A `FORTRAN 90` numerical library

Alberto Ramos. Madrid, November 2006.

Contents

List of Tables

Listings

Generalities

This is the documentation of a total of fourteen `FORTRAN 2003` 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 2003`, this means that it should run on any machine and with any compiler that conforms the standard. In particular the code of all these modules has been compiled using `GNU gfortran`, `INTEL ifort`.

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".

Traditionally the source of the code hass been hosted in `http://www.sourceforge.net`. Due to the access blocking of some countries the code has been moved to `http://github.com`.

To obtain a copy of the source code and documentation, follow one of the following two options

- If you have `git` installed in your system, simply type

        user@pc $ git clone git://github.com/ramos/afnl.git

- In any other case, go to `http://github.com/ramos/afnl/downloads` and get the latest version of the library.

Minuit minimisation routines

The library includes an API to minuit minimisation routines. If you want to use this routines, you should have the fortran version of minuit available in your system. There is the option of building the `afnl` library without minuit support (see next section).

---

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

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 `libafnl.a`, and probably (that depends on the particular compiler) some `.mod` files. Copy the `libafnl.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 case that you do not have a version of the fortran minuit library in your system, use `make lib-no-minuit` to build a version of `afnl` without the minuit API.

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.

```
1    Program Types_of_Data
     USE NumTypes
3

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

```fortran
 7        Complex (kind=SPC) :: Ac
          Complex (kind=DPC) :: Dc
 9
          Write(*,*)Kind(A), Kind(Aa)
11


13        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.

<div align="center">

Three

</div>

<div align="right">

MODULE Error

</div>

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.

```fortran
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.

```
 1  Program Test
      USE Error
 3
      Integer :: N1, N2
 5
      Write(*,*) 'Two integer numbers:'
 7    Read(*,*)N1,N2
 9    If (N2 == 0) Then
        CALL Perror('Test', 'Division by cero. See the product: ')
11      Write(*,*)N1*N2
      Else
13      Write(*,*)N1/N2
      End If
15
      Stop
17  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.

```
 1  Program Test
      USE Error
 3
      Integer :: N1, N2
 5
      Write(*,*) 'Two integer numbers:'
 7    Read(*,*)N1,N2
 9    If (N2 == 0) Then
```

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

15    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.

```fortran
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`.

```
1  Program Test
     USE NumTypes
3    USE Integration

5    Real (kind=DP) :: Tol

7    Interface
        Function Fint(X)
9          USE NumTypes

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

```
      End Interface
15
      Tol = 1.0E-6_DP
17    Write(*,*) 'Integral of x**2 between 0 and 1:'
      Write(*,*) Simpson(0.0_DP, 1.0_DP, Fint, Tol)
19
      Stop
21  End Program Test

23  ! ************************************************
    ! *
25  Function Fint(X)
    ! *
27  ! ************************************************

29    USE NumTypes

31    Real (kind=DP), Intent (in) :: X
      Real (kind=DP) :: Fint
33
      Fint = X**2
35
      Return
37  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.

```fortran
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.

```
1  Program Test
     USE NumTypes
3    USE Integration

5    Real (kind=DP) :: Tol

7    Interface
        Function Fint(X)
9          USE NumTypes

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

```fortran
     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:

```fortran
    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 ∞.

```fortran
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.

```
1  Program Test
     USE NumTypes
3    USE Integration

5    Real (kind=DP) :: Tol

7    Interface
       Function Fint(X)
9        USE NumTypes

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

```fortran
            Real (kind=DP) :: Fint
13      End Function Fint
    End Interface
15
    Tol = 1.0E-6_DP
17  Write(*,*) 'Integral of e**(-x**2) between -infinity and 0:'
    Write(*,*) SimpsonInfDw(0.0_DP, Fint, Tol)
19
    Stop
21 End Program Test

23 ! ************************************************
   ! *
25 Function Fint(X)
   ! *
27 ! ************************************************

29    USE NumTypes

31    Real (kind=DP), Intent (in) :: X
      Real (kind=DP) :: Fint
33
      Fint = exp(-X**2)
35
      Return
37 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:

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

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:

```fortran
    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.

```fortran
1  Program Test
     USE NumTypes
3    USE Integration

5    Real (kind=DP) :: Tol

7    Interface
        Function Fint(X)
9         USE NumTypes

11        Real (kind=DP), Intent (in) :: X
          Real (kind=DP) :: Fint
13      End Function Fint
     End Interface
15
     Tol = 1.0E-6_DP
17   Write(*,*) 'Integral of 1/sqrt(-x) between -1 and 0:'
     Write(*,*) SimpsonSingUp(-1.0_DP, 0.0_DP, Fint, Tol, 0.5_DP)
19
     Stop
21 End Program Test

23 ! ************************************************
   ! *
25 Function Fint(X)
   ! *
27 ! ************************************************

29   USE NumTypes

31   Real (kind=DP), Intent (in) :: X
     Real (kind=DP) :: Fint
33
```

```
       Fint  =  Sqrt(−X)

35
    Return
37 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:

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

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.

```fortran
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:

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

and the initial conditions:

$$y_i(x_0)$$

After the integration we get:

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

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:

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

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.

```fortran
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:

$$\frac{\dot{\text{y}}_i(x)}{\dot{\text{x}}} = f_i(y_j, x)$$

and the initial conditions:

$$y_i(x_0)$$

After the integration we get:

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

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:

$$\frac{\dot{y}_1(x)}{\dot{x}} = y_2(x); \qquad \frac{\dot{y}_2(x)}{\dot{x}} = -y_1(x)$$

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
```

```
32   ! **************************************************
     ! *                                               *
       Function FEuler(X, Y) Result (Func)
34   ! *                                               *
     ! **************************************************
36
         Real (kind=8), Intent (in) :: X, Y(:)
38       Real (kind=8) :: Func(Size(Y))

40       Func(1) = Y(2)
         Func(2) = -Y(1)
42
         Return
44   End Function FEuler
```

4.11   Function `IntQuadrilateral(P1,P2,P3,P4,Fval)`

4.11.1   Description

Given four 2D points (`P1(2)`, `P2(2)`, `P3(2)`, `P4(2)`), and the values of a function in this points, this routine computes the bilineal interpolation polynomial, and returns the value of this polynomial in the quadrilateral.

4.11.2   Arguments

`P1(:)`, `P2(:)`, `P3(:)`, `P4(:)`: Real single or double precision two component vectors. Four points that delimitate a quadrilateral. The corners of the quadrilateral must be given in (counter-)clock wise order.

`Fval:` Real singler or double precision one dimensional array of four elements. The value of a functio in the four corners of the quadrilateral.

4.11.3   Output

Real single or double precision (same as input). An aproximation of the value of the integral of the function over the quadrilateral.

4.11.4   Examples

Listing 4.11: Computing the area of a quadrilateral.

```
Program TQ
2
     USE NumTypes
4    USE Integration
     USE SpaceGeometry
6
     Real (kind=DP) :: P1(3), P2(3), P3(3), P4(3), F(4), Tp(3)
8
     P1(:) = (/-2.0_DP, -1.4.0_DP, 0.0_DP/)
```

```fortran
10    P2(:)  =  (/1.2_DP,  0.0_DP,  0.0_DP/)
      P3(:)  =  (/2.0_DP,  3.0_DP,  0.0_DP/)
12    P4(:)  =  (/0.0_DP,  2.5_DP,  0.0_DP/)

14    F(:)   =  (/1.0_DP,  1.0_DP,  1.0_DP,  1.0_DP/)


16

      Write(*,*) IntQuadrilateral(P1(1:2),P2(1:2),P3(1:2),P4(1:2),F)
18    Tp(:)  =  Cross_Product(P1,P2)
      Tp(:)  =  Tp + Cross_Product(P2,P3)
20    Tp(:)  =  Tp + Cross_Product(P3,P4)
      Tp(:)  =  Tp + Cross_Product(P4,P1)
22    Write(*,*) Dot_Product((/0.0_DP,0.0_DP,0.5_DP/),Tp)


24

      Stop
26  End Program TQ
```

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 (simplex, quasi-Newton methods, etc...).

5.1 Subroutine `Bracket(X1, X2, X3, Func)`

5.1.1 Description

The routine `Bracket(X1, X2, X3, FStep)` "brackets" a minimum of the function `Func`. That is to say, after calling, '`X1, X2` and `X3` obey

$$\texttt{X1} < \texttt{X2} < \texttt{X3} \tag{5.1}$$

and

$$\texttt{Func(X1)} > \texttt{Func(X2)} \quad \text{and} \quad \texttt{Func(X3)} > \texttt{Func(X2)} \tag{5.2}$$

This assures that `Func` has a minimum in the interval $(\texttt{X1}, \texttt{X3})$. The bracketing of the minimum obey the golden rule, that is to say

$$(\texttt{X2} - \texttt{X1}) = \Phi(\texttt{X3} - \texttt{X2}) \tag{5.3}$$

where Φ is the golden number

$$\Phi = \frac{1 + \sqrt{5}}{2} \tag{5.4}$$

5.1.2 Arguments

X1,X2,X3: Real single or double precision. At output, they bracket a minimum of `Func`. If you have a guess of the minimum of `Func`, introduce it in `X2`. of the minimum.

Func: The function whose minimum we want to bracket. An interface like the following should be declared

```
Interface
    Function Func(Xo)
```

```
      USE NumTypes

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

5.1.3 Example

Listing 5.1: Bracketing a minimum.

```fortran
Program TestMin

  USE NumTypes
  USE Optimization

  Real (kind=DP) :: X1, X2, X3

  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
  X2 = 2.0_DP

  CALL Bracket(X1,X2,X3, FstepM)
  Write(*,*)
  Write(*,*) 'Minimum bracketed: '

  Write(*, '(1A,1ES33.25)') 'X1: ', X1
  Write(*, '(1A,1ES33.25)') 'X2: ', X2
  Write(*, '(1A,1ES33.25)') 'X3: ', X3

  Stop
End Program TestMin

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

  USE NumTypes

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

```
42

44    FstepM = Sin (Xo)

46

      Return
48 End Function  FstepM
```

5.2 Subroutine `LineSrch(X, Func[, Tol])`

5.2.1 Description

The function `LineSrch(X, FStep, Tol)` returns the position of the minimum of the Function `Fstep` with an optional precision `Tol`. This routine does not need the values of the derivative(s) of the function to work.

5.2.2 Arguments

`X[(:)]:` Real single or double precision. An initial guess of the position of the minimum when input. At output, the position of the minimum.

`Func:` 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 Func(Xo)
     USE NumTypes

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

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

```
Interface
   Function Func(Xo)
     USE NumTypes

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

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

5.2.3 Example

Listing 5.2: Minimising a function.

```fortran
Program TestMin

   USE NumTypes
   USE Optimization

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

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

   Stop
End Program TestMin

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

   USE NumTypes

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

```fortran
50    Real (kind=DP) :: FstepM

52
      FstepM = (Xo(1)-1.0_DP)**2 + &
54         & (Xo(2)-2.0_DP)**2 + &
           & (Xo(3)+3.0_DP)**4 + &
56         & (Xo(4)-4.0_DP)**8

58
      Return
60 End Function FstepM
```

5.3 Subroutine `ConjGrad(X, F, Fd[, Tol])`

5.3.1 Description

The function `ConjGrad(X, F, Fd[, Tol])` returns the position of the minimum of the Function `F` with an optional precision `Tol`. This routine uses the conjugate gradient method, and should be much faster that `LineSrch`, but you must be able to compute the derivatives of `F`

5.3.2 Arguments

`X(:):` Real single or double precision. An initial guess of the position of the minimum when input. At output, the position of the minimum.

`F:` The function that we want to minimise. An interface block like the following should be defined.

