4 Examples

Listing 9.1: Computing the Mean of a vector of numbers.

```fortran
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 100, Npinta = 100, Npar = 4
  Real (kind=DP) :: X(Nmax), Y(Nmax), Yer(Nmax), &
       & Coef(Npar), Cerr(Npar), Corr, Xd(Nmax,2)

```

```
  CALL Random_Number(X)
  Write(*,'(ES33.25)')Mean(X)


  Stop
End Program Tests
```

9.2 Function Var(X)

9.2.1 Description

Compute the variance of a vector of numbers `X(:)`

9.2.2 Arguments

`X(:)`: Double (DP) or simple (SP) precision one dimensional array. The values whose variance we want to compute.

9.2.3 Output

A real double or simple precision (same type as the input). The variance of the values.

9.2.4 Examples

Listing 9.2: Computing the Variance of a set of numbers.

```
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 100, Npinta = 100, Npar = 4
  Real (kind=DP) :: X(Nmax), Y(Nmax), Yer(Nmax), &
       & Coef(Npar), Cerr(Npar), Corr, Xd(Nmax,2)


  CALL Random_Number(X)
  Write(*,'(ES33.25)')Var(X)


  Stop
End Program Tests
```

9.3 Function Stddev(X)

9.3.1 Description

Computes the standard deviation of the numbers stored in the vector `X(:)`.

9.3.2 Arguments

`X(:)`: Double (DP) or simple (SP) precision one dimensional array. The values whose standard deviation we want to compute.

9.3.3 Output

Real Single or Double precision, the same as the input values. The standard deviation of the values.

9.3.4 Examples

Listing 9.3: Compputing the standard deviation.

```
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 100, Npinta = 100, Npar = 4
  Real (kind=DP) :: X(Nmax), Y(Nmax), Yer(Nmax), &
       & Coef(Npar), Cerr(Npar), Corr, Xd(Nmax,2)


  CALL Random_Number(X)
  Write(*,'(ES33.25)')Stddev(X)


  Stop
End Program Tests
```

9.4 Function Moment(X, k)

9.4.1 Description

Returns the $k^{\underline{th}}$ moment of the values stored in the vector `X(:)`.

9.4.2 Arguments

`X(:)`: Real (Single or Double precision). The numbers whose $k^{\underline{th}}$ moment we want to compute.

`k`: Integer. Which moment we want to compute.

9.4.3 Output

Real single or double precision. The $k^{\underline{th}}$ moment of the numbers.

9.4.4 Examples

Listing 9.4: Computing the k^{th} moment of a data set.

```
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 100, Npinta = 100, Npar = 4
  Real (kind=DP) :: X(Nmax), Y(Nmax), Yer(Nmax), &
       & Coef(Npar), Cerr(Npar), Corr, Xd(Nmax,2)


  CALL Random_Number(X)
  Write(*,*)'We should obtain the same numbers twice: '
  Write(*,'(ES33.25)')Moment(X,2), Var(X)

  Stop
End Program Test
```

9.5 Subroutine Normal(X, [Rm], [Rsig])

9.5.1 Description

Fills `X(:)` with numbers from a normal distribution with mean `Rm`, and standard deviation `Rsig`. The parameters `Rm` and `Rsig` are optional. If they are not given the mean will be 0, and the standard deviation 1.

9.5.2 Arguments

X(:): Real (Single or Double precision) one dimensional array. A vector that will be filled with numbers according to the normal distribution.

Rm: Real (Single or Double precision), Optional. The mean of the normal distribution. If not present the default value is 0.

Rsig: Real (Single or Double precision), Optional. The standard deviation of the normal distribution. If not present the default value is 1.

9.5.3 Examples

Listing 9.5: Obtaining numbers with a normal distribution.

```
Program Tests

  USE NumTypes
  USE Error
  USE Statistics
```

```
  Integer, Parameter :: Nmax = 100
  Real (kind=DP) :: X(Nmax)


  CALL Normal(X, 1.23_DP, 0.345_DP)
  ! Now compute the mean and standard deviation of the data
  Write(*,*)'We should obtain 1.23 and 0.345: '
  Write(*,'(ES33.25)')Mean(X), Stddev(X)


  Stop
End Program Tests
```

9.6 Subroutine Laplace(X, Rm, Rb)

9.6.1 Description

Fills `X(:)` with numbers from a Laplace distribution with mean `Rm`, and variance $2\texttt{Rb}^2$.

9.6.2 Arguments

X(:): Real (Single or Double precision) one dimensional array. A vector that will be filled with numbers according to the normal distribution.

Rm: Real (Single or Double precision). The mean of the Laplace distribution.

Rb: Real (Single or Double precision). The width of the Laplace distribution (i.e. The variance is $2\texttt{Rb}^2$).

9.6.3 Examples

Listing 9.6: Obtaining numbers with a Laplace distribution.

```
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 100
  Real (kind=DP) :: X(Nmax)


  CALL Laplace(X, 1.23_DP, 1.0_DP)
  ! Now compute the mean and standard deviation of the data
  Write(*,*)'We should obtain 1.23 and sqrt(2): '
  Write(*,'(ES33.25)')Mean(X), Stddev(X)


  Stop
End Program Tests
```

9.7 Subroutine Histogram(Val, Ndiv, Ntics, Vmin, Vmax, h)

9.7.1 Description

Given a set of points Val(:), this routine makes Ndiv divisions between the minimum and the greatest value of Val (respectively returned in Vmin and Vmax), each of size h (also returned), and returns in the integer vector Nticks(:) the number of points that are in each interval.

9.7.2 Arguments

Val(:): Real (Single or Double precision) one dimensional array. The original values.

Ndiv: Integer. The number of divisions.

Nticks: Integer one dimensional array. Ndiv(I) Tells how many points of Val(:) are between $\texttt{Vmin} + (\texttt{I} - 1)\texttt{h}$ and $\texttt{Vmin} + \texttt{Ih}$.

Vmin, Vmax: Real (Single or Double precision). The minimum and maximum values of Val.

h: Real (Single or Double precision). After calling the routine has the step of the division.

9.7.3 Examples

Listing 9.7: Making Histograms.

```
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 500000, Npinta = 100, Npar = 4, Ndiv = 100
  Real (kind=DP) :: X(Nmax), Y(Nmax), Yer(Nmax), &
       & Coef(Npar), Cerr(Npar), Corr, Xd(Nmax,2), &
       & Xmin, Xmax, h, Xac
  Integer :: Ntics(Ndiv)

  CALL Normal(X, 1.23_DP, 0.345_DP)
  CALL Histogram(X, Ndiv, Ntics, Xmin, Xmax, h)

  Do I = 1, Ndiv
     Xac = Xmin + (I-1)*h
     Write(*,'(1ES33.25,1I)')Xac, Ntics(I)
  End Do

  Stop
End Program Tests
```

9.8 Subroutine LinearReg(X, Y, Yerr, [Func], Coef, Cerr, ChisqrV)

9.8.1 Description

Given a set of points X(:) and Y(:), this routine performs a linear fit to a set of functions defined by Func.

$$Y = \sum_i a_i f_i(X)$$

This routine also performs multi-dimensional fitting, in which case the points are specified as X(:,:), where the first argument tells which point, and the second which variable.

