

November 23, 2009

**MULTISTAGE WEIBULL TIME-TO-TUMOR MODEL
IN EPA'S BENCHMARK DOSE SOFTWARE (BMDS)**

SOURCE CODE INSTALLATION AND DESCRIPTION

**BATTELLE
505 King Avenue
Columbus, OH 43201-2693**

**EPA Contract Number EP-C-05-030
Work Assignment 4-17**

Prepared for

**John Fox, Work Assignment Manager
National Center for Environmental Assessment
Office of Research and Development
U.S. ENVIRONMENTAL PROTECTION AGENCY
Washington, DC 20460**

**Marla Smith, Project Officer
Engineering and Analysis Division
Office of Science and Technology
Office of Water
U.S. ENVIRONMENTAL PROTECTION AGENCY
Washington, DC 20460**

EPA Disclaimer

The material in this document has not been subject to Agency technical and policy review. Views expressed by the authors are their own and do not necessarily reflect those of the U.S. Environmental Protection Agency. Mention of trade names, products, or services does not convey, and should not be interpreted as conveying, official EPA approval, endorsement, or recommendation. Do not quote or cite this document.

Battelle Disclaimer

This report is an interim work prepared for the United States Government by Battelle and is for discussion purposes only. In no event shall either the United States Government or Battelle have any responsibility or liability for any consequences of any use, misuse, inability to use, or reliance upon the information contained herein, nor does either warrant or otherwise represent in any way the accuracy, adequacy, efficacy, or applicability of the contents hereof.

Table of Contents

	<u>Page</u>
1.0 INTRODUCTION AND OBJECTIVES	1
1.1. Intended Audience	1
2.0 COMPILING AND INSTALLING THE SOURCE CODE	2
2.1. Installing the MinGW GNU Compilers and Other Utilities	2
2.2. Installing BMDS	3
2.3. Compiling the Multistage Weibull Model	4
2.4. Setting the Silent Mode option in donlp3	6
3.0 COMPONENT FILES WITHIN THE SOURCE CODE	6
3.1. Relationships Among Component Files	6
3.2. Descriptions of Component Files.....	9
3.2.1. Components <i>model.h</i> and <i>model.c</i>	9
3.2.2. Components <i>input.h</i> and <i>input.c</i>	10
3.2.3. Components <i>process.h</i> and <i>process.c</i>	11
3.2.4. Components <i>check.h</i> and <i>check.c</i>	11
3.2.5. Components <i>estimate.h</i> and <i>estimate.c</i>	12
3.2.6. Components <i>output.h</i> and <i>output.c</i>	15
3.2.7. Components <i>utilities.h</i> and <i>utilities.c</i>	15
3.2.8. Components <i>gev.h</i> and <i>gev.c</i>	20
3.2.9. Components <i>donlpfun.h</i> and <i>donlpfun.c</i>	28

List of Figures

	<u>Page</u>
Figure 1. Diagram describing the file component relationships	8

MULTISTAGE WEIBULL TIME-TO-TUMOR MODEL IN EPA'S BENCHMARK DOSE SOFTWARE (BMDS)

SOURCE CODE INSTALLATION AND DESCRIPTION

1.0 INTRODUCTION AND OBJECTIVES

A time-to-tumor model describes the probability of a test subject exhibiting a tumor response, such as tumor onset or death from tumor, by a specified observation time t when the subject is exposed to a toxin at a given dosage rate. A computer module has been developed to allow for time-to-tumor modeling within EPA's Benchmark Dose Modeling Software (BMDS) System using a multistage Weibull model. This document explains how to compile and install the source code for Version 1.6 of the BMDS time-to-tumor module. It is a technical presentation that describes the relationships between the component files in the source code, and it specifies the declarations and functions contained in each individual component file.