```fortran
Interface
   Function F(Xo)
     USE NumTypes

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

`Fd:` The gradient of the function that we want to minimise. An interface block like the following should be defined.

```fortran
Interface
   Subroutine Fd(X, D)
     USE NumTypes
     Real (kind=DP), Intent (in) :: X(:)
     Real (kind=DP), Intent (out) :: D(Size(X))
   End Subroutine Fd
End Interface
```

where the vector D returns the value of the derivatives.

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

5.3.3   Example

Listing 5.3: Minimising a function.

```
Program TestMin

Program OOOO

  USE NumTypes
  USE Optimization

  Real (kind=DP) Xx(2)

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

  Interface
     Subroutine Fd(X, D)
       USE NumTypes
       Real (kind=DP), Intent (in) :: X(:)
       Real (kind=DP), Intent (out) :: D(Size(X))
     End Subroutine Fd
  End Interface


  Xx = 112.0_DP

  CALL ConjGrad(Xx, F, Fd, 1.E-10_DP)
  Write(*,*) 'Minimo: ', Xx

  Stop
End Program Test


Function F(X)

  USE NumTypes

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


  F = (X(1)-2.23_DP)**2 + (X(2)+1.23_DP)**2 + X(1)*X(2) + Sin(X(1)*X(2))
```

```
46
   Return
48 End Function F


50

Subroutine Fd(X, D)
52   USE NumTypes

54   Real (kind=DP), Intent (in) :: X(:)
     Real (kind=DP), Intent (out) :: D(Size(X))
56
     D(1) = (X(1)-2.23_DP)*2 + X(2) + Cos(X(1)*X(2))*X(2)
58   D(2) = (X(2)+1.23_DP)*2 + X(1) + Cos(X(1)*X(2))*X(1)

60   Return
   End Subroutine Fd
```

5.4   **Function** `Step(X, FStep[, Tol])`

5.4.1   Description

The function `Step(X, FStep, Tol)` returns the position of the minimum of the Function `Fstep` with an optional precision `Tol`. Unless you know very well what you are doing, you should use `LineSrch` or `ConjGrad` to minimize functions.

5.4.2   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}.

5.4.3   Output

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

5.4.4   Example

Listing 5.4: Minimising a function.

```fortran
1  Program TestMin

3    USE NumTypes
     USE Optimization
5
     Integer, Parameter :: Ndim = 4
7    Real (kind=DP) :: XoM(Ndim), Xmin(Ndim)

9    Interface
        Function FstepM(Xo)
11         USE NumTypes

13         Real (kind=DP), Intent (in) :: Xo(:)
           Real (kind=DP) :: FstepM
15      End Function FstepM
     End Interface
17

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

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

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

```
37
      Stop
39  End Program  TestMin

41  !  *******************************
    !  *
43  Function  FstepM(Xo)
    !  *
45  !  *******************************

47    USE  NumTypes

49    Real (kind=DP), Intent (in) :: Xo(:)
      Real (kind=DP) :: FstepM
51

53    FstepM = (Xo(1)−1.0_DP)**2 + &
            & (Xo(2)−2.0_DP)**2 + &
55          & (Xo(3)+3.0_DP)**4 + &
            & (Xo(4)−4.0_DP)**8
57

59    Return
    End Function  FstepM
```

5.5 Function MaxPosition(FVal, IpX, IpY)

5.5.1 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(:).

5.5.2 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.

5.5.3 Output

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

5.5.4 Example

Listing 5.5: Example of the usage of the routine MaxPosition.

```fortran
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 `MinuitAPI`

This is the documentation of the MODULE `MinuitAPI`, a set of routines to Optimise (maximise or minimise) functions of one or several variables. This module is a simple API to the CERN minuit library[1].

6.1 Subroutine `Minimize(Func, X, Fval, [Bound], [Release], [logfile])`

6.1.1 Description

The routine `Minimize`, minimises the function of several variables Func. As an output you get the position of the minima in the vector `X(:)`, and the value of the function in the minima in the variable `Fval`.

Several optional parameters can be used to put boundaries in the values of the parameters, or to specify a release order for the parameters.

6.1.2 Arguments

`Func:` The function we want to minimise. An interface like the following should be declared

```
Interface
   Function Func(X)
     USE NumTypes

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

`X(:):`   Real double precision one dimensional array. As input an estimate of the position of the minima. As output the position of the minima.

`Fval:` Real double precision. Output. The value of the function at the minima.

---

[1]http://lcgapp.cern.ch/project/cls/work-packages/mathlibs/minuit/home.html

`Bound(:):`   Real double precision one dimensional array. Optional. Parameter limits. The minimum value for parameter n is given by `Bound(2n-1)`, and the maximum value is given by `Bound(2n)`. If both limits are `0.0D0` the parameter has no limits.

`Release(0:,:):` Integer two dimensional array. Optional. The integer two dimensional array can be used to specify a release order of parameters. The first dimension of `Release(0:,:)` contains the steps in which you want to release the parameters. The vector `Release(0,:)` contains the number of parameters released in each step. `Release(1,:)` contains the parameters released in the first step. `Release(2,:)` contains the parameters released in the second step, etc. . .

For example, the array (an $*$ means that the value of this element of the array is irrelevant).

$$R_{ij} = \begin{pmatrix} 2 & 2 & 3 & * \\ 1 & 2 & * & * \\ 3 & 4 & * & * \\ 5 & 6 & 7 & * \end{pmatrix} \tag{6.1}$$

Means that parameters $1, 2$ are free in the first step of the fit. Parameters $(1, 2, 3, 4)$ are free in the second step, and finally parameters $(1, 2, 3, 4, 5, 6, 7)$ are released in the last step of the fit.

`logfile:`   Character (len=*). Optional. A file name where MINUIT will write some information about the minimisation process.

6.1.3   Example

Listing 6.1: Using minuit library to minimize a function.

```fortran
Program TestAPI

  USE MinuitAPI
  USE Statistics
  USE Constants

  Integer, Parameter :: N = 2
  Real (kind=8) :: X(N), Y(N), Ye(N), C(2), Ch

  Interface
     Function Func(X)
        Real (kind=8), Intent (in) :: X(:)
        Real (kind=8) :: Func
     End Function Func
  End Interface


  X(:) = -23.0D0
  CALL Minimize(Func, X, Ch)
  Write(*,*)Ch
  Write(*,*)X
  Write(*,*)Tan(X(1)), Cos(X(2))

```

```
      Stop
25  End Program  TestAPI

27  Function  Func(X)

29    Real (kind=8), Intent (in) :: X(:)
      Real (kind=8) :: Func
31
      Func = (X(1)−tan(X(1)))**2 + (X(2) − Cos(X(2)))**2
33
      Return
35  End Function  Func
```

Seven

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.

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

7.1.1 Description

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

7.1.2 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` $= -1$

7.1.3 Examples

Listing 7.1: Pivoting data of a matrix label

```
1   Program TestLinear

3     USE NumTypes
      USE Linear
5
      Integer, Parameter :: Nord = 4
7
      Real (kind=DP) :: M(Nord,Nord), L(Nord,Nord), U(Nord,Nord), &
9         & Mcp(Nord,Nord)
      Integer :: Ipiv(Nord), Iperm
11
```

```fortran
13    ! Fill M of random numbers
      CALL Random_Number(M)
15
      Write(*,*) 'Original M: '
17    Do I = 1, Nord
          Write(*,'(100ES10.3)')(M(I,J), J = 1, Nord)
19    End Do

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

27    Stop
    End Program TestLinear
```

7.2 Subroutine LU(M, Ipiv, Idet)

7.2.1 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

$$M = LU \tag{7.1}$$

where L is lower triangular, and U upper triangular.

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots \\ L_{21} & 1 & 0 & \dots \\ L_{31} & L_{32} & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}; \quad U = \begin{pmatrix} U_{11} & U_{12} & U_{13} & \dots \\ 0 & U_{22} & U_{23} & \dots \\ 0 & 0 & U_{33} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{7.2}$$

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.

7.2.2 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 $= -1$

7.2.3 Examples

Listing 7.2: Making the LU decomposition.

```fortran
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
```

```
50        Write(*,'(100ES10.3)')(M(I,J), J = 1, Nord)
     End Do
52


54     Stop
   End Program TestLinear
```

7.3 Subroutine LUsolve(M, b)

7.3.1 Description

Solves the linear system of equations

$$M_{11}x_1 + M_{12}x_2 + M_{13}x_3 + M_{14}x_4 + \ldots = b_1 \qquad (7.3)$$
$$M_{21}x_1 + M_{22}x_2 + M_{23}x_3 + M_{24}x_4 + \ldots = b_2$$
$$\vdots$$

7.3.2 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.*

7.3.3 Examples

Listing 7.3: Solving systems of linear equations.

```
1  Program TestLinear

3    USE NumTypes
     USE Linear
5
     Integer, Parameter :: Nord = 10
7
     Real (kind=DP) :: M(Nord,Nord), L(Nord,Nord), U(Nord,Nord), &
9          & Mcp(Nord,Nord), b(Nord), bcp(Nord)
     Integer :: Ipiv(Nord), Iperm
11
     ! Fill M and b of random numbers, and make a copy of both
13   CALL Random_Number(M)
     CALL Random_Number(b)
15   Mcp = M
     bcp = b
17
     ! Solve the linear system
19   CALL LUsolve(M,b)
```

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

25


27    Stop
   End Program TestLinear
```

7.4 Function `Det(M)`

7.4.1 Description

Computes the determinant of the matrix M.