9.8.2 Arguments

X(:[,:]): Real single or double precision one dimensional array (for a one dimensional fit) or two dimensional array (for a multidimensional fit). The independent variables. For a multidimensional fit, the first argument tells which point, and the second which variable. So the size of the array should be X(Npoints,Ndim).

Y(:): Real single or double precision one dimensional array. The dependent variable.

Yerr(:): Real single or double precision one dimensional array. The errors of the points. If you don't have them, you should put all of hem to some non-zero value.

Func: Optional. This routine define the functions to fit. An interface like this should be provided

```
Interface
   Function Func(Xx, i)

     USE NumTypes

     Real (kind=SP), Intent (in) :: Xx
     Integer, Intent (in) :: i
     Real (kind=SP) :: Func

   End Function Func
End Interface
```

if you want to perform a one dimensional fitting, and like this

```
Interface
   Function Func(Xx, i)

     USE NumTypes

     Real (kind=SP), Intent (in) :: Xx(:)
     Integer, Intent (in) :: i
```

```
      Real (kind=SP) :: Func

    End Function Func
 End Interface
```

if it is a multidimensional fitting. Since you are making a fitting to a function of the type

$$Y = \sum_i a_i f_i(X)$$

the values $f_i(X)$ are given by this function as `Func(X, I)`. If the functions are not specified (i.e. you don't put this argument), a fit to a polynomial is made (this only work for one-dimensional fittings).

`Coef(:)`: Real single or double precision one dimensional array. The parameters that you want to determine.

`Cerr(:)`: Real single or double precision one dimensional array. The errors in the parameters.

`ChiSqr`: Real single or double precision. The χ^2 per degree of freedom of the fit.

9.8.3 Examples

Listing 9.8: Doing linear regressions.

```
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 200, Npinta = 100, Npar = 4, Ndiv = 100
  Real (kind=DP) :: X(Nmax), Y(Nmax), Yer(Nmax), &
       & Coef(Npar), Cerr(Npar), Corr, Xd(Nmax,2), &
       & Xmin, Xmax, h, Xac
  Integer :: Ntics(Ndiv)

  Interface
     Function Fd(Xx, i)

       USE NumTypes

       Real (kind=DP), Intent (in) :: Xx(:)
       Integer, Intent (in) :: i
       Real (kind=DP) :: Fd

     End Function FD
  End Interface


  CALL Random_Number(Xd)
  Xd(:,:) = 10.0_DP*(Xd(:,:) - 0.8_DP)
```

```fortran

  CALL Normal(Yer, 0.0_DP, 1.0E-3_DP)
  Y(:) = 12.34_DP*Xd(:,1)*sin(Xd(:,2)) - 2.23_DP + &
       & 0.67_DP*Xd(:,1)**2*Xd(:,2) +  0.23_DP*Xd(:,1) + Yer(:)


  CALL LinearReg(Xd, Y, Yer, Fd, Coef, Cerr, Corr)

  ! This should print the adjusted parameters,
  ! that have values: 12.34, -2.23, 0.67, 0.23
  Do I = 1, Npar
     Write(*,'(2ES33.25)')Coef(I), Cerr(I)
  End Do

  ! This prints the ChiSqr, that should be very
  ! close to 1.
  Write(*,'(1A,1ES33.25)')'ChiSqr of the Fit: ', Corr


  Stop
End Program Tests

! ***********************************
! *
Function Fd(X, i)
! *
! ***********************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X(:)
  Integer, Intent (in) :: i
  Real (kind=DP) :: Fd

  If (I==1) Then
     Fd = 1.0_DP
  Else If (I==2) Then
     Fd = X(1)*sin(X(2))
  Else If (I==3) Then
     Fd = X(1)**2*X(2)
  Else If (I==4) Then
     Fd = X(1)
  End If

  Return
End Function FD
```

Ten

MODULE `Polynomial`

This is the documentation of the `MODULE Polynomial`, a set of `FORTRAN 90` routines to work with polynomials. This module make use of the `MODULE NumTypes`, `MODULE Constants`, `MODULE Error` and `MODULE Linear` so please read the documentation of these modules *before* reading this.

10.1 `Type Pol`

10.1.1 Description

A new data type `Pol` is defined to work with polynomials. This type has two components: The coefficients of the polynomial, and the degree.

10.1.2 Components

`Coef(:)`: Real double precision one dimensional array.

`dg`: Integer. The degree of the polynomial.

10.1.3 Examples

A small example showing how to define a polynomial.

Listing 10.1: Defining a polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Type (Pol) :: P1

  Stop
End Program TestPoly
```

10.2 Type CmplxPol

10.2.1 Description

A new data type CmplxPol is defined to work with complex polynomials. This type has two components: The coefficients of the polynomial, and the degree.

All the routines, operators, etc... defined in this module works for complex as well as for real polynomials.

10.2.2 Components

Coef(:): Complex double precision one dimensional array.

dg: Integer. The degree of the polynomial.

10.2.3 Examples

A small example showing how to define a polynomial of complex coefficients.

Listing 10.2: Defining a polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Type (CmplxPol) :: P1

  Stop
End Program TestPoly
```

10.3 Assignment

10.3.1 Description

You can directly assign one defined polynomial (complex or real) to another, or to an array of real numbers, that are interpreted as the coefficients.

10.3.2 Examples

Listing 10.3: Assigning polynomials.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1)
```

```
  Type (Pol) :: Hermite4

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
  Hcoef(3) = -6.0_DP
  Hcoef(5) =  1.0_DP

  Hermite4 = Hcoef

  ! Now Show what we have in our data type:
  Do I = 0, Hermite4%dg
     Write(*,'(1I5,ES33.25)')I, Hermite4%Coef(I)
  End Do

  Stop
End Program TestPoly
```

10.4 Operator +

10.4.1 Description

You can naturally sum `Pol` or `CmplxPol` data types.

10.4.2 Examples

Listing 10.4: Adding polynomials.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1)
  Type (Pol) :: Hermite4, Hermite3, Sum

  ! The Third Hermite polynomial is x^3 - 3x, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(2) = -3.0_DP
  Hcoef(4) =  1.0_DP

  Hermite3 = Hcoef(1:4)

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
```

```
  Hcoef(3) =  -6.0_DP
  Hcoef(5) =   1.0_DP

  Hermite4 = Hcoef

  ! Now Add the two polynomials, and show the result.
  Sum = Hermite3 + Hermite4
  Do I = 0, Sum%dg
     Write(*,'(1I5,ES33.25)')I, Sum%Coef(I)
  End Do

  Stop
End Program TestPoly
```

10.5 Operator -

10.5.1 Description

You can subtract `Pol` or `CmplxPol` data types.