This document is one of a series of documents that details the development and use of the time-to-tumor model within BMDS. The other documents in this series include:

- "Multistage Weibull Time-to-Tumor Model in EPA's Benchmark Dose Software (BMDS): Methodology Description," which details the algorithms and statistical methodology used to fit the model, estimate model parameters and the benchmark dose (BMD), and calculate profile likelihood confidence intervals.
- "Multistage Weibull Time-to-Tumor Model in EPA's Benchmark Dose Software (BMDS): Testing Document," which presents the approach and outcome of validation and verification efforts associated with the development of this module within BMDS.
- "Multistage Weibull Time-to-Tumor Model in EPA's Benchmark Dose Software (BMDS): Basic User Installation and Guide," which describes the installation process, the command line execution of the module, and the format of the input and output files for this module.

1.1. Intended Audience

This document is written for users who wish to examine, compile, and perhaps modify the source code for the BMDS multistage Weibull time-to-tumor module. To ensure a sufficient understanding and benefit from the contents of this document, the reader should have familiarity with the following:

1. Intermediate familiarity with C programming and basic familiarity with Makefile scripting.
2. Knowledge of Microsoft Windows XP at an intermediate level. Familiarity with the use of command shell and modification of internal settings.

3. Intermediate-to-advanced familiarity with mathematics and statistics, along with some familiarity with the methodology and algorithms in the time-to-tumor module, as described in the Methodology Description document.

Familiarity with the base BMDS source code is not required, because the module was developed as a semi-independent entity from other modules in the system.

For instructions on installing the pre-compiled version of the module, or on setting up input files and running the module, please consult the Basic User Installation and Guide document.

2.0 COMPILING AND INSTALLING THE SOURCE CODE

The module described in this document adds a multistage Weibull time-to-tumor modeling capability to BMDS. The current release of the module (Version 1.6) will generate estimates for model parameters (with asymptotic standard errors and correlation matrix), the BMD (with profile likelihood confidence interval), and an analysis of deviance table. Other targeted module capabilities, including slope estimation, may be activated in future releases as development and testing are completed.

The source code installation instructions presented in this document are for users running the Windows XP SP2 operating system with Internet connection. The user may be required to have administrative privileges in order to install this module. (Please consult your IT personnel regarding such privileges.)

2.1. Installing the MinGW GNU Compilers and Other Utilities

Because it was created semi-independently from other BMDS modules, the BMDS multistage Weibull time-to-tumor module can potentially be integrated with any BMDS version. However, in terms of input and output file structures, the module was initially developed to be consistent with a beta release of BMDS Version 1.4. It was compiled using gcc-3.4.2. All development and testing was performed using Windows XP Service Pack (SP) 2.

Links to the MinGW downloads are available at <http://www.mingw.org/download.shtml#hdr1>. The following steps are used to unpack and install the compilers and other required executable utilities:

1. Download the MinGW Setup Wizard. The recommended versions are 5.0.0 or higher (MinGW-5.0.0.exe, etc.).
2. Run the MinGW Setup Wizard. Note the directory where you choose to install MinGW, e.g, in C: \Mi nGW. The directory name will be referenced in this documentation as (Mi nGW_HOME).

3. Select the required MinGW components for download, unpacking, installation. In the Wizard, select the current version of MinGW, full installation, and the following components:
 - a. base tools (includes w32api, binutils, and gcc-core)
 - b. gcc-g++ compiler
 - c. gcc-g77 compiler
 - d. mingw32-make

The other components (ada, java and objective C compilers) and compiler source codes are not required. (Although it is possible to separately download the components individually, the Wizard simplifies installation and removal, and avoids any difficulties with unpacking tar / gzip files using Winzip.)

4. Edit the executable path, if required. To examine the path, execute the following sequence:

My Computer [right click] > Properties > Advanced [tab] > Environment Variables [button] > System Variables [scroll box] > Path [select and highlight] > Edit [button].

If the Variable value box does not include the MinGW executables directory (Mi nGW_HOME)\bi n, edit the string by adding in the directory name at the beginning, ending with a semicolon (;) separator. The variable value should look something like:

```
C: \Mi nGW\bi n; %SystemRoot%\system32; %SystemRoot%; %SystemRoot%\Syst
em32\Wbem; C: \ORANT\BI N
```

Optional: Rename mingw32-make.exe to make.exe in (Mi nGW_HOME)\bi n. Make sure that it does not clash with another executable also called make.exe in any of the path directories. Then, to run “Make”, type ‘make’ instead of ‘mi ngw32-make’ at the command prompt.