7.4.2 Arguments

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

7.4.3 Output

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

7.4.4 Examples

Listing 7.4: Computing the determinant of a matrix.

```
   Program TestLinear
2
      USE NumTypes
4     USE Linear

6     Integer, Parameter :: Nord = 10

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


12

      ! Fill M of randoms numbers
14    CALL Random_Number(M)

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

18


20    Stop
   End Program TestLinear
```

Eight

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.

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

8.1.1 Description

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

8.1.2 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.

8.1.3 Examples

Listing 8.1: Sorting data.

```fortran
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
```

```fortran
15        Write(*,'(1000I10)')Ima(I)
       End Do
17
       ! Swap first and last elemetns of Ima(:) and plot them.
19       CALL Swap(Ima, 1, Nmax)
       Write(*,*)'# With first and last elements permuted: '
21       Do I = 1, Nmax
          Write(*,'(1000I10)')Ima(I)
23       End Do


25
       Stop
27  End Program TestNN
```

8.2 Subroutine Insrt(X[, Ipt])

8.2.1 Description

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

8.2.2 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.

8.2.3 Examples

Listing 8.2: Sorting data.

```fortran
1  Program TestNN

3    USE NumTypes
     USE NonNumeric
5
     Integer, Parameter :: Nmax = 10
7    Integer :: Ima(Nmax)
     Real (kind=DP) :: X(Nmax), Y(Nmax)
9

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

15    ! Plot an unsorted data table
     Do I = 1, Nmax
```

---

[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.

```
17          Write(*,'(1000ES13.5)')X(I), Y(I)
         End Do
19
         ! Sort them, and plot the table again. Same points, but this time
21       ! sorted
         CALL Insrt(X, Ima)
23       Write(*,*)'# Again, this time sorted: '
         Do I = 1, Nmax
25          Write(*,'(1000ES13.5)')X(I), Y(Ima(I))
         End Do
27

29       Stop
      End Program TestNN
```

8.3 Subroutine `Qsort(X[, Ipt])`

8.3.1 Description

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

8.3.2 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.

8.3.3 Examples

Listing 8.3: Sorting data.

```
Program TestNN
2
   USE NumTypes
4  USE NonNumeric

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

10
   ! Fill X(:) with random data, and define Y(:)
12 CALL Random_Number(X)
   Y = Sin(12.34_DP*(X-0.5_DP))
14
   ! Plot an unsorted data table
16 Do I = 1, Nmax
      Write(*,'(1000ES13.5)')X(I), Y(I)
18 End Do
```

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

28

      Stop
30 End Program TestNN
```

8.4   Function `Locate(X, X`$_0$`[, Iin])`

8.4.1   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(:)`

8.4.2   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.

8.4.3   Output

Integer. The position n such that

$$\texttt{X(n)} < \texttt{X}_0 < \texttt{X(n + 1)}$$

8.4.4   Examples

Listing 8.4: Searching data position in an ordered list.

```fortran
Program TestNN
2
   USE NumTypes
4  USE NonNumeric

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

10
   ! Fill X(:) with random data, and set X0 to some arbitrary value.
```

```fortran
12      CALL  Random_Number(X)
        X0 = 0.276546754_DP
14
        ! Sort X(:), find the position of X0, and plot the neightborr
16      ! elements.
        CALL  Qsort(X)
18      Idx = Locate(X, X0)
        Write(*, '(1A,1ES33.25)') 'Searched element: ', X0
20      Write(*, '(1A,1ES33.25)') 'Previous element in the list: ', X(Idx)
        Write(*, '(1A,1ES33.25)') 'Next element in the list:     ', X(Idx+1)
22
        Stop
24  End Program TestNN
```

Nine

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.

9.1 Function `GammaLn(X)`

9.1.1 Description

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

9.1.2 Arguments

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

9.1.3 Output

A real Double precision (DP).

9.1.4 Examples

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

Listing 9.1: Computing the Gamma Function.

```fortran
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))))
```

```
12      End Do

14

        Stop
16   End Program  TestSpecialFunc
```

9.2   Function `erf(X)`

9.2.1   Description

Compute values of the error function erf(X). A high precision algorithm by W. J. Candy [2] is used.

9.2.2   Arguments

X: Single (SP) or double (DP) precision. The point in which we want to know the value of erf(X).

9.2.3   Output

Same type as input.

9.2.4   Examples

This program should compute the erf(X) and its complementary in an given point.

Listing 9.2: Computing the Error Function.

```
    Program  TestSpecialFunc
2
       USE  NumTypes
4      USE  SpecialFunc

6      Real (kind=DP)  :: X

8      Write(*,*) 'Enter  a  number:  '
       Read(*,*)X
10     Write(*, '(ES33.25)') erf(X)
       Write(*, '(ES33.25)') erfc(X)
12
       Stop
14   End Program  TestSpecialFunc
```

9.3   Function `erfc(X)`

9.3.1   Description

Compute values of the complementary error function erf(X). A high precision algorithm by W. J. Candy [2] is used.

9.3.2 Arguments

X: Single (SP) or double (DP) precision. The point in which we want to know the value of $\mathrm{erfc}(X)$.

9.3.3 Output

Same type as input.

9.3.4 Examples

This program should compute the $\mathrm{erf}(X)$ and its complementary in an given point.

Listing 9.3: Computing the complementary Error Function.

```
Program  TestSpecialFunc

   USE  NumTypes
   USE  SpecialFunc

   Real (kind=DP) :: X

   Write(*,*) 'Enter a number: '
   Read(*,*)X
   Write(*, '(ES33.25)')erf(X)
   Write(*, '(ES33.25)')erfc(X)

   Stop
End Program  TestSpecialFunc
```

9.4 Function `Theta(i, z, tau[, Prec])`

9.4.1 Description

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

$$\vartheta_i(z|\tau) \tag{9.1}$$

For a definition and properties of these functions take a look [3], here we will only say that following the conventions of the cited reference, our Theta functions have quasi-periods π and $\tau\pi$.

9.4.2 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 ($\mathrm{Im}(\tau) > 0$).

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}

9.4.3   Output

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

9.4.4   Examples

Listing 9.4: Computing the Jacobi Theta functions.

```fortran
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
```

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

9.5.1   Description

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

$$\vartheta \left[\begin{array}{c} a \\ b \end{array} \right] (z|\tau) \tag{9.2}$$

9.5.2   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 $(\text{Im}(\tau) > 0)$.

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

9.5.3 Output

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

9.5.4 Examples

Listing 9.5: Computing the Jacobi Theta functions with characteristics.

```fortran
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
```

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

9.6.1 Description

Returns the value of the nth Hermite polynomial in the point X. If `Dval` is specified, the value of the Derivative of the nth Hermite polynomial in the point X is also returned.

9.6.2 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.

9.6.3 Output

Real single or double precision (same as input). The value of the nth Hermite Polynomial in the point X.

9.6.4 Examples

Listing 9.6: 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
```

9.7 Function `HermiteFunc(n, x[, Dval])`

9.7.1 Description

Returns the value of the nth Hermite function

$$\frac{1}{\sqrt{n!2^n\sqrt{\pi}}}e^{-x^2/2}H_n(x) \tag{9.3}$$

in the point X. If `Dval` is specified, the value of the Derivative of the nth Hermite function in the point X is also returned.

9.7.2 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.

9.7.3   Output

Real single or double precision (same as input). The value of the nth Hermite function in the point X.

9.7.4   Examples

Listing 9.7: Compute the Hermite functions.

```fortran
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
```

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

9.8.1   Description

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

$$\mid n,s\rangle = e^{i\frac{f}{2}x_1 x_2}\sum_{k\in s+q\mathbb{Z}}e^{-u^2/2}H_n(u)e^{2\pi i k\frac{x_1}{l_1}}\qquad n=0,\dots\infty; s=1,\dots,q \qquad (9.4)$$

defined in the appendix of [1] (look there for more details and properties).

9.8.2   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: $\texttt{itau} = l_2/l_1$.

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

9.8.3   Output

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

9.8.4   Examples

Listing 9.8: Computing the bassi of a special Hilbert space (details in [1]).

```fortran
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
```

9.9 Function `Factorial(N)`

9.9.1 Description

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

9.9.2 Arguments

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

9.9.3 Output

A real Double precision (DP).

9.9.4 Examples

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

Listing 9.9: Computing the factorial.

```fortran
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
```

9.10 Function `Legendre(l, m, X)`

9.10.1 Description

Computes the value at X of the (l, m) Legendre polynomial $P_l^m(X)$.

9.10.2 Arguments

`l,m:` Integer. Specifies which Legendre polynomial we want to compute.

`X:` Real sinple or double precision. The point at which we want to compute the polinomial.

9.10.3   Output

A real simple or double precision (same as `X`).

9.10.4   Examples

This program writes the values of the Legendre polynomials $P_l^m(0.5)$ for $0 \leq m \leq l$.

Listing 9.10: Computing some legendre polynomials.

```fortran
Program TestLeg

  USE NumTypes
  USE SpecialFunc

  Real (kind=DP) :: Y

  Read(*,*)L

  Do I = 0, L
     Y = Legendre(L, I, 0.50_DP)
     Write(*,*)I, Y, Legendre(L, I, 0.5_SP)
  End Do

  Stop
End Program TestLeg
```

9.11   Function `SphericalHarmonics(l, m, th,ph)`

9.11.1   Description

Computes the value at (θ, ϕ) of the (l, m) spherical harmonic $Y_l^m(\theta, \phi)$.

9.11.2   Arguments

`l,m:` Integer. Specifies which spherical harmonic we want to compute.

`th,ph:` Real single or double precision. (θ, ϕ) coordinates of the point at which we we want to compute the vlue of the spherical harmonic.

9.11.3   Output

A complex simple or double precision (same as `th`).

9.11.4   Examples

This program checks the relation $Y_l^m(\theta, phi) = (-)^m Y_l^{*-m}(\theta, \phi)$ for $\theta = 0.5, \phi = 0.35$.