10.5.2 Examples

Listing 10.5: Substracting polynomials.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1)
  Type (Pol) :: Hermite4, Hermite3, Sum

  ! The Third Hermite polynomial is x^3 - 3x, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(2) = -3.0_DP
  Hcoef(4) =  1.0_DP

  Hermite3 = Hcoef(1:4)

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
  Hcoef(3) = -6.0_DP
  Hcoef(5) =  1.0_DP

  Hermite4 = Hcoef

```

```
  ! Now Subtract the two polynomials, and show the result.
  Sum = Hermite3 - Hermite4
  Do I = 0, Sum%dg
     Write(*,'(1I5,ES33.25)')I, Sum%Coef(I)
  End Do

  Stop
End Program TestPoly
```

10.6 Operator *

10.6.1 Description

You can naturally multiply `Pol` data types, `Pol` data types with double precision real numbers, `CmplxPol` data types and `CmplxPol` data types with real or complex numbers.

10.6.2 Examples

Listing 10.6: Computing the product of two polynomials.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1)
  Type (Pol) :: Hermite4, Hermite3, Sum

  ! The Third Hermite polynomial is x^3 - 3x, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(2) = -3.0_DP
  Hcoef(4) =  1.0_DP

  Hermite3 = Hcoef(1:4)

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
  Hcoef(3) = -6.0_DP
  Hcoef(5) =  1.0_DP

  Hermite4 = Hcoef

  ! Now multiply the two polynomials, and show the result.
  Sum = Hermite3 * Hermite4
  Do I = 0, Sum%dg
     Write(*,'(1I5,ES33.25)')I, Sum%Coef(I)
```

```
  End Do

  Stop
End Program TestPoly
```

10.7 Subroutine Init(P, Dgr)

10.7.1 Description

Allocate memory space for the coefficients of a Pol or a CmplxPol type.

10.7.2 Arguments

P: Type Pol or CmplxPol. The polynomial that you want to allocate space for.

Dgr: Integer. The degree of the polynomial.

10.7.3 Examples

Listing 10.7: Initialising a polynomial data type.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1)
  Type (Pol) :: Hermite4, Hermite3, Sum


  ! An alternative way of setting the third Hermite
  ! polynomial.
  CALL Init(Hermite3, 3)
  Hermite3%Coef(0) =  0.0_DP
  Hermite3%Coef(1) = -3.0_DP
  Hermite3%Coef(2) =  0.0_DP
  Hermite3%Coef(3) =  1.0_DP
  Hermite3%dg = 3


  Stop
End Program TestPoly
```

10.8 Function Degree(P)

10.8.1 Description

Returns the degree of the polynomial P.

10.8.2 Arguments

P: Type Pol or CmplxPol. The polynomial whose degree we want to know.

10.8.3 Output

Integer. The degree of the polynomial P.

10.8.4 Examples

Listing 10.8: Returns the degree of a polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1), X
  Type (Pol) :: Hermite4, Hermite3, Sum

  ! The Third Hermite polynomial is x^3 - 3x, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(2) = -3.0_DP
  Hcoef(4) =  1.0_DP

  Hermite3 = Hcoef(1:4)

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
  Hcoef(3) = -6.0_DP
  Hcoef(5) =  1.0_DP

  Hermite4 = Hcoef

  ! Now Mutiply the two polynomials, and show the result.
  Sum = Hermite3 * Hermite4

  ! Show the degree of the product. It should be 4+3=7.
  Write(*,*)Degree(Sum)


  Stop
End Program TestPoly
```

10.9 Function Value(P, X)

10.9.1 Description

Computes the value of the polynomial P in the point X.

10.9.2 Arguments

P: Type Pol or CmplxPol. The polynomial.

X: Real double precision if P is of type Pol and Complex double precision if P is CmplxPol. The point in which you want to compute the value.

10.9.3 Output

Real double precision. The value of the polynomial P in the point X.

10.9.4 Examples

Listing 10.9: Computes the values of a polynomial at some points.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1), X
  Type (Pol) :: Hermite4, Hermite3, Sum

  ! The Third Hermite polynomial is x^3 - 3x, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(2) = -3.0_DP
  Hcoef(4) =  1.0_DP

  Hermite3 = Hcoef(1:4)

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
  Hcoef(3) = -6.0_DP
  Hcoef(5) =  1.0_DP

  Hermite4 = Hcoef

  ! Now Mutiply the two polynomials, and show the result.
  Sum = Hermite3 * Hermite4

  ! Compute the valuye of the product in some point in two
```

```
  ! different ways.
  X = 9.34564_DP
  Write(*,'(ES33.25)')Value(Sum, X)
  Write(*,'(ES33.25)')Value(Hermite3, X)*Value(Hermite4, X)


  Stop
End Program TestPoly
```

10.10 Function Deriv(P)

10.10.1 Description

Computes the derivative of the polynomial `P`.

10.10.2 Arguments

P: Type `Pol` or `CmplxPol`. The polynomial whose derivative we want to compute.

10.10.3 Output

Type `Pol`. Another polynomial: the derivative of `P`.

10.10.4 Examples

Listing 10.10: Computing the derivative of a polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1), X
  Type (Pol) :: Hermite4, Hermite3, Res, Sum

  ! The Third Hermite polynomial is x^3 - 3x, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(2) = -3.0_DP
  Hcoef(4) =  1.0_DP

  Hermite3 = Hcoef(1:4)

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
  Hcoef(3) = -6.0_DP
  Hcoef(5) =  1.0_DP
```

```
  Hermite4 = Hcoef

  ! Now compute the derivative of Hermite4
  Res = Deriv(Hermite4)

  ! From the recursion relation of the Hermite polynomials
  ! we should obtain twwice the same number:
  X = 7.346582_DP
  Write(*,'(ES33.25)')Value(Res, X)
  Write(*,'(ES33.25)')4.0_DP*Value(Hermite3, X)


  Stop
End Program TestPoly
```

10.11 Function Integra(P[, Cte])

10.11.1 Description

Computes the integral of the polynomial P. If Cte is present then it is used as *integration constant*.

10.11.2 Arguments

P: Type Pol or CmplxPol. The polynomial whose integral we want to compute.

Cte: Optional. Real double precision if P is of type Pol and Complex double precision if P is CmplxPol. If not present, the default value is 0.

10.11.3 Output

Type Pol. Another polynomial: the integral of P.