2.2. Installing BMDS

To install BMDS, download the ZIP file containing the BMDS software from the EPA website and unzip the contents of the ZIP file into a user selected file (e.g. “C:\Documents and Settings\jd\BMDS”). The directory name for the file will be referenced in this document as (BMDS_HOME). Check to make sure that both donlp3 and as274 have been installed by looking in the Assist subfolder, which should include both subfolders donlp3 and as274_fc. Otherwise, unzip the donlp3.zip and as274_fc.zip into directories (BMDS_HOME)\Assi st\donl p3 and (BMDS_HOME)\Assi st\as274_fc, respectively. In the last step, unzip MSW_src.zip into (BMDS_HOME)\Ti me-to-Tumor.

2.3. Compiling the Multistage Weibull Model

The steps to compiling the multistage Weibull time-to-tumor module within BMDS are as follows. Illustrations provided within a given step include command prompts, user entries provided at these prompts (specified in bold), and results generated by these user entries.

1. Edit the file Makefile.conf in (BMDS_HOME). Verify that the following settings are specified for the following seven variables within the file **Makefile.conf**. If any settings need to be changed, make the appropriate changes to this file using a text editor:

```
CC = gcc (or CC = gcc32 if gcc32.exe is available in (Mi nGW_HOME)\bi n)
LN = g77
STRIP = strip
RM = del
CP = copy
MORE_CFLAGS = -DWI N32
BMDS_HOME = . .
```

2. Open up a command line window. Execute the sequence Start > Run..., then type 'cmd' in the box.
3. Go to the (BMDS_HOME) directory. Type the cd command at the command prompt along with the name of the directory in which BMDS was installed in order to go to this directory:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\j d> cd "\BMDS"

C:\Documents and Settings\j d\BMDS>
```

4. Check the path to the MinGW executables. Type 'echo %path%' at the command prompt. The returned value should include the directory of MinGW executables, like the example in Step 4 of Section 2.1. If it does not, add to the path (only for the current Windows session) by pasting the following command in the command window, replacing "(MinGW_Home)" by the actual pathname

```
> path (Mi nGW_Home)\bi n; (Mi nGW_Home); %PATH%
```

5. Check installation and compilation of the BMDS archive library. Type the dir command at the command prompt in order to list the contents of the "Assist" subdirectory (or check the contents of this subdirectory within Windows Explorer):

```
C:\Documents and Settings\j d\BMDS> di r Assi st

12/12/2005  08: 56 AM    <DI R>          .
12/12/2005  08: 56 AM    <DI R>          ..
11/17/2005  09: 37 AM                   7,118 al l o_memo. c
11/17/2005  09: 37 AM                   385 al l o_memo. h
12/12/2005  08: 55 AM    <DI R>          as274_fc
```

etc....

Verify that this list includes the file `libassist.a`, and **no files with extension '.o'**. If this verification fails,

- a. If `libassist.a` does not exist in this subdirectory, go to the `Assist` subdirectory (using the `cd` command) and “Make” the archive library in the command line window:

```
C:\Documents and Settings\jd\BMDs> cd Assist
```

```
C:\Documents and Settings\jd\BMDs\Assist> mingw32-make
```

```
gcc32 -Wall -g -DWIN32 -DASSIST -c -o confint.o confint.c
gcc32 -Wall -g -DWIN32 -DASSIST -c -o ERRORPRT.o ERRORPRT.c
gcc32 -Wall -g -DWIN32 -DASSIST -c -o gettimeofday.o
gettimeofday.c
etc....
```

- b. Verify that **libassist.a** has been created.
 - c. Delete all files with extension `.o` by typing `del *.o` at the command prompt.
 - d. Return to (BMDs_HOME), going up one parent directory by typing `cd ..` at the command prompt.
6. Unzip the ZIP file containing the multistage Weibull time-to-tumor source code and executable. Place the ZIP file named `MSW.ZIP` into the `BMDs_HOME` folder in which you will install the file. Extract the contents of `MSW.ZIP` within this folder by executing the following sequence:
- a. Right click on the file “`MSW.ZIP`.”
 - b. Select the **WinZip** option.
 - c. Left click on the “Extract to Here” option.

The folder should include the executable program `msw.exe` and a subfolder `Test Inputs`.

7. Compile the multistage Weibull time-to-tumor executable. Go to subdirectory (BMDs_HOME)\Time-to-tumor (using the `cd` command) and “Make”, e.g.,