Listing 9.11: Computing some spherical harmonics.

```fortran
Program TestSph
```

```fortran
    USE  NumTypes
    USE  SpecialFunc

    Complex (kind=DPC)  :: Y

    Read(*,*) L

    Do I = 0, L
        Y = SphericalHarmonic(L, I, 0.50_DP, 0.35_DP)
        Write(*,'(1A,1I4,1A,1I4)') 'Spherical Harmonic l=', L, ' m=+-', I
        Write(*,*) '  +m ', Y
        Write(*,*) '  -m ', SphericalHarmonic(L, -I, 0.5_DP, 0.35_DP)
    End Do

    Stop
End Program TestSph
```

Ten

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.

10.1   Function `Mean(X)`

10.1.1   Description

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

10.1.2   Arguments

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

10.1.3   Output

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

10.1.4   Examples

Listing 10.1: Computing the Mean of a vector of numbers.

```
1   Program Tests

3      USE NumTypes
       USE Error
5      USE Statistics

7      Integer, Parameter :: Nmax = 100
       Real (kind=DP) :: X(Nmax)
9
       CALL Random_Number(X)
11     Write(*,'(ES33.25)')Mean(X)
```

```fortran
13       Stop
     End Program Tests
```

10.2   Function `Median(X)`

10.2.1   Description

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

10.2.2   Arguments

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

10.2.3   Output

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

10.2.4   Examples

Listing 10.2: Computing the Median of a vector of numbers.

```fortran
   Program Tests
2
     USE NumTypes
4    USE Error
     USE Statistics
6
     Integer, Parameter :: Nmax = 5
8    Real (kind=SP) :: X(Nmax) = (/1.0, 1.0, 2.0, 4.0, 1.5/)

10   Write(*,'(ES33.25)')Median(X)

12
     Stop
14 End Program Tests
```

10.3   Function `WMedian(X, w)`

10.3.1   Description

Compute the weighted median of the numbers stored in `X(:)` with weights `w(:)`.

10.3.2   Arguments

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

`X(:):` Double (DP) or simple (SP) precision one dimensional array. The weights.

10.3.3 Output

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

10.3.4 Examples

Listing 10.3: Computing the Weighted Median of a vector of numbers.

```fortran
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 5
  Real (kind=SP) :: X(Nmax) = (/1.0, 1.0, 2.0, 4.0, 1.5/)
  Real (kind=SP) :: w(Nmax) = (/10.0, 2.0, 3.0, 4.0, 1.5/)

  Write(*,'(ES33.25)')WMedian(X, w)


  Stop
End Program Tests
```

10.4 Function `WPercentile(X, w, p)`

10.4.1 Description

Compute the weighted percentile `p` of the numbers stored in `X(:)` with weights `w(:)`.

10.4.2 Arguments

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

`X(:):` Double (DP) or simple (SP) precision one dimensional array. The weights.

`p:` Double (DP) or simple (SP) precision number. The percentile (should be between 0 and 100).

10.4.3 Output

A real double or simple precision (same type as the input). The weighted percentile `p` of the values.

10.4.4 Examples

Listing 10.4: Computing the Weighted Median in two ways.

```fortran
Program Tests
```

```fortran
 3      USE NumTypes
        USE Error
 5      USE Statistics

 7      Integer, Parameter :: Nmax = 5
        Real (kind=SP) :: X(Nmax) = (/1.0, 1.0, 2.0, 4.0, 1.5/)
 9      Real (kind=SP) :: w(Nmax) = (/10.0, 2.0, 3.0, 4.0, 1.5/)

11      Write(*, '(ES33.25)')WMedian(X, w)
        Write(*, '(ES33.25)')WPercentile(X, w, 50.0_DP)

13


15      Stop
     End Program Tests
```

10.5 Subroutine WConfInt(X, w, Xmin, Xmax)

10.5.1 Description

Compute the weighted 16th and 84th percentiles f the numbers stored in `X(:)` with weights `w(:)`, and return the values in `Xmin` and `Xmax`.

10.5.2 Arguments

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

`X(:):` Double (DP) or simple (SP) precision one dimensional array. The weights.

`Xmin:` Double (DP) or simple (SP) precision number. Output. The 16th percentile.

`Xmax:` Double (DP) or simple (SP) precision number. Output. The 84th percentile.

10.5.3 Examples

Listing 10.5: Computing the 1 sigma confidence interval in two ways.

```fortran
     Program Tests
 2
        USE NumTypes
 4      USE Error
        USE Statistics
 6
        Integer, Parameter :: Nmax = 500
 8      Real (kind=SP) :: X(Nmax), w(Nmax), X1, X2

10      CALL Normal(X)
        CALL Random_Number(w)
12
        Write(*, '(ES33.25)')WPercentile(X, w, 16.0_DP)
14      Write(*, '(ES33.25)')WPercentile(X, w, 84.0_DP)
```

```
      CALL WConfInt(X, W, X1, X2)
16    Write(*,'(ES33.25)')X1, X2

18    Stop
   End Program Tests
```

10.6   Function `Var(X)`

10.6.1   Description

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

10.6.2   Arguments

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

10.6.3   Output

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

10.6.4   Examples

Listing 10.6: Computing the Variance of a set of numbers.

```
1  Program Tests

3     USE NumTypes
      USE Error
5     USE Statistics

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


11
      CALL Random_Number(X)
13    Write(*,'(ES33.25)')Var(X)


15
      Stop
17 End Program Tests
```

10.7   Function `Stddev(X)`

10.7.1   Description

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

10.7.2 Arguments

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

10.7.3 Output

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

10.7.4 Examples

Listing 10.7: Compputing the standard deviation.

```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)')Stddev(X)


   Stop
End Program  Tests
```

10.8 Function `Moment(X, k)`

10.8.1 Description

Returns the kth moment of the values stored in the vector `X(:)`.

10.8.2 Arguments

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

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

10.8.3 Output

Real single or double precision. The kth moment of the numbers.

10.8.4 Examples

Listing 10.8: Computing the kth 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
```

10.9 Subroutine `Histogram(Val, Ndiv, Ntics, Vmin, Vmax, h)`

10.9.1 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.

10.9.2 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 $\mathtt{Vmin} + (\mathtt{I} - 1)\mathtt{h}$ and $\mathtt{Vmin} + \mathtt{Ih}$.

`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.

10.9.3 Examples

Listing 10.9: Making Histograms.

```
Program Tests

   USE NumTypes
   USE Error
```

```
5    USE  Statistics

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

13   CALL Normal(X, 1.23_DP, 0.345_DP)
     CALL Histogram(X, Ndiv, Ntics, Xmin, Xmax, h)
15
     Do I = 1, Ndiv
17      Xac = Xmin + (I−1)*h
        Write(*,'(1ES33.25,1I)')Xac, Ntics(I)
19   End Do

21   Stop
   End Program Tests
```

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

10.10.1   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.

10.10.2   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 or two dimensional array. If a one dimensional array is inserted, they are the errors of the points (if you don't have them, you should put all of them to some non-zero value). If a two dimensional array is given, it is treated as the correlation matrix.

`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.

10.10.3   Examples

Listing 10.10: Doing linear regressions.

```
Program Tests
2
   USE NumTypes
4  USE Error
   USE Statistics
```

```fortran
      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)

      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(:)
```

```fortran
       Integer, Intent (in) :: i
60     Real (kind=DP) :: Fd

62     If (I==1) Then
           Fd = 1.0_DP
64     Else If (I==2) Then
           Fd = X(1)*sin(X(2))
66     Else If (I==3) Then
           Fd = X(1)**2*X(2)
68     Else If (I==4) Then
           Fd = X(1)
70     End If

72     Return
     End Function FD
```

10.11 Subroutine `NonLinearReg(X, Y, Yerr, Func, Coef, Cerr, ChisqrV)`

10.11.1 Description

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

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.

This routine uses the Levenberg-Marquardt algorithm to perform the optimisation[1].

10.11.2 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 or two dimensional array. If a one dimensional array is inserted, they are the errors of the points (if you don't have them, you should put all of them to some non-zero value). If a two dimensional array is given, it is treated as the correlation matrix.

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

```fortran
    Interface
        Subroutine Func(X, Cf, Valf, ValD)

            USE NumTypes
```

---

[1] http://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

```
      Real (kind=SP), Intent (in) :: X, Cf(:)
      Real (kind=SP), Intent (out) :: Valf, ValD(Size(Cf))

   End Subroutine Func
End Interface
```

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

```
Interface
   Subroutine Func(X, Cf, Valf, ValD)

      USE NumTypes

      Real (kind=SP), Intent (in) :: X(:), Cf(:)
      Real (kind=SP), Intent (out) :: Valf, ValD(Size(Cf))

   End Subroutine Func
End Interface
```

if it is a multidimensional fitting.

This routine returns the values of the function at X for some values of the parameters given in `Cf` in the variable `Valf`, and a vector with the derivatives (respect with the parameters) in `ValD(:)`.

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

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

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

10.11.3   Examples

In this example we will fit some generated data, with a Normal noise to the function

$$f(x_1, x_2; a, b) = \sin(ax_1) + bx_1x_2$$

We will generate the data with the values $a = 2.0$ and $b = 0.2$, so our fitting routine *should* return this values within errors.

The derivatives of the function (repect the parameters a and b), as well as the value of function are given by the user routine `Func`. The derivatives are:

$$\begin{aligned}
\frac{\partial f(x_1, x_2; a, b)}{\partial a} &= x_1 \cos ax_1 \\
\frac{\partial f(x_1, x_2; a, b)}{\partial b} &= x_1 x_2
\end{aligned}$$

Listing 10.11: Doing non linear regressions.