10.11.4 Examples

Listing 10.11: Computing the integral of a polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4
  Real (kind=DP) :: Hcoef(Deg+1), X
  Type (Pol) :: Hermite4, Hermite3, Res, Sum

  ! The Third Hermite polynomial is x^3 - 3x, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
```

```
  Hcoef(2) = -3.0_DP
  Hcoef(4) =  1.0_DP

  Hermite3 = Hcoef(1:4)

  ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
  ! we first assign the values of the coefficients.
  Hcoef    =  0.0_DP
  Hcoef(1) =  3.0_DP
  Hcoef(3) = -6.0_DP
  Hcoef(5) =  1.0_DP

  Hermite4 = Hcoef

  ! Now compute the derivative of Hermite4
  Res = Integra(Hermite3, 3.0_DP/4.0_DP)

  ! From the recursion relation of the Hermite polynomials
  ! we should obtain twwice the same number:
  X = 7.346582_DP
  Write(*,'(ES33.25)')Value(Res, X)
  Write(*,'(ES33.25)')0.25_DP*Value(Hermite4, X)


  Stop
End Program TestPoly
```

10.12 Function InterpolValue(X, Y, Xo)

10.12.1 Description

Computes the value of the interpolation polynomial that pass trough (X(:), Y(:)) in the point Xo.

10.12.2 Arguments

X(:), Y(:): Real double precision or Complex double precision one dimensional arrays. Specify the points at which the interpolation polynomial should pass.

Xo: Same type as X(:) and Y(.). The point at which you want to compute the interpolation polynomial.

10.12.3 Output

Real double precision if input is real and complex double precision if input is complex. The value of the interpolation polynomial in Xo.

10.12.4 Examples

Listing 10.12: Compute values of the Interpolation polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial

  Integer, Parameter :: Deg = 4, Np = 7
  Real (kind=DP) :: Hcoef(Deg+1), X, Xp(Np), Yp(Np)
  Type (Pol) :: Hermite4, Hermite3, Res, Sum


  CALL Random_Number(Xp)
  Yp = 3.347234_DP*Xp - 2.475875_DP*Xp**3 - 7.23467_DP*Xp**4 + &
       & 1.47854_DP*Xp**6

  ! Now we compute the value of the interpolation polynomial
  ! at X, and compare it with the real value of the Polynomial
  X = -1.23899843_DP
  Write(*,'(ES33.25)')InterpolValue(Xp, Yp, X)
  Write(*,'(ES33.25)')3.347234_DP*X - 2.475875_DP*X**3 - &
       & 7.23467_DP*X**4 + 1.47854_DP*X**6


  Stop
End Program TestPoly
```

10.13 Function `Interpol(X, Y)`

Computes the interpolation polynomial that pass trough `(X(:), Y(:))`. **Note that using this function can be very unstable.**

10.13.1 Arguments

`X(:), Y(:)`: Real double precision or Complex double precision one dimensional arrays. Specify the points at which the interpolation polynomial should pass.

10.13.2 Output

Type `Pol` if input is real, and `CmplxPol` if input is complex. The interpolation polynomial.

10.13.3 Examples

Listing 10.13: Computes the interpolation polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial
```

```
  Integer, Parameter :: Deg = 4, Np = 7
  Real (kind=DP) :: Hcoef(Deg+1), X, Xp(Np), Yp(Np)
  Type (Pol) :: Hermite4, Hermite3, Res, Sum


  CALL Random_Number(Xp)
  Yp = 3.347234_DP*Xp - 2.475875_DP*Xp**3 - 7.23467_DP*Xp**4 + &
       & 1.47854_DP*Xp**6

  ! Now we compute the interpolation polynomial
  ! at X, and compare it with the real value of the Polynomial
  X = -1.23899843_DP
  Res = Interpol(Xp,Yp)
  Write(*,'(ES33.25)')Value(Res, X)
  Write(*,'(ES33.25)')3.347234_DP*X - 2.475875_DP*X**3 - &
       & 7.23467_DP*X**4 + 1.47854_DP*X**6


  Stop
End Program TestPoly
```

10.14 Subroutine `Spline(X, Y, Ypp0, YppN, Pols)`

10.14.1 Description

Compute the cubic spline interpolation polynomial that pass trough `(X(:), Y(:))`.

10.14.2 Arguments

`X(:), Y(:)`: Real double precision one dimensional arrays. Specify the points at which the cubic spline interpolation polynomial should pass.

`Ypp0, YppN`: The values of the second derivatives of the cubic spline interpolation polynomial in the first and last points.

`Pols(:)`: Type `Pol` one dimensional array. Returns the `N-1` cubic interpolation polynomials.

10.14.3 Examples

Listing 10.14: Computes the cubic spline interpolation polynomial.

```
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial
  USE NonNumeric

  Integer, Parameter :: Deg = 4, Np = 7
  Real (kind=DP) :: Hcoef(Deg+1), X, Xp(Np), Yp(Np)
```

```
  Type (Pol) :: Hermite4, Hermite3, Res, Sum, Spl(Np-1)


  CALL Random_Number(Xp)
  ! Order Xp
  CALL Qsort(Xp)
  Yp = 3.347234_DP*Xp - 2.475875_DP*Xp**3 - 7.23467_DP*Xp**4 + &
       & 1.47854_DP*Xp**6

  ! Now we compute the interpolation polynomial
  ! at X, and compare it with the real value of the Polynomial, and
  ! the value of the spline cubic interpolation polynomial.
  X = 0.23899843_DP
  Res = Interpol(Xp,Yp)
  CALL Spline(Xp, Yp, 0.0_DP, 0.0_DP, Spl)
  Write(*,'(ES33.25)')Value(Res, X)
  Write(*,'(ES33.25)')Value(Spl(Locate(Xp, X)), X)
  Write(*,'(ES33.25)')3.347234_DP*X - 2.475875_DP*X**3 - &
       & 7.23467_DP*X**4 + 1.47854_DP*X**6


  Stop
End Program TestPoly
```

Eleven

MODULE Root

This is the documentation of the `MODULE Root`, a set of `FORTRAN 90` routines to compute roots of functions. This module make use of the `MODULE NumTypes`, `MODULE Constants` and `MODULE Error` so please read the documentation of these modules *before* reading this.

11.1 Subroutine `RootPol(a, b, [c, d], z1, z2, [z3, z4])`

11.1.1 Description

Returns the complex roots of a polynomial of degree 2, 3 or 4.

11.1.2 Arguments

a, b, c, d: The coefficients of the polynomial. The meaning of the coefficieents a, b, c, d depends on the degree of the polynomial:

$$
\begin{aligned}
P(x) &= x^2 + \mathtt{a}x + \mathtt{b} \\
P(x) &= x^3 + \mathtt{a}x^2 + \mathtt{b}x + \mathtt{c} \\
P(x) &= x^4 + \mathtt{a}x^3 + \mathtt{b}x^2 + \mathtt{c}x + \mathtt{d}
\end{aligned}
$$

z1,z2,z3,z4: Complex simple or double precision. The roots of the polynomial.

11.1.3 Examples

Listing 11.1: Computing roots of polynomials.