```
C:\Documents and Settings\jd\BMDs> cd Time-to-Tumor
```

```
C:\Documents and Settings\jd\BMDs\Time-to-Tumor> mingw32-make
```

```
gcc32 -ansi -O3 -ffloat-store -I../Assist/donlp3 -
I../Assist/as274_fc/C -I../Assist -c -o utilities.o utilities.c
gcc32 -ansi -O3 -ffloat-store -I../Assist/donlp3 -
I../Assist/as274_fc/C -I../Assist -c -o input.o input.c
etc.....
```

8. Move executable and other files to the appropriate locations. Move the executable `msw.exe` to the directory (BMDs_HOME)\bin. Move the folder `Test Inputs` to the directory (BMDs_HOME)\TestData, and rename the folder `MSW`.

2.4. Setting the Silent Mode option in donlp3

The **donlp3** optimizer can be used in two different modes that differ based on the level of performance messages which are sent to the standard output. When the optimizer runs in Silent mode, neither the results protocol nor the messages protocol is written to the standard output. The level of optimizer messages written to output can be controlled by setting the following parameters in the **model.h** file:

- **DONLP_SILENT** – 1/0 (TRUE/FALSE). The Silent mode can be turned on by setting this parameter to 1/TRUE. The default and recommended mode of operation for the **donlp3** optimizer is 0/FALSE.
- **DONLP_TE0** – 1/0(TRUE/FALSE). In combination with the **SILENT** parameter, this parameter controls the level of messages sent to the standard output. The recommended setting for this parameter is 1/TRUE.

When running the multistage Weibull module, the recommended values for these two parameters are **DONLP_SILENT** = 0/FALSE and **DONLP_TE0** = 1/TRUE.

3.0 COMPONENT FILES WITHIN THE SOURCE CODE

3.1. Relationships Among Component Files

Figure 1 shows the major relationships between most of the component files in the ANSI C software for the BMDS multistage Weibull time-to-tumor module.

The header file **model.h** appears at the top of the component file relationship structure. It contains **#include** statements for headers of ANSI C standard libraries, definitions of global constants, definitions and declarations of global structures and variables. The contents of **model.h** are commonly accessed and used throughout the software. Therefore, they are included in all program files. Containing the **main()** function, the program file **model.c** is at the center of the relationship structure. The **main()** function serves as the “backbone” of the relationship structure, because it specifically lays out the sequence in which the software carries out particular tasks. These tasks are split across the following five program files:

1. **input.c**: Inputs (most of) the model information and all the data from the input batch file.
2. **check.c**: Checks the model information and the data for consistency.
3. **process.c**: Pre-processes the input information and data to prepare for estimation.

4. `estimate.c`: Carries out all estimations (e.g., model parameters, confidence intervals, BMD)