```fortran
Program NLFit

   USE NumTypes
   USE Statistics

   Interface
      Subroutine Func(X, Cf, Valf, ValD)

         USE NumTypes

         Real (kind=DP), Intent (in) :: X(:), Cf(:)
         Real (kind=DP), Intent (out) :: Valf, ValD(Size(Cf))

      End Subroutine Func
   End Interface

   Integer, Parameter :: Np = 20, Ndim = 2
   Real (kind=DP) :: X(Np, Ndim), Y(Np), Ye(Np), Co(2), Vd(2), Ce(2), Ch

   ! First Fill the data
   CALL Random_Number(X)
   X = 2.0_DP*(X - 0.5_DP)
   Co(1) = 2.0_DP
   Co(2) = 0.2_DP
   CALL Normal(Ye, 0.0_DP, 0.5_DP)
   Do I = 1, Np
      CALL Func(X(I,:), Co, Y(I), Vd)
      Y(I) = Y(I) + Ye(I)
   End Do


   ! Now Perform the non linear fit
   Co = 1.0_DP
   CALL NonLinearReg(X, Y, Abs(Ye), Func, Co, Ce, Ch)
   Do I = 1, Npar
     Write(*,'(1A,100ES33.25)') 'Parameter and error: ', Co(I), Ce(I)
   End Do
   Write(*,'(1A,100ES33.25)')&
   & 'Chi Square per degree of freedom of the Fit: ', Ch


   Stop
End Program NLFit

Subroutine Func(X, Cf, Valf, ValD)

   USE NumTypes

   Real (kind=DP), Intent (in) :: X(:), Cf(:)
   Real (kind=DP), Intent (out) :: Valf, ValD(Size(Cf))
```

```
      Valf = Sin(Cf(1)*X(1)) + Cf(2)*X(1)*X(2)
53    ValD(1) = X(1)*Cos(Cf(1)*X(1))
      ValD(2) = X(1)*X(2)
55


57  End Subroutine Func
```

10.12 Subroutine SetLuxLevel(Ilevel)

10.12.1 Description

This function changes the behaviour of the "intrinsic" Random_Number subroutine. To call the intrinsic FORTRAN 90 routine, you should set the Luxury level to zero. Other values makes the Random_Number subroutine use the Marsaglia and Zaman algorithm modified by M. Lüscher [4]. This part of the code has some inspiration in the FORTRAN 90 implementation of Allan Miller. **Note:** This routine affects the way all the other routines that use pseudo random number generators works. That is to say, all the routines of this module that follow this one.

10.12.2 Arguments

Ilevel: Integer. Set the way the random number generator works. We have the following options:

- Ilevel $= 0$. The default intrinsic FROTRAN 90 routine is used. This is usually considered a *bad*, non portable pseudo-random number generator, that you do not want to use when correlations are important.

- Ilevel $= 1$. This is the original Marsaglia and Zaman algorithm. Probably better than the intrinsic procedure, but still with large correlations.

- Ilevel $= 2$. The M. Lüscher modified version of the algorithm with $p = 48$. Still fail many tests, but a very good pseudo-number generator for many things.

- Ilevel $= 3$. The M. Lüscher modified version of the algorithm with $p = 97$. Theoretically still defective (fail the serial correlation test).

- Ilevel $= 4$. The M. Lüscher modified version of the algorithm with $p = 218$. Any theoretically possible correlation has a very small probability of being observed. A value very similar to this has been used for large scale high precision lattice gauge theory computations. This corresponds with the Lüscher recommended level 0.

- Ilevel $= 5$. The M. Lüscher modified version of the algorithm with $p = 404$. Good for anything you need the pseudo-random numbers (but cryptography, of course). It seems completely pointless to use values of p higher than this. This corresponds with the Lüscher recommended level 1. This is the **default value**.

- Ilevel $= 6$. The M. Lüscher modified version of the algorithm with $p = 794$. Good for anything you need the pseudo-random numbers (but cryptography, of course). Probably waste of computer resources. This corresponds with the Lüscher recommended level 2.

- Other values of Ilevel (> 24). Sets the value of p in the M. Lüscher modified version of the algorithm. Don't use it, unless you know what you are doing.

10.12.3   Examples

Listing 10.12: Setting the Luxury level of the pseudo random number generator.

```fortran
1  Program Tests

3    USE NumTypes
     USE Statistics
5
     Integer, Parameter :: NN = 70
7    Real (kind=SP) :: rr(NN)

9    CALL SetLuxLevel(5)
     CALL Random_Number(rr)
11   Write(*,*)rr


13
     Stop
15 End Program Tests
```

10.13   Subroutine `PutLuxSeed([ISeed(25)])`

10.13.1   Description

Restarts the position in the Luxury pseudo-random number generator.

10.13.2   Arguments

`ISeed(25):` Integer one dimensional array of 25 elements. Optional. If present, restarts the
    generator from an output of `GetLuxSeed`. If not present, initialises the random number
    generator.

10.13.3   Examples

Listing 10.13: Using a previously saved point in the generating process.

```fortran
1  Program Tests

3    USE NumTypes
     USE Statistics
5
     Integer, Parameter :: NN = 70
7    Real (kind=SP) :: rr(NN)
     Integer :: Sd(25)
9

11   Open(Unit=69, File="seed.dat")
     Read(69,*)Sd
13   CALL PutLuxSeed(Sd)
     CALL Random_Number(rr)
15   Write(*,*)rr
```

```
17      Stop
19   End Program  Tests
```

10.14 Subroutine `GetLuxSeed(ISeed(25))`

10.14.1 Description

Saves the position in the Luxury pseudo-random number generator.

10.14.2 Arguments

`ISeed(25):` Integer one dimensional array of 25 elements. Saves the position in the random number generator.

10.14.3 Examples

Listing 10.14: Saving a point in the generating process.

```
1   Program  Tests

3      USE  NumTypes
       USE  Statistics
5
       Integer , Parameter  ::  NN = 70
7      Real ( kind=SP )  ::  rr (NN)
       Integer  ::  Sd ( 2 5 )
9

11     CALL  Random_Number ( rr )
       CALL  GetLuxSeed ( Sd )
13     Open ( Unit =69, File=" seed . dat " )
       Write (69 ,∗) Sd
15

17     Stop
   End Program  Tests
```

10.15 Subroutine `Normal(X, [Rm], [Rsig])`

10.15.1 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.

10.15.2 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.

10.15.3 Examples

Listing 10.15: Obtaining numbers with a normal distribution.

```fortran
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
```

10.16 Subroutine `FishTipp(X, Rm, Rb)`

10.16.1 Description

Fills `X(:)` with numbers from a Fisher-Tippet distribution with parameters[2] `Rm`, and `2Rb`^{2}.

10.16.2 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).

Rb: Real (Single or Double precision).

---

[2]More info about this distribution in the Wikipedia: `http://en.wikipedia.org/wiki/Fisher-Tippett_distribution`

10.16.3   Examples

Listing 10.16: Obtaining numbers with a Fisher-Tippet distribution.

```fortran
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

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


  CALL FishTipp(X, 2.00_DP, 1.00_DP)
  ! Now compute the mean and standard deviation of the data
  Write(*,*) 'We should obtain 2.57721... and 1.2782...: '
  Write(*, '(ES33.25)')Mean(X), Stddev(X)


  Stop
End Program Tests
```

10.17   Subroutine `Laplace(X, Rm, Rb)`

10.17.1   Description

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

10.17.2   Arguments

`X(:)`: Real (Single or Double precision) one dimensional array. A vector that will be filled with numbers according to the Laplace 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 $2Rb^2$).

10.17.3   Examples

Listing 10.17: Obtaining numbers with a Laplace distribution.

```fortran
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

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

```fortran
10
     CALL Laplace(X, 1.23_DP, 1.0_DP)
12   ! Now compute the mean and standard deviation of the data
     Write(*,*) 'We should obtain 1.23 and sqrt(2): '
14   Write(*,'(ES33.25)')Mean(X), Stddev(X)

16

     Stop
18 End Program Tests
```

10.18   Subroutine/Function `Irand([Irnd], N, M)`

10.18.1   Description

If present, fills `Irnd(:)` with random integer numbers between `N` and `M` with an uniform distribution. If `Irnd(:)` is not present returns a integer random number between `N` and `M`.

10.18.2   Arguments

`Irnd(:):` Integer, Optional. A vector that will be filled with integer numbers according to a uniform distribution.

`N:` Integer. The minimum number that we can obtain.

`M:` Integer. The maximum number that we can obtain.

10.18.3   Examples

Listing 10.18: Obtaining integer random numbers.

```fortran
Program Tests
2
   USE NumTypes
4  USE Error
   USE Statistics
6
   Integer, Parameter :: Nmax = 100
8  Integer :: Irnd(Nmax)

10
   CALL Irand(Irnd, 0, 1)
12 ! Now compute the mean
   Write(*,*) 'We should obtain 0.5: '
14 Write(*,'(ES33.25)')Mean(Real(Irnd(:),kind=DP))

16

   Stop
18 End Program Tests
```

10.19 Subroutine `Permutation(Idx)`

10.19.1 Description

Returns a random permutation of `N` elements. It uses the Knuth shuffle algorithm[3].

10.19.2 Arguments

`Idx(:):`   Integer one dimensional array. Output. The random permutation.

Listing 10.19: Obtaining a permutation.

```fortran
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 10
  Integer :: Id(Nmax)


  CALL Permutation(Id)
  Write(*,'(100I3)')(Id(I), I = 1, Nmax)

  Stop

End Program Tests
```

10.20 Subroutine `BootStrap(Ibt)`

10.20.1 Description

Generates N_b bootstrap sequence of `N` numbers each. These bootstraps are returned in the two dimensional integer array `Ibt(:,:)` of size $N \times N_b$.

10.20.2 Arguments

`Ibt(:):` Integer. A two dimensional array of size $N \times N_b$, where N_b is the number of bootstraps, and `N` is the range of each bootstrap.

10.20.3 Examples

Listing 10.20: Resampling some data.

```fortran
Program Tests

  USE NumTypes
  USE Error
```

---

[3]`http://en.wikipedia.org/wiki/Knuth_shuffle`

```fortran
      USE Statistics

      Integer, Parameter :: Nmax = 8, Nbt = 5
      Real (kind=DP) :: X(Nmax)
      Integer :: Ib(Nmax, Nbt), I, J


      CALL Random_Number(X)
      ! Generate 5 bootstraps
      CALL BootStrap(Ib)

      ! Write the original sample, and the bootstraps
      Write(*,'(1000ES33.25)')(X(J), J=1, Nmax)
      Do I = 1, Nbt
          Write(*,'(1000ES33.25)')(X(Ib(J)), J=1, Nmax)
      End Do

      Stop
End Program Tests
```

10.21   Subroutine SaveBstrp(Ibt, Filename)

10.21.1   Description

Saves the bootstrap stored in Ibt and saves it in the file Filename.

10.21.2   Arguments

Ibt(:): Integer. A two dimensional array of size $N \times N_b$, where N_b is the number of bootstraps, and N is the range of each bootstrap.

Filename: Character (len=*). A file name to save the resampling data.

10.21.3   Examples

Listing 10.21: Reading the resampling info.

```fortran
Program Tests

   USE NumTypes
   USE Error
   USE Statistics

   Integer, Parameter :: Nmax = 8, Nbt = 5
   Integer :: Ib(Nmax, Nbt)


   ! Generate 5 bootstraps
   CALL BootStrap(Ib)

   ! Save it
```

```
15      SaveBstrp(Ibt, 'example.bst')

17      Stop
     End Program Tests
```

10.22 Subroutine `ReadBstrp(Ibt, Filename)`

10.22.1 Description

Reads the bootstrap stored in the file `Filename`, and returns it in `Ibt`.