```
Program TestRoot

  USE NumTypes
  USE Error
  USE Root

  Real (kind=DP) :: a, b, c, d
  Complex (kind=DPC) ::  z1, z2, z3, z4, ac, bc, cc, dc
```

```

   CALL Random_Number(a)
   CALL Random_Number(b)
   CALL Random_Number(c)
   CALL Random_Number(d)
   CALL RootPol(a,b,z1,z2)
   Write(*,'(3ES20.12)')Z1, Abs(z1**2 + a*z1 + Cmplx(b,kind=DPC))
   Write(*,'(3ES20.12)')Z2, Abs(z2**2 + a*z2 + Cmplx(b,kind=DPC))

   CALL RootPol(a,b,c, z1,z2, z3)
   Write(*,*)
   Write(*,'(3ES20.12)')Z1, Abs(z1**3+a*z1**2+b*z1+Cmplx(c,kind=DPC))
   Write(*,'(3ES20.12)')Z2, Abs(z2**3+a*z2**2+b*z2+Cmplx(c,kind=DPC))
   Write(*,'(3ES20.12)')Z3, Abs(z3**3+a*z3**2+b*z3+Cmplx(c,kind=DPC))

   ac = Cmplx(a,kind=DPC)
   bc = Cmplx(b,a,kind=DPC)
   cc = Cmplx(c,kind=DPC)
   dc = Cmplx(d,kind=DPC)
   CALL RootPol(ac,bc,z1,z2)
   Write(*,*)
   Write(*,'(3ES20.12)')Z1, Abs(z1**2 + ac*z1 + Cmplx(bc,kind=DPC))
   Write(*,'(3ES20.12)')Z2, Abs(z2**2 + ac*z2 + Cmplx(bc,kind=DPC))
   CALL RootPol(ac,bc,cc, dc, z1,z2, z3, z4)
   Write(*,*)
   Write(*,'(3ES20.12)')Z1, Abs(z1**4+ac*z1**3+bc*z1**2+cc*z1+dc)
   Write(*,'(3ES20.12)')Z2, Abs(z2**4+ac*z2**3+bc*z2**2+cc*z2+dc)
   Write(*,'(3ES20.12)')Z3, Abs(z3**4+ac*z3**3+bc*z3**2+cc*z3+dc)
   Write(*,'(3ES20.12)')Z4, Abs(z4**4+ac*z4**3+bc*z4**2+cc*z4+dc)


   Stop
End Program TestRoot
```

11.2 Function `Newton(Xo, Fnew, [Tol])`

11.2.1 Description

Compute a root of the function defined by the routine `Fnew`.

11.2.2 Arguments

`Xo`: Real simple or double precision. An initial guess of the position of the root.

`Fnew`: The function whose root we want to compute. It is defined as a subroutine that returns the value of the function and of its derivative. If it is an external function, an interface block like this should be defined

```
Interface
   Subroutine FNew(Xo, F, D)
```

```
      USE NumTypes

      Real (kind=DP), Intent (in) :: Xo
      Real (kind=DP), Intent (out) :: F, D
    End Subroutine FNew
  End Interface
```

where `F` is the value of the function in `Xo`, and `D` the value of the derivative in `Xo`. If the arguments are of simple precision, a similar interface should be provided, where the arguments of `Fnew` are of single precision.

`Tol`: Real single or double precision. Optional. An estimation of the desired accuracy of the position of the root.

11.2.3 Output

Real single or double precision. The position of the root.

11.2.4 Examples

Listing 11.2: Computing roots of non-linear functions with the Newton method.

```
Program TestRoot

  USE NumTypes
  USE Error
  USE Root

  Real (kind=DP) :: a, b, c, d, X
  Complex (kind=DPC) ::  z1, z2, z3, z4, ac, bc, cc, dc


  Interface
     Subroutine FNew(Xo, F, D)

       USE NumTypes

       Real (kind=DP), Intent (in) :: Xo
       Real (kind=DP), Intent (out) :: F, D
     End Subroutine FNew
  End Interface


  ! Compute the value such that cos(x) = x
  X = Newton(0.0_DP, Fnew, 1.0E-10_DP)
  Write(*,'(1A,ES33.25)')'Point:         ', X
  Write(*,'(1A,ES33.25)')'Value of Cos:  ', Cos(X)


  Stop
```

```
End Program TestRoot

! ********************************
! *
Subroutine FNew(Xo, F, D)
! *
! ********************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: Xo
  Real (kind=DP), Intent (out) :: F, D


  F = Xo - Cos(Xo)
  D = 1.0_DP   + Sin(Xo)

  Return
End Subroutine FNew
```

11.3 Function Bisec(a, b, Fbis, [Tol])

11.3.1 Description

Compute the root of the function defined by Fbis.

11.3.2 Arguments

a, b: Real single or double precision. Initial points, such that Fbis(a)Fbis(b) < 0.

Fbis: The function whose root we want to compute. It is defined as a function that returns the value of the function. If it is an external function, an interface block like this should be defined

```
Interface
   Function F(X)

     USE NumTypes

     Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: F
   End Function F
End Interface
```

where F is the value of the function in X. If the arguments are of simple precision, a similar interface should be provided, where the arguments of F are of single precision.

Tol: Real single or double precision. Optional. An estimation of the desired accuracy of the position of the root.

11.3.3 Output

Real single or double precision. The position of the root of `Fbis`.

11.3.4 Examples

Listing 11.3: Computing roots with the bisection method.

```fortran
Program TestRoot

  USE NumTypes
  USE Error
  USE Root

  Real (kind=DP) :: a, b, c, d, X
  Complex (kind=DPC) ::  z1, z2, z3, z4, ac, bc, cc, dc

  Interface
     Function Fbis(X)

       USE NumTypes

       Real (kind=DP), Intent (in) :: X
       Real (kind=DP) :: Fbis
     End Function Fbis
  End Interface

  ! Compute the value such that cos(x) = x
  X = Bisec(0.0_DP, 1.1_DP, Fbis, 1.0E-10_DP)
  Write(*,'(1A,ES33.25)')'Point:         ', X
  Write(*,'(1A,ES33.25)')'Value of Cos: ', Cos(X)


  Stop
End Program TestRoot

! ********************************
! *
Function FBis(X)
! *
! ********************************

  USE NumTypes

  Real (kind=DP), Intent (in) :: X
  Real (kind=DP) :: Fbis


  Fbis = X - Cos(X)

  Return
End Function FBis
```

Twelve

MODULE Fourier

This is the documentation of the `MODULE Fourier`, a set of `FORTRAN 90` routines to work with Fourier series. This module make use of the `MODULE NumTypes` and the `MODULE Constants` so please read the documentation of these modules *before* reading this.

12.1 Type Fourier_Serie

12.1.1 Description

A new data type `Fourier_Serie` is defined to work with Fourier series. This type has two components: The modes, and the number of modes.

12.1.2 Components

`Coef(:)`: Complex double precision one dimensional array. The modes.

`Nterm`: Integer. The number of terms of the Fourier series.

12.1.3 Examples

A small example showing how to define a Fourier serie.

Listing 12.1: Defining a Fourier serie.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie) :: Ff

  Stop
End Program TestPoly
```

12.2 Type Fourier_Serie_2D

12.2.1 Description

A new data type `Fourier_Serie_2D` is defined to work with two dimensional Fourier series. This type has two components: The modes, and the number of modes.

12.2.2 Components

`Coef(:,:)`: Complex double precision two dimensional array. The modes.

`Nterm`: Integer. The number of terms of the Fourier series.

12.2.3 Examples

A small example showing how to define a polynomial.