5. `output.c`: Prints all the results into output file.

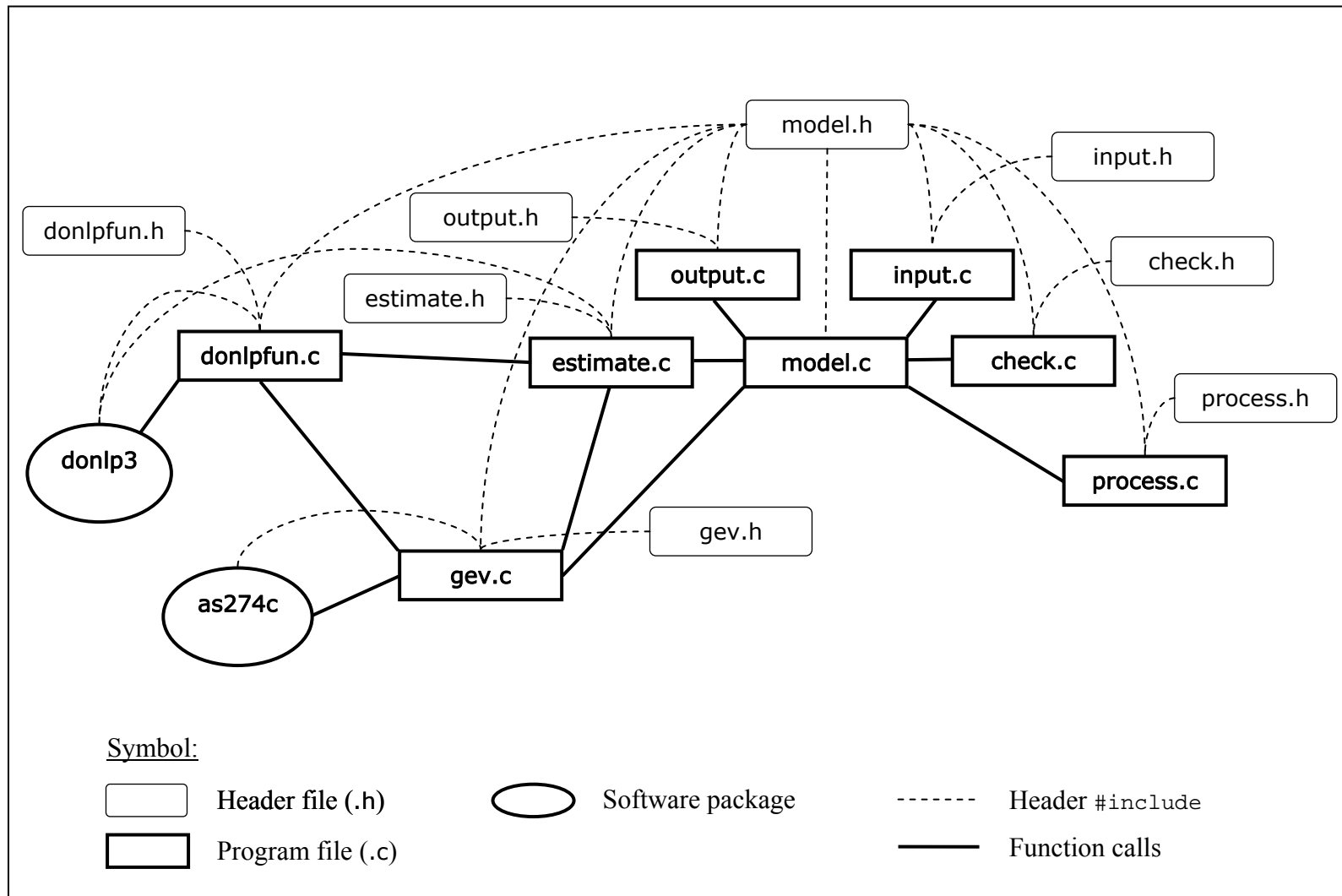


Figure 1. Diagram describing the file component relationships

[Note: The graph excludes utilities.h, utilities.c, the BMDS library libassist.a, and other program components (e.g., ANSI C standard library).]

Three additional program files exist:

- `donlpfun.c`: Contains functions that interface to the donlp3 optimization software.
- `gev.c`: Contains functions that are specific to the multistage Weibull/GEV model, e.g., log-likelihoods, automatic starting values, etc.
- `utilities.c`: Contains general utility functions.

The header file `utilities.h` is also included in all other program files. This header file avoids the need for an `extern` statement when accessing functions in `utilities.c`, which are used by different functions in the other files. Note that `utilities.h` and `utilities.c` have been excluded from Figure 1 so as not to obscure the major relationships between the file components.

In addition to the component files, the BMDS multistage Weibull time-to-tumor module also requires the public domain software packages donlp3 (for optimization) and asc274c (for linear regression). Figure 1 shows both “header `#include`” and “function call” links for the software packages. Both include header files (`.h`) because they were written in ANSI C.

Note that for clarity, the BMDS library `libbassi.st.a` as well as the standard ANSI C libraries, have been excluded from Figure 1.

3.2. Descriptions of Component Files

3.2.1. Components `model.h` and `model.c`

The header file `model.h` consists of:

- a. `#include <*****.h>` statements for ANSI C library headers,
- b. Definitions of preprocessor constants and functions, and
- c. Definitions or declarations of global structures and variables.

Note that `model.h` does *not* include any declarations for functions, and all program files (`.c`) incorporate the header file with the `#include "model.h"` statement.

The `model.c` file only consists of a single function:

- a. **`void main(int argc, char *argv[])`**
Description: “Backbones” the software module. Reads the single argument to the executable containing the character string the batch input filename (`argv[1]`), opens up the input stream, determines model type, sets global functions for the model type, and calls (external or global) functions in the following sequence:

- | | |
|-----------------------------------|--|
| 1. <code>input()</code> : | Input information and analysis data in the batch file. |
| 2. <code>check()</code> : | Carry out consistency checks of the input information and analysis data. |
| 3. <code>process()</code> : | Process the input information and analysis data. |
| 4. <code>g_s_fun.init()</code> : | Initialize and set up the model, e.g., set parameter names. |
| 5. <code>mle()</code> : | Fit the model by maximum likelihood estimation, estimate BMD, and calculate profile likelihood confidence intervals. |