10.22.2 Arguments

`Ibt(:):` Integer. A two dimensional array of size $N \times N_b$, where N_b is the number of bootstraps, and N is the range of each bootstrap.

`Filename:` Character (len=*). A file name to read the resampling data.

10.22.3 Examples

Listing 10.22: Saving the resampling info.

```
     Program Tests
2
        USE NumTypes
4       USE Error
        USE Statistics
6
        Integer, Parameter :: Nmax = 8, Nbt = 5
8       Integer :: Ib(Nmax, Nbt)

10
        ! Read a saved Bootstrap.
12      ReadBstrp(Ibt, 'example.bst')

14      Stop
     End Program Tests
```

10.23 Subroutine `EstBstrp(Data, Ibt, Func, Val, Err[, Rest])`

10.23.1 Description

Estimates using the Bootstrap method the average and error of an estimator given as a user supplied function.

10.23.2   Arguments

`Data(:):` Double precision Real. A one dimensional array with the original sampling.

`ibt(:,:):`   Integer two dimensional array. The bootstrap that we want to use to make the estimation.

`Func:`   A user suplied function that returns the value of the estimator. An interface block of the following type should be defined.

```
Interface
   Function Func(X)
     USE NumTypes

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

   End Function Func
End Interface
```

`Val:`   Double precision real. Output. The value of the estimation of the parameter.

`Err:`   Double precision real. Output. An estimation of the error in the estimation of the parameter.

`Rest:`   Double precision real one dimensional array (same dimension as the number of bootstraps in `Ibt`). Output. The value of the estimator for each resampling.

10.23.3   Examples

Listing 10.23: Estimating the average.

```fortran
Program Tests

  USE NumTypes
  USE Error
  USE Statistics

  Integer, Parameter :: Nmax = 100, Nbt = 50
  Integer :: Ib(Nmax, Nbt)
  Real (kind=DP) :: Avg, Err, Data(Nmax), Rest(Nbt)

  Interface
     Function F(X)
       USE NumTypes

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

     End Function F
  End Interface
```

```fortran
21    ! Read a saved Bootstrap, and the data from a file
      ReadBstrp(Ibt, 'example.bst')
23    Open (Unit=22, File="data.dat")
      Read(22,*)Data
25    Close(22)

27    ! And estimate the average
      CALL EstBstrp(Data, Ibt, F, Avg, Err, Rest)
29
      ! Print the Average of each resampling
31    Do I = 1, Nbt
         Write(*,*)I, Rest(I)
33    End Do

35    Stop
    End Program Tests
37
    Function F(X)
39    USE NumTypes
      USE Statistics
41
43    Real (kind=DP), Intent (in) :: X(:)
      Real (kind=DP) :: F
45
      F = Mean(X)
47
    End Function F
```

10.24 Subroutine `BstrpConfInt(Data, Ibt, alpha, Func, dmin, dpls))`

10.24.1 Description

Gives an Confidence interval for the estimator given as the user supplied function `Func`, such that

$$\mathcal{P}(dmin < Func(Data) < dpls) = 1 - 2\alpha$$

10.24.2 Arguments

`Data(:):` Double precision Real. A one dimensional array with the original sampling.

`ibt(:,:):` Integer two dimensional array. The bootstrap that we want to use to make the estimation.

`alpha:` Double precision real. The level of the confidence interval.

`Func:` A user suplied function that returns the value of the estimator. An interface block of the following type should be defined.

```
            Interface
               Function Func(X)
                  USE NumTypes

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

               End Function Func
            End Interface
```

dmin: Double precision real. Output. The lower limit of the confidence interval.

dpls: Double precision real. Output. The higher limit of the confidence interval.

10.24.3 Examples

Listing 10.24: Giving a confidence interval.

```fortran
1  Program Tests

3    USE NumTypes
     USE Error
5    USE Statistics

7    Integer, Parameter :: Nmax = 1000, Nb = 10000, Ndiv = 50
     Real (kind=DP) :: Rdata(Nmax), Rmean(Nb), avg, Xmin, Xmax, h, Xac,&
9        & err, dmin, dpls
     Integer :: Id(Nmax), Ib(Nb, Nmax), Ntics(Ndiv)

11
     Interface
13      Function F(X)
           USE NumTypes
15
           Real (kind=DP), Intent (in) :: X(:)
17         Real (kind=DP) :: F

19      End Function F
     End Interface
21
     CALL Normal(Rdata)
23
     avg = Mean(Rdata)
25   ! Now create the resamples
     CALL Bootstrap(Ib)
27
     CALL EstBstrp(Rdata, Ib, F, avg, Err, Rmean)
29   Write(*,*) '#', avg, Err, Mean(Rdata)

31   CALL BstrpConfInt(Rdata, Ib, 0.1_DP, F, dmin, dpls)
     Write(*,*) 'Intervalo:', dmin, dpls
33   Write(*,*)Avg - Dmin, Dpls - Avg
     Write(*,*)(dpls - dmin)/2.0_DP, 1.64485_DP/Sqrt(Real(Nmax,kind=DP))
```

```
35
     Stop
37
   End Program Tests
39
   Function F(X)
41   USE NumTypes
     USE Statistics
43
     Real (kind=DP), Intent (in) :: X(:)
45   Real (kind=DP) :: F
47   F = Mean(X)
49   Return
   End Function F
```

10.25   Function `Prop_Error(X, Dx, Func[, N])`

10.25.1   Description

Being `Func` a function of one or several variables, and `X` a point known with accuracy given by `Dx`, this routine propagate the errors in the position of the point, to obtain the accuracy of `F(X)`.

10.25.2   Arguments

`X[(:)]:` Double (DP) or simple (SP) precision one dimensional array or number. The value(s) of the variables.

`Dx[(:)]:` Double (DP) or simple (SP) precision one dimensional array or number. The value(s) of the error in the variables.

`Func:` A user supplied function. The function of the variables whose errors you want to propagate. An interface block of the following type should be defined.

```
        Interface
           Function Func(X)
             USE NumTypes

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

           End Function Func
        End Interface
```

`N:` Integer, Optional. The number of samples used to propagate the error. The default value is 1000.

10.25.3   Output

A real double or simple precision (same type as the input). An stimate of the accuracy of the value `F(X)`.

10.25.4   Examples

Here we compare our method of propagating errors, with the traditional propagation of errors formula for the case of the function $f(x, y) = x^2 + y^2$.

Listing 10.25: Propagating errors in a function.

```fortran
Program Tests

   USE NumTypes
   USE Statistics

   Real (kind=DP) :: Xm(2), Em(2)

   Interface
      Function F2(X)
        USE NumTypes
        Real (kind=DP), Intent (in) :: X(:)
        Real (kind=DP) :: F2
      End Function F2
   End Interface


   Xm(1) = 1.23_DP
   Xm(2) = 1.62_DP
   Em(1) = 0.02_DP
   Em(2) = 0.03_DP
   Write(*,*) 'Value of the function at Xm: ', F2(Xm)
   Write(*,*) ' Error estimate:            ', Prop_Error(Xm, Em, F2)
   Write(*,*) ' Error estimate by hand:    ', &
        & Sqrt((Em(1)/Xm(1))**2 + (Em(2)/Xm(2))**2)*F2(Xm)

   Stop
End Program Tests


Function F2(X)
   USE NumTypes

   Real (kind=DP), Intent (in) :: X(:)
   Real (kind=DP) :: F2

   F2 = X(1)**2 + X(2)**2

   Return
End Function F2
```

10.26 Subroutine `MCIntegration(X1, X2, Func, Val, Err[, Tol])`

10.26.1 Description

Uses the Monte Carlo method to stimate the value of the integral of the several variables function `Func` between the limits (`X1(1)`, `X1(2)`, ..., `X1(Ndim)`) and (`X2(1)`, `X2(2)`, ..., `X2(Ndim)`). The value of the integral is returned in `Val`, and an estimation of the error in `Err`. The desired precision can be introduced via the optional argument `Tol`.

10.26.2 Arguments

`X1(:):` Double or simple precision Real. A one dimensional array with the lower limits of the integrals.

`X2(:):` Double or simple precision Real. A one dimensional array with the upper limits of the integrals.

`Func:` A user suplied function that returns the value of the function to integrate. An interface block of the following type should be defined.

```
Interface
   Function Func(X)
     USE NumTypes

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

   End Function Func
End Interface
```

`Val:` Double or simple precision real (same as input). Output. An estimation of the value of the integral.

`Err:` Double or simple precision real (same as input). Output. An estimation of the error.

`Tol:` Double or simple precision real. Optional. An estimation of the desired precision.

10.26.3 Examples

Listing 10.26: Integrating a two dimensional function.

```
1  MODULE ExtData

3    USE NumTypes

5    Real (kind=DP) :: Sig = 2.345763450_DP
     Real (kind=DP) :: X1 = -0.324563562345643650_DP, X2 = 1.623473_DP, &
7        & Y1 = 1.34764574570_DP, Y2 = 5.3476347634_DP
```

```fortran
 9
   End MODULE ExtData
11
   Program MonteCarlo
13
      USE NumTypes
15    USE Error
      USE Statistics
17
      USE ExtData
19


21    IMPLICIT NONE

23    Real (kind=DP) :: Res, Err

25    Interface
         Function F(X)
27          USE NumTypes

29          Real (kind=DP), Intent (in) :: X(:)
            Real (kind=DP) :: F
31       End Function F
      End Interface
33
      CALL MCintegration((/X1,Y1/), (/X2,Y2/), F, Res, Err, 1.0E-4_DP)
35    Write(*,'(1A,100ES33.25)')'# Value of the integral: ', Res, Err


37

      Stop
39 End Program Monte

41

   Function F(X)
43
      USE NumTypes
45    USE ExtData

47    Real (kind=DP), Intent (in) :: X(:)
      Real (kind=DP) :: F
49


51    F = exp(-1.0_DP/(2.0_DP*Sig**2) * (Sum(X(:)**2)))*Sin(X(1)*X(2))

53    Return
   End Function F
```

Eleven

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.

11.1   Type `Pol`

11.1.1   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.

11.1.2   Components

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

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

11.1.3   Examples

A small example showing how to define a polynomial.