Listing 12.2: Defining a two-dimensional Fourier serie.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie_2D) :: Ff

  Stop
End Program TestPoly
```

12.3 Assignment

12.3.1 Description

You can directly assign one defined Fourier series (one or two dimensional) to another.

12.3.2 Examples

This example uses the `Init_Serie` subroutine. For details of the usage of this function look at the section (12.8), page (86).

Listing 12.3: Assigning Fourier series.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie) :: FS1, FS2

  CALL Init_Serie(FS1, 20)
  CALL Init_Serie(FS2, 20)
```

```
  FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  FS2 = FS1

  Write(*,'(2ES33.25)')FS2%Coef( 1)
  Write(*,'(2ES33.25)')FS2%Coef(-1)

  Stop
End Program TestFourier
```

12.4 Operator +

12.4.1 Description

You can naturally sum one or two dimensional Fourier series. If they have different sizes, it is assumed that the non defined modes of the short Fourier Series are zero.

12.4.2 Examples

This example uses the `Init_Serie` subroutine. For details of the usage of this function look at the section (12.8), page (86).

Listing 12.4: Adding Fourier series.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie_2D) :: FS1, FS2, FS3
  Integer :: Nt

  Nt = 4
  CALL Init_Serie(FS1, Nt)
  CALL Init_Serie(FS2, Nt)

  FS1%Coef( 1,1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1,1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  FS2%Coef( 1,1) = Cmplx(-1.0_DP, 4.5_DP, kind=DPC)
  FS2%Coef(-1,1) = Cmplx(-1.0_DP, -6.78745_DP, kind=DPC)


  FS3 = FS1 + FS2
  Write(*,'(2ES33.25)')FS3%Coef( 1,1)
  Write(*,'(2ES33.25)')FS3%Coef(-1,1)

  Stop
End Program TestFourier
```

12.5 Operator -

12.5.1 Description

You can naturally subtract one or two dimensional Fourier series. If they have different sizes, it is assumed that the non defined modes of the short Fourier Series are zero.

12.5.2 Examples

Listing 12.5: Subtracting Fourier series.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie) :: FS1, FS2, FS3
  Integer :: Nt

  Nt = 4
  CALL Init_Serie(FS1, Nt)

  FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  FS2 = FS1

  FS3 = FS1 - FS2
  Write(*,'(2ES33.25)')FS3%Coef( 1)
  Write(*,'(2ES33.25)')FS3%Coef(-1)

  Stop
End Program TestFourier
```

12.6 Operator *

12.6.1 Description

You can naturally multiply one or two dimensional Fourier series, in which case the convolution of the Fourier Modes is performed. If they have different sizes, it is assumed that the non defined modes of the short Fourier Series are zero.

12.6.2 Examples

Listing 12.6: Computing the convolution of Fourier series.

```
Program TestFourier

  USE NumTypes
  USE Constants
```

```
  USE Fourier

  Type (Fourier_Serie) :: FS1, FS2, FS3
  Integer :: Nt

  Nt = 4
  CALL Init_Serie(FS1, Nt)

  FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  FS2 = FS1

  FS3 = FS1 * FS2
  Write(*,'(2ES33.25)')FS3%Coef( 0)

  Stop
End Program TestFourier
```

12.7 Operator **

12.7.1 Description

You can naturally compute the integer power of a one or two dimensional Fourier series, in which case the convolution of the Fourier modes with themselves are performed a certain number of times.

12.7.2 Examples

Listing 12.7: "Exponentiating" Fourier series.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie) :: FS1, FS2, FS3
  Integer :: Nt

  Nt = 4
  CALL Init_Serie(FS1, Nt)
  CALL Init_Serie(FS2, Nt)
  CALL Init_Serie(FS3, Nt)

  FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  FS3%Coef(0) = Cmplx(1.0_DP, 0.0_DP, kind=DPC)

  FS2 = FS1**8
```

```
  Do I = 1, 8
     FS3 = FS3 * FS1
  End Do

  Write(*,'(2ES33.25)')FS2%Coef( 0)
  Write(*,'(2ES33.25)')FS3%Coef( 0)

  Stop
End Program TestFourier
```

12.8 Subroutine `Init_Serie(FS,Ns)`

12.8.1 Description

Allocate memory space for the modes of a one or two dimensional Fourier series.

12.8.2 Arguments

`FS`: Type `Fourier_Serie` or type `Fourier_Serie_2D`. The Fourier series that you want to allocate space for.

`Ns`: Integer. The number of modes.

12.8.3 Examples

Any of the examples of some of the previous sections are aldo good examples of the use of the `Init_Serie` subroutine. Here we simply repeat one of them.

Listing 12.8: Initialising a Fourier series.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie) :: FS1, FS2, FS3
  Integer :: Nt

  Nt = 4
  CALL Init_Serie(FS1, Nt)

  FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  FS2 = FS1

  FS3 = FS1 * FS2
  Write(*,'(2ES33.25)')FS3%Coef( 0)

  Stop
End Program TestFourier
```

12.9 Function Eval_Serie(FS, X, [Y], Tx, [Ty])

12.9.1 Description

Compute the value of the Fourier series FS with periods Tx,Ty at the point X,Y.

12.9.2 Arguments

FS: Type Fourier_Serie or type Fourier_Serie_2D. The Fourier series that you want to evaluate.

X,Y: Real double precision. The point in which you want to evaluate the Fourier series. If FS is a two dimensional Fourier series, then Y must be present.

Tx,Ty: Real double precision. The period(s). If FS is a two dimensional Fourier series, then Ty must be present.

12.9.3 Output

Real double precision. The value of the function defined by the modes in FS at the point (X[,Y]).

12.9.4 Examples

Listing 12.9: Evaluating a Fourier series at a point.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie) :: FS1, FS2, FS3
  Integer :: Nt

  Nt = 4
  CALL Init_Serie(FS1, Nt)
  CALL Init_Serie(FS2, Nt)
  CALL Init_Serie(FS3, Nt)


  FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  FS2 = FS1**2

  FS3 = FS1*FS2

  Write(*,'(2ES33.25)') Eval_Serie(FS1,0.12_DP,1.0_DP) * &
                       & Eval_Serie(FS2,0.12_DP,1.0_DP)
  Write(*,'(2ES33.25)') Eval_Serie(FS3,0.12_DP,1.0_DP)

```

```
  Stop
End Program TestFourier
```

12.10 Function Unit(FS, Ns)

12.10.1 Description

Allocate memory space for the modes of a one or two dimensional Fourier series and sets the zero mode equal to 1.

12.10.2 Arguments

FS: Type Fourier_Serie or type Fourier_Serie_2D. The Fourier series that you want to allocate space for.

Ns: Integer. The number of modes.

12.10.3 Examples

Listing 12.10: Obtaining a constant Fourier series.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie) :: FS1, FS2, FS3
  Integer :: Nt

  Nt = 4
  CALL Init_Serie(FS1, Nt)
  CALL Init_Serie(FS2, Nt)
  CALL Init_Serie(FS3, Nt)


  FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
  FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)

  CALL Unit(FS2, Nt)

  FS3 = FS1*FS2

  Write(*,'(2ES33.25)') Eval_Serie(FS1,0.12_DP,1.0_DP)
  Write(*,'(2ES33.25)') Eval_Serie(FS3,0.12_DP,1.0_DP)

  Stop
End Program TestFourier
```

12.11 Function DFT(Data, Is)

12.11.1 Description

Compute the Discrete Fourier Transform of the values stored in the complex array Data. If Is is present and is set to -1, the inverse Discrete Fourier Transform is performed. The direct Fourier transform is defined as

$$\tilde{f}(k) = \sum_{n=0}^{N} f_n e^{\frac{2\pi\imath n}{N}} \qquad \forall k \in \left[-\frac{N}{2}, \frac{N}{2}\right]$$

the inverse one is defined as

$$\tilde{f}(k) = \frac{1}{N}\sum_{n=0}^{N} f_n e^{\frac{-2\pi\imath n}{N}} \qquad \forall k \in \left[-\frac{N}{2}, \frac{N}{2}\right]$$

12.11.2 Arguments

Data(:[,:]): One or two dimensional double precision complex array. The data whose Discrete Fourier Transform we want to compute.