Listing 11.1: Defining a polynomial.

```
Program  TestPoly

   USE  NumTypes
   USE  Error
   USE  Polynomial

   Type (Pol) :: P1

   Stop
End Program  TestPoly
```

11.2   Type `CmplxPol`

11.2.1   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.

11.2.2   Components

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

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

11.2.3   Examples

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

Listing 11.2: Defining a polynomial.

```
Program  TestPoly

   USE  NumTypes
   USE  Error
   USE  Polynomial

   Type  (CmplxPol)  ::  P1

   Stop
End Program  TestPoly
```

11.3   Assignment

11.3.1   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.

11.3.2   Examples

Listing 11.3: Assigning polynomials.

```
Program  TestPoly

   USE  NumTypes
   USE  Error
   USE  Polynomial

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

```fortran
      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
```

11.4   Operator +

11.4.1   Description

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

11.4.2   Examples

Listing 11.4: Adding polynomials.

```fortran
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
24      Hcoef(5) =    1.0_DP

26      Hermite4 = Hcoef

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

34      Stop
    End Program TestPoly
```

11.5   Operator -

11.5.1   Description

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

11.5.2   Examples

Listing 11.5: Substracting polynomials.

```
1  Program TestPoly

3      USE NumTypes
       USE Error
5      USE Polynomial

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

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

17     Hermite3 = Hcoef(1:4)

19     ! The fourth Hermite polynomial is x^4 - 6x^2 + 3, so
       ! we first assign the values of the coefficients.
21     Hcoef    =    0.0_DP
       Hcoef(1) =    3.0_DP
23     Hcoef(3) = -6.0_DP
       Hcoef(5) =    1.0_DP
25
       Hermite4 = Hcoef

27
```

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

33

     Stop
35 End Program TestPoly
```

11.6   Operator *

11.6.1   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.

11.6.2   Examples

Listing 11.6: Computing the product of two polynomials.

```
1 Program TestPoly

3    USE NumTypes
     USE Error
5    USE Polynomial

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

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

17   Hermite3 = Hcoef(1:4)

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

25

     Hermite4 = Hcoef

27

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

```
          End Do
33
          Stop
35   End Program  TestPoly
```

11.7   Subroutine `Init(P, Dgr)`

11.7.1   Description

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

11.7.2   Arguments

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

Dgr: Integer. The degree of the polynomial.

11.7.3   Examples

Listing 11.7: Initialising a polynomial data type.

```fortran
1    Program  TestPoly

3      USE  NumTypes
       USE  Error
5      USE  Polynomial

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


11
       ! An  alternative  way  of  setting  the  third  Hermite
13     !  polynomial .
       CALL  Init (Hermite3 ,  3)
15     Hermite3%Coef (0)  =    0.0_DP
       Hermite3%Coef (1)  =  −3.0_DP
17     Hermite3%Coef (2)  =    0.0_DP
       Hermite3%Coef (3)  =    1.0_DP
19     Hermite3%dg  =  3


21
       Stop
23   End Program  TestPoly
```

11.8   Function `Degree(P)`

11.8.1   Description

Returns the degree of the polynomial P.

11.8.2 Arguments

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

11.8.3 Output

Integer. The degree of the polynomial P.

11.8.4 Examples

Listing 11.8: Returns the degree of a polynomial.

```fortran
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
```

11.9 Function `Value(P, X)`

11.9.1 Description

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

11.9.2 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.

11.9.3 Output

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

11.9.4 Examples

Listing 11.9: 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
```

```
32    ! different ways.
      X = 9.34564_DP
34    Write(*, '(ES33.25)') Value(Sum, X)
      Write(*, '(ES33.25)') Value(Hermite3, X)*Value(Hermite4, X)
36

38    Stop
    End Program TestPoly
```

11.10 Function `Deriv(P)`

11.10.1 Description

Computes the derivative of the polynomial `P`.

11.10.2 Arguments

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

11.10.3 Output

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

11.10.4 Examples

Listing 11.10: Computing the derivative of a polynomial.

```
1    Program TestPoly

3      USE NumTypes
       USE Error
5      USE Polynomial

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

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

17     Hermite3 = Hcoef(1:4)

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

```
25
       Hermite4 = Hcoef
27
       ! Now compute the derivative of Hermite4
29     Res = Deriv(Hermite4)

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

37

       Stop
39  End Program TestPoly
```

11.11   Function `Integra(P[, Cte])`

11.11.1   Description

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

11.11.2   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.

11.11.3   Output

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

11.11.4   Examples

Listing 11.11: Computing the integral of a polynomial.

```
1  Program TestPoly

3     USE NumTypes
      USE Error
5     USE Polynomial

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

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

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

17    Hermite3 = Hcoef(1:4)

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

25
      Hermite4 = Hcoef

27
      ! Now compute the derivative of Hermite4
29    Res = Integra(Hermite3,  3.0_DP/4.0_DP)

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

37
      Stop
39 End Program TestPoly
```

11.12   Function `InterpolValue(X, Y, Xo)`

11.12.1   Description

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

11.12.2   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.

11.12.3   Output

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

11.12.4   Examples

Listing 11.12: Compute values of the Interpolation polynomial.

```fortran
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
```

11.13   Function `Interpol(X, Y)`

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

11.13.1   Arguments

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

11.13.2   Output

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

11.13.3   Examples

Listing 11.13: Computes the interpolation polynomial.

```fortran
Program TestPoly

  USE NumTypes
  USE Error
  USE Polynomial
```

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


11

     CALL Random_Number(Xp)
13   Yp = 3.347234_DP*Xp - 2.475875_DP*Xp**3 - 7.23467_DP*Xp**4 + &
         & 1.47854_DP*Xp**6
15
     ! Now we compute the interpolation polynomial
17   ! at X, and compare it with the real value of the Polynomial
     X = -1.23899843_DP
19   Res = Interpol(Xp,Yp)
     Write(*,'(ES33.25)')Value(Res, X)
21   Write(*,'(ES33.25)')3.347234_DP*X - 2.475875_DP*X**3 - &
         & 7.23467_DP*X**4 + 1.47854_DP*X**6
23

25   Stop
   End Program TestPoly
```

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

11.14.1 Description

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

11.14.2 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.

11.14.3 Examples

Listing 11.14: Computes the cubic spline interpolation polynomial.

```
Program TestPoly
2
     USE NumTypes
4    USE Error
     USE Polynomial
6    USE NonNumeric

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

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

12
        CALL Random_Number(Xp)
14      ! Order Xp
        CALL Qsort(Xp)
16      Yp = 3.347234_DP*Xp - 2.475875_DP*Xp**3 - 7.23467_DP*Xp**4 + &
            & 1.47854_DP*Xp**6
18
        ! Now we compute the interpolation polynomial
20      ! at X, and compare it with the real value of the Polynomial, and
        ! the value of the spline cubic interpolation polynomial.
22      X = 0.23899843_DP
        Res = Interpol(Xp,Yp)
24      CALL Spline(Xp, Yp, 0.0_DP, 0.0_DP, Spl)
        Write(*, '(ES33.25)')Value(Res, X)
26      Write(*, '(ES33.25)')Value(Spl(Locate(Xp, X)), X)
        Write(*, '(ES33.25)')3.347234_DP*X - 2.475875_DP*X**3 - &
28          & 7.23467_DP*X**4 + 1.47854_DP*X**6

30
        Stop
32  End Program TestPoly
```

Twelve

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.

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

12.1.1 Description

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

12.1.2 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.

12.1.3 Examples

Listing 12.1: 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
```

```fortran
10      CALL  Random_Number(a)
12      CALL  Random_Number(b)
        CALL  Random_Number(c)
14      CALL  Random_Number(d)
        CALL  RootPol(a,b,z1,z2)
16      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))
18
        CALL  RootPol(a,b,c,  z1,z2,  z3)
20      Write(*,*)
        Write(*,'(3ES20.12)')Z1, Abs(z1**3+a*z1**2+b*z1+Cmplx(c,kind=DPC))
22      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))
24
        ac = Cmplx(a,kind=DPC)
26      bc = Cmplx(b,a,kind=DPC)
        cc = Cmplx(c,kind=DPC)
28      dc = Cmplx(d,kind=DPC)
        CALL  RootPol(ac,bc,z1,z2)
30      Write(*,*)
        Write(*,'(3ES20.12)')Z1, Abs(z1**2 + ac*z1 + Cmplx(bc,kind=DPC))
32      Write(*,'(3ES20.12)')Z2, Abs(z2**2 + ac*z2 + Cmplx(bc,kind=DPC))
        CALL  RootPol(ac,bc,cc,  dc,  z1,z2,  z3,  z4)
34      Write(*,*)
        Write(*,'(3ES20.12)')Z1, Abs(z1**4+ac*z1**3+bc*z1**2+cc*z1+dc)
36      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)
38      Write(*,'(3ES20.12)')Z4, Abs(z4**4+ac*z4**3+bc*z4**2+cc*z4+dc)

40
        Stop
42  End Program  TestRoot
```

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

12.2.1   Description

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

12.2.2   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

```fortran
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.

12.2.3   Output

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

12.2.4   Examples

Listing 12.2: Computing roots of non-linear functions with the Newton 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
      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
```

```fortran
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
```

12.3 Function `Bisec(a, b, Fbis, [Tol])`

12.3.1 Description

Compute the root of the function defined by `Fbis`.

12.3.2 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

```fortran
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.

12.3.3   Output

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

12.3.4   Examples

Listing 12.3: 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
```

Thirteen

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.

13.1 Type Fourier_Serie

13.1.1 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.

13.1.2 Components

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

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

13.1.3 Examples

A small example showing how to define a Fourier serie.

Listing 13.1: Defining a Fourier serie.

```
Program TestFourier

   USE NumTypes
   USE Constants
   USE Fourier

   Type (Fourier_Serie) :: Ff

   Stop
End Program TestPoly
```

13.2 Type Fourier_Serie_2D

13.2.1 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.

13.2.2 Components

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

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

13.2.3 Examples

A small example showing how to define a polynomial.

Listing 13.2: Defining a two-dimensional Fourier serie.

```
Program TestFourier

  USE NumTypes
  USE Constants
  USE Fourier

  Type (Fourier_Serie_2D) :: Ff

  Stop
End Program TestPoly
```

13.3 Assignment

13.3.1 Description

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

13.3.2 Examples

This example uses the `Init_Serie` subroutine. For details of the usage of this function look at the section (13.8), page (122).

Listing 13.3: 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)
```

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

14
        FS2 = FS1

16
        Write(∗,'(2ES33.25)')FS2%Coef(  1)
18      Write(∗,'(2ES33.25)')FS2%Coef(−1)


20      Stop
     End Program TestFourier
```

13.4   Operator +

13.4.1   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.

13.4.2   Examples

This example uses the `Init_Serie` subroutine. For details of the usage of this function look at the section (13.8), page (122).

Listing 13.4: Adding Fourier series.