Is: Integer. Optional. A flag to tell if we want to compute the direct or the inverse Fourier transform.

12.11.3 Output

Type Fourier_Serie if Data(:) is one dimensional, and type Fourier_Serie_2D if Data(:,:) is two dimensional.

12.11.4 Examples

This example compute the discrete Fourier transform of $f(x_i) = \sin(x_i)$.

Listing 12.11: Computing the Discrete Fourier Transform.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Integer, Parameter :: Nmax=20
  Type (Fourier_Serie) :: FS1, FS2, FS3
  Complex (kind=DPC) :: Data(Nmax), X
  Integer :: Nt

  Do I = 1, Nmax
     X = Cmplx(TWOPI_DP*I/Nmax)
     Data(I) = Sin(X)
  End Do

  FS1 = DFT(Data)
```

```
  Write(*,'(1A,2ES33.25)')'Mode k= 1:  ', FS1%Coef( 1)
  Write(*,'(1A,2ES33.25)')'Mode k=-1:  ', FS1%Coef(-1)
  Write(*,'(ES33.25)' )Sum(Abs(FS1%Coef(:)))

  Stop
End Program TestFourier
```

12.12 Function `Conjg(FS)`

12.12.1 Description

Computes the Fourier modes that correspond to the conjugate function. This means: If the modes of `FS` are $\tilde{f}(k)$, this function returns a Fourier series with modes $\tilde{f}(-k)$.

12.12.2 Arguments

`FS`: Type `Fourier_Serie` or type `Fourier_Serie_2D`. The Fourier series whose conjugate you want to compute.

12.12.3 Output

Type `Fourier_Serie` if `FS` is of type `Fourier_Serie`, and type `Fourier_Serie_2D` if `FS` is of Type `Fourier_Serie_2D`.

12.12.4 Examples

Listing 12.12: Computing the Conjugate Fourier Series.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Integer, Parameter :: Nmax=20
  Type (Fourier_Serie) :: FS1, FS2, FS3
  Complex (kind=DPC) :: Data(Nmax), X
  Integer :: Nt

  Do I = 1, Nmax
     X = Cmplx(TWOPI_DP*I/Nmax,kind=DPC)
     Data(I) = Sin(X) + Cmplx(0.0_DP,I*2.0_DP,kind=DPC)
  End Do

  FS1 = DFT(Data)

  Write(*,'(2ES33.25)')Eval_Serie(FS1,0.23_DP,1.0_DP)
  Write(*,'(2ES33.25)')Eval_Serie(Conjg(FS1),0.23_DP,1.0_DP)
```

```
  Stop
End Program TestFourier
```

12.13 Subroutine Save_Serie(FS, File)

12.13.1 Description

Write the Fourier series FS to the file File.

12.13.2 Arguments

FS: Type Fourier_Serie or type Fourier_Serie_2D. The Fourier series that you want to store in a file.

File: Character string of arbitrary length. The name of the file in which you want to save FS.

12.13.3 Examples

Listing 12.13: Saving a Fourier Serie in a file.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Integer, Parameter :: Nmax=20
  Type (Fourier_Serie) :: FS1, FS2, FS3
  Complex (kind=DPC) :: Data(Nmax), X
  Integer :: Nt

  Do I = 1, Nmax
     X = Cmplx(TWOPI_DP*I/Nmax,kind=DPC)
     Data(I) = Sin(X) + Cmplx(0.0_DP,I*2.0_DP,kind=DPC)
  End Do

  FS1 = DFT(Data)

  CALL Save(FS1,'datamodes.dat')

  Stop
End Program TestFourier
```

12.14 Subroutine Read_Serie(FS, File)

12.14.1 Description

Reads the Fourier series FS stored in the file File.

12.14.2 Arguments

FS: Type `Fourier_Serie` or type `Fourier_Serie_2D`. The name of the Fourier series data type in which you want to store that data.

File: Character Character string of arbitrary length. The name of the file in which the saved series is.

12.14.3 Examples

Listing 12.14: Reading a Fourier serie from a file.

```fortran
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Integer, Parameter :: Nmax=20
  Type (Fourier_Serie) :: FS1, FS2, FS3
  Complex (kind=DPC) :: Data(Nmax), X
  Integer :: Nt

  Do I = 1, Nmax
     X = Cmplx(TWOPI_DP*I/Nmax,kind=DPC)
     Data(I) = Sin(X) + Cmplx(0.0_DP,I*2.0_DP,kind=DPC)
  End Do

  FS1 = DFT(Data)

  CALL Save_Serie(FS1,'datamodes.dat')
  CALL Read_Serie(FS2,'datamodes.dat')

  Write(*,'(ES33.25)')Sum(Abs(FS1%Coef(:) - FS2%Coef(:)))


  Stop
End Program TestFourier
```

Thirteen

MODULE `Time`

Th `MODULE` Time is a module to provide access to date and time properties.

13.1 Type `tm`

13.1.1 Description

A new data type, called `tm` is defined. It has some properties common with the same derived type defined in the `C` standard library. The components of the type specify a time: Day, year, month, hour, etc...

13.1.2 Components

`hour`: Integer. Hour of the day [0-23].

`min`: Integer, Minutes after the hour [0-59].

`sec`: Integer. Seconds after the minute [0-59].

`msec`: Integer. Miliseconds after the second [0-999].

`year`: Integer. Year.

`month`: Integer. Month of the year [0-11].

`mday`: Integer. Day of the month [1-31].

`wday`: Integer. Day of the week since Sunday [0-6].

13.1.3 Example

A small example defining a `tm` data type.

Listing 13.1: Defining a Time data type.

```
Program Test

  USE NumTypes
  USE Time
```

```
  Type (tm) :: Oneday

  OneDay%hour = 12
  OneDay%min  = 0
  OneDay%sec  = 0
  OneDay%mday = 10
  OneDay%mon  = 0
  OneDay%year = 2007
  OneDay%wday = 3

  Stop
End Program Test
```

13.2 Function gettime()

13.2.1 Description

The function `gettime()` returns the current time and date in a `type tm` data type.

13.2.2 Arguments

This function has no arguments.

13.2.3 Output

`Type tm`, containing all the information about the date and time.

13.2.4 Example

A small program that prints the current year.

Listing 13.2: Obtaining the current date and time.