```
1    Program TestFourier

3       USE NumTypes
        USE Constants
5       USE Fourier

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

9
        Nt = 4
11      CALL Init_Serie(FS1, Nt)
        CALL Init_Serie(FS2, Nt)

13
        FS1%Coef(  1,1) = Cmplx(1.0_DP,  0.5_DP, kind=DPC)
15      FS1%Coef(−1,1) = Cmplx(1.0_DP,  0.7_DP, kind=DPC)

17      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)

19

21      FS3 = FS1 + FS2
        Write(∗,'(2ES33.25)')FS3%Coef(  1,1)
23      Write(∗,'(2ES33.25)')FS3%Coef(−1,1)


25      Stop
     End Program TestFourier
```

13.5 Operator -

13.5.1 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.

13.5.2 Examples

Listing 13.5: Subtracting Fourier series.

```fortran
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
```

13.6 Operator *

13.6.1 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.

13.6.2 Examples

Listing 13.6: Computing the convolution of Fourier series.

```fortran
Program TestFourier

  USE NumTypes
  USE Constants
```

```
 5    USE Fourier

 7    Type (Fourier_Serie) :: FS1, FS2, FS3
      Integer :: Nt
 9
      Nt = 4
11    CALL Init_Serie (FS1, Nt)

13    FS1%Coef( 1) = Cmplx (1.0_DP, 0.5_DP, kind=DPC)
      FS1%Coef(−1) = Cmplx (1.0_DP, 0.7_DP, kind=DPC)
15
      FS2 = FS1
17
      FS3 = FS1 ∗ FS2
19    Write(∗, '(2ES33.25) ')FS3%Coef( 0)

21    Stop
    End Program TestFourier
```

13.7 Operator **

13.7.1 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.

13.7.2 Examples

Listing 13.7: "Exponentiating" Fourier series.

```
    Program TestFourier
 2
      USE NumTypes
 4    USE Constants
      USE Fourier
 6
      Type (Fourier_Serie) :: FS1, FS2, FS3
 8    Integer :: Nt

10    Nt = 4
      CALL Init_Serie (FS1, Nt)
12    CALL Init_Serie (FS2, Nt)
      CALL Init_Serie (FS3, Nt)
14
      FS1%Coef( 1) = Cmplx (1.0_DP, 0.5_DP, kind=DPC)
16    FS1%Coef(−1) = Cmplx (1.0_DP, 0.7_DP, kind=DPC)

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

20    FS2 = FS1∗∗8
```

```
         Do I = 1, 8
22           FS3 = FS3 * FS1
         End Do
24
         Write(*, '(2ES33.25) ')FS2%Coef( 0)
26       Write(*, '(2ES33.25) ')FS3%Coef( 0)

28       Stop
     End Program TestFourier
```

13.8   Subroutine `Init_Serie(FS,Ns)`

13.8.1   Description

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

13.8.2   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.

13.8.3   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 13.8: Initialising a Fourier series.

```
1  Program TestFourier

3    USE NumTypes
     USE Constants
5    USE Fourier

7    Type (Fourier_Serie) :: FS1, FS2, FS3
     Integer :: Nt
9
     Nt = 4
11   CALL Init_Serie(FS1, Nt)

13   FS1%Coef( 1) = Cmplx(1.0_DP, 0.5_DP, kind=DPC)
     FS1%Coef(-1) = Cmplx(1.0_DP, 0.7_DP, kind=DPC)
15
     FS2 = FS1
17
     FS3 = FS1 * FS2
19   Write(*, '(2ES33.25) ')FS3%Coef( 0)

21   Stop
     End Program TestFourier
```

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

13.9.1 Description

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

13.9.2 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.

13.9.3 Output

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

13.9.4 Examples

Listing 13.9: Evaluating a Fourier series at a point.

```fortran
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
28   End Program  TestFourier
```

13.10   Function `Unit(FS, Ns)`

13.10.1   Description

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

13.10.2   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.

13.10.3   Examples

Listing 13.10: Obtaining a constant Fourier series.

```fortran
Program  TestFourier
2
     USE  NumTypes
4    USE  Constants
     USE  Fourier
6
     Type ( Fourier_Serie )  ::  FS1, FS2, FS3
8    Integer  ::  Nt
10   Nt = 4
     CALL  Init_Serie (FS1,  Nt)
12   CALL  Init_Serie (FS2,  Nt)
     CALL  Init_Serie (FS3,  Nt)
14

16   FS1%Coef(  1) = Cmplx (1.0_DP,  0.5_DP,  kind=DPC)
     FS1%Coef(-1) = Cmplx (1.0_DP,  0.7_DP,  kind=DPC)
18
     CALL  Unit (FS2,  Nt)
20
     FS3 = FS1*FS2
22
     Write (*, '(2ES33.25) ') Eval_Serie (FS1,0.12_DP,1.0_DP)
24   Write (*, '(2ES33.25) ') Eval_Serie (FS3,0.12_DP,1.0_DP)
26   Stop
End Program  TestFourier
```

13.11 Function `DFT(Data, Is)`

13.11.1 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

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

the inverse one is defined as

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

13.11.2 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.

13.11.3 Output

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

13.11.4 Examples

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

Listing 13.11: 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(TWOPLDP*I/Nmax)
      Data(I) = Sin(X)
   End Do

   FS1 = DFT(Data)
```

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

23      Stop
      End Program TestFourier
```

13.12 Function `Conjg(FS)`

13.12.1 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 $\tilde{f}(-k)$.

13.12.2 Arguments

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

13.12.3 Output

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

13.12.4 Examples

Listing 13.12: Computing the Conjugate Fourier Series.

```
Program TestFourier
2
      USE NumTypes
4     USE Constants
      USE Fourier
6
      Integer, Parameter :: Nmax=20
8     Type (Fourier_Serie) :: FS1, FS2, FS3
      Complex (kind=DPC) :: Data(Nmax), X
10    Integer :: Nt

12    Do I = 1, Nmax
         X = Cmplx(TWOPI_DP*I/Nmax,kind=DPC)
14       Data(I) = Sin(X) + Cmplx(0.0_DP,I*2.0_DP,kind=DPC)
      End Do
16
      FS1 = DFT(Data)
18
      Write(*, '(2ES33.25)')Eval_Serie(FS1,0.23_DP,1.0_DP)
20    Write(*, '(2ES33.25)')Eval_Serie(Conjg(FS1),0.23_DP,1.0_DP)
```

```
22
       Stop
24  End Program  TestFourier
```

13.13   Subroutine `Save_Serie(FS, File)`

13.13.1   Description

Write the Fourier series `FS` to the file `File`.

13.13.2   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`.

13.13.3   Examples

Listing 13.13: Saving a Fourier Serie in a file.

```
    Program  TestFourier
2
       USE  NumTypes
4      USE  Constants
       USE  Fourier
6
       Integer , Parameter  ::  Nmax=20
8      Type ( Fourier_Serie )  ::  FS1, FS2, FS3
       Complex (kind=DPC)  ::  Data(Nmax), X
10     Integer  ::  Nt
12     Do  I = 1, Nmax
          X = Cmplx (TWOPI_DP*I/Nmax, kind=DPC)
14        Data(I) = Sin(X) + Cmplx(0.0_DP, I*2.0_DP, kind=DPC)
       End Do
16
       FS1 = DFT(Data)
18
       CALL  Save (FS1, 'datamodes.dat ')
20
       Stop
22  End Program  TestFourier
```

13.14   Subroutine `Read_Serie(FS, File)`

13.14.1   Description

Reads the Fourier series `FS` stored in the file `File`.

13.14.2 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.

13.14.3 Examples

Listing 13.14: 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
```

Fourteen

MODULE `Time`

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

14.1 Type `tm`

14.1.1 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...

14.1.2 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].

14.1.3 Example

A small example defining a `tm` data type.

Listing 14.1: Defining a Time data type.

```
Program Test

  USE NumTypes
  USE Time
```

129

```
6      Type  (tm)  ::  Oneday

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

16     Stop
    End Program  Test
```

14.2   Function `gettime()`

14.2.1   Description

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

14.2.2   Arguments

This function has no arguments.

14.2.3   Output

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

14.2.4   Example

A small program that prints the current year.

Listing 14.2: Obtaining the current date and time.

```
1   Program  Test

3      USE  NumTypes
       USE  Time
5
       Type  (tm)  ::  Oneday
7
       Oneday  =  gettime ()
9
       Write (∗,∗) 'Current  year:  ',  Oneday%year
11
       Stop
13  End Program  Test
```

14.3   Function `isleap(Nyr)`

14.3.1   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].

14.3.2   Arguments

`Nyr:`   Integer. The year.

14.3.3   Output

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

14.3.4   Example

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

Listing 14.3: Are we in a leap year?.

```
1  Program  Test

3    USE  NumTypes
     USE  Time

5
     Type  (tm)  ::  Oneday

7
     Oneday  =  gettime ()

9
     If  (isleap (Oneday%year ))  Then
11     Write ( ∗ ,∗ ) 'We  are  in  a  leap  year. '
     Else
13     Write ( ∗ ,∗ ) 'We  are  not  in  a  leap  year. '
     End If

15
     Stop
17 End Program  Test
```

14.4   Function `asctime(t)`

14.4.1   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
```

14.4.2 Arguments

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

14.4.3 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.

14.4.4 Example

A small program that prints the current time.

Listing 14.4: Printing current date/time.

```
1  Program Test

3    USE NumTypes
     USE Time
5

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

9

     Stop
11 End Program Test
```

14.5 Function `Day_of_Week(Day, Month, Year)`

14.5.1 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`.

14.5.2 Arguments

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

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

Year: Integer. The year.

14.5.3 Output

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

14.5.4   Example

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

Listing 14.5: Day of week of the first of January 1900.

```fortran
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

Copyright © 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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.

135

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

[2] W. J. Cody, *Rational Chebyshev approximations for the error function*, *Math. Comp.* (1969) 631–638.

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

[4] M. Luscher, *A Portable high quality random number generator for lattice field theory simulations*, *Comput. Phys. Commun.* **79** (1994) 100–110, [`hep-lat/9309020`].

Index

145