```
Program Test

  USE NumTypes
  USE Time

  Type (tm) :: Oneday

  Oneday = gettime()

  Write(*,*)'Current year: ', Oneday%year

  Stop
End Program Test
```

13.3 Function `isleap(Nyr)`

13.3.1 Description

The function `isleap(Nyr)` returns `.true.` if `Nyr` is a leap year, and `.false.` otherwise. Note that the leap years are different in the Julian and Gregorian calendars. In this code the Gregorian calendar is supposed valid *after* 1582[1].

13.3.2 Arguments

`Nyr:` Integer. The year.

13.3.3 Output

Logical. `.true.` if `Nyr` is a leap year, and `.false.` otherwise.

13.3.4 Example

A small program that tell us if the current year is leap.

Listing 13.3: Are we in a leap year?.

```
Program Test

  USE NumTypes
  USE Time

  Type (tm) :: Oneday

  Oneday = gettime()

  If (isleap(Oneday%year)) Then
    Write(*,*)'We are in a leap year.'
  Else
    Write(*,*)'We are not in a leap year.'
  End If

  Stop
End Program Test
```

13.4 Function `asctime(t)`

13.4.1 Description

The function `asctime`, returns a 24 length character string from a type `tm` data type, containing the date and time, in a similar way that the function `asctime` of the `C` standard library, for example:

[1] For more details, take a look at

`http://en.wikipedia.org/wiki/Gregorian_calendar`

```
Wed Jan 10 19:15:49 2007
```

13.4.2 Arguments

`t:` Type `tm`. A Type `tm` data type containing the date and time.

13.4.3 Output

Character (len=24). A 24 length character string with the format `Www Mmm dd hh:mm:ss yyyy`, where `Www` is the weekday, `Mmm` the month in letters, `dd` the day of the month, `hh:mm:ss` the time, and `yyyy` the year.

13.4.4 Example

A small program that prints the current time.

Listing 13.4: Printing current date/time.

```
Program Test

  USE NumTypes
  USE Time


  Write(*,'(1A)')asctime(gettime())


  Stop
End Program Test
```

13.5 Function Day_of_Week(Day, Month, Year)

13.5.1 Description

The function `Day_of_Week(Day, Month, Year)`, returns the day of the week since sunday (sunday is 0), of the date that correspond to the input `Day, Month, Year`.

13.5.2 Arguments

`Day:` Integer. The day of the month [1-31].

`Month:` Integer. The month of the year [0-11].

`Year:` Integer. The year.

13.5.3 Output

Integer. The day of the week since sunday, thus a number between 0 and 6, with 0 corresponding to sunday.

13.5.4 Example

A small program that prints the date and time of the first of january of 1900.

Listing 13.5: Day of week of the first of January 1900.

```
Program Test

  USE NumTypes
  USE Time

  Type (tm) :: Oneday

  Oneday%hour = 12
  Oneday%min  = 0
  OneDay%sec  = 0
  OneDay%mday = 1
  OneDay%mon  = 0
  OneDay%year = 1900

  OneDay%wday = Day_of_Week(Oneday%mday, Oneday%mon, Oneday%year)

  Write(*,*)asctime(Oneday)

  Stop
End Program Test
```

GNU Free Documentation License

Version 1.2, November 2002

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliography

[1] G. N. W. E. T. Whittaker, *A course on modern analysis.* Cambridge University Press, 1969.

[2] A. González-Arroyo and A. Ramos, *Expansion for the solutions of the Bogomolny equations on the torus*, *JHEP* **07** (2004) 008, [`hep-th/0404022`].

Index

Subroutine abort([routine], msg), 6
Function asctime(t), 95

Function Basis(X1, X2, n, s, q, itau[, Prec]) , 48
Function Bisec(a, b, Fbis, [Tol]), 78

Function Conjg(FS), 90

Function Day_of_Week(Day, Month, Year), 96
Function Degree(P), 66
Function Deriv(P), 69
Function Det(M), 35
Function DFT(Data, Is), 89

Function Euler(Init, Xo, Xfin, Feuler, [Tol]), 21
Function Eval_Serie(FS, X, [Y], Tx, [Ty]) , 87

Function Factorial(N), 49

Function GammaLn(X), 43
Function gettime(), 94

Function Hermite(n,x[, Dval]), 46
Function HermiteFunc(n, x[, Dval]), 47
Subroutine Histogram(Val, Ndiv, Ntics, Vmin, Vmax, h), 56

Subroutine Init(P, Dgr), 66
Subroutine Init_Serie(FS,Ns), 86
Subroutine Insrt(X[, Ipt]), 38
Function Integra(P, Cte), 70
Function Interpol(X, Y), 72
Function InterpolValue(X, Y, Xo), 71
Function isleap(Nyr), 95

Subroutine Laplace(X, [Rm], [Rsig]), 55

Subroutine LinearReg(X, Y, Yerr, [Func], Coef, Cerr, ChisqrV), 57
Function Locate(X, X_0[, Iin]), 40
Subroutine LU(M, Ipiv, Idet), 32
Subroutine LUsolve(M, b), 34

Function MaxPosition(FVal, IpX, IpY), 29
Function Mean(X), 51
Function Moment(X, k), 53

Function Newton(Xo, Fnew, [Tol]) , 76
Subroutine Normal(X, [Rm], [Rsig]), 54

Subroutine perror([routine], msg), 5
Subroutine Pivoting(M,Ipiv,Idet), 31

Subroutine Qsort(X[, Ipt]), 39

Subroutine Read_Serie(FS, File), 91
Function Rgnkta(Init, Xo, Xfin, Feuler, [Tol]), 24
Subroutine RootPol(a, b, [c, d], z1, z2, [z3, z4]), 75

Subroutine Save_Serie(FS, File), 91
Function Simpson(a, b, Func, [Tol]), 11
Function SimpsonAb(a, b, Func, [Tol]), 14
Function SimpsonInfDw(a, Func, [Tol]), 17
Function SimpsonInfUp(a, Func, [Tol]), 15
Function SimpsonSingDw(a, b, Func, [Tol], gamma), 20
Function SimpsonSingUp(a, b, Func, [Tol], gamma), 18
Subroutine Spline(X, Y, Ypp0, YppN, Pols), 73

Function `Stddev(X)`, 52
Function `Step(X, FStep[, Tol])`, 27
Subroutine `Swap(X,Ind1,Ind2)`, 37

Function `Theta(i, z, tau[, Prec])`, 44
Function `ThetaChar(a, b, z, tau[, Prec])`, 45
Type `tm`, 93
Function `Trapecio(a, b, Func, [Tol])`, 9
Function `TrapecioAb(a, b, Func, [Tol])`, 12

Function `Unit(FS, Ns)`, 88

Function `Value(P, X)`, 68
Function `Var(X)`, 52