| 6. <code>g_s_fun.clear()</code> : | Clean up the model results, e.g., free dynamic memory, reverse re-parameterization. |
| 7. <code>output()</code> : | Output the results |
| 8. <code>process_clr()</code> : | Free dynamic memory (data summary structures) |
| 9. <code>input_clr()</code> : | Free dynamic memory (data structure) |

Arguments:

- | | |
|-----------------------------|---|
| <code>int argc</code> : | Number of arguments in executable (should equal 2). |
| <code>char *argv[]</code> : | Array, size <code>argc</code> , of character strings. <code>argv[1]</code> should contain (directory name and) batch file name. |

3.2.2. Components *input.h* and *input.c*

The header file `input.h` contains declarations for functions stored within `input.c`.

The `input.c` file consists of the following functions:

- a. **`void input(FILE *p_file)`**
Description: Calls `info_in()`, dynamically allocates space for the data structure, calls `data_in()`, and reallocates space for data structure to adjust for the number of observations in the data.
Arguments:

<code>FILE *p_file</code> :	Pointer to file input stream.
-----------------------------	-------------------------------
- b. **`void input_clr(void)`**
Description: Frees the dynamically allocated space in the data structure.
- c. **`void info_in(FILE *p_file)`**
Description: Reads in modeling information from the batch file.
Arguments:

FILE *p_file: Pointer to file input stream.

d. **void data_in(FILE *p_file)**

Description: Reads in data from the batch file.

Arguments:

FILE *p_file: Pointer to file input stream.

3.2.3. **Components process.h and process.c**

The header file process.h contains declarations for functions stored in process.c.

The process.c file consists of the following functions:

a. **void process(void)**

Description: Dynamically allocates space for parameter MLEs asymptotic covariance matrix, calls info_prc(), dynamically allocates space for data summary structures, calls data_prc(), and reallocates space for data summary structures to adjust for the number of category levels in the data.

b. **void process_clr(void)**

Description: Frees the dynamically allocated space parameter MLEs asymptotic covariance matrix and the data summary structures.

c. **void info_prc(void)**

Description: Processes modeling information, e.g., counts number of free parameters.

d. **void data_prc(void)**

Description: Processes data, grouped by dosage and response categories.

3.2.4. **Components check.h and check.c**

The header file check.h contains declarations for functions stored in check.c.

The check.c file consists of the following functions:

a. **void check(void)**

Description: Calls info_chk() and data_chk().

b. **void info_chk(void)**

Description: Checks modeling information for consistency, e.g., model specification, I/O files, optimization inputs, BMD calculations.

c. **void data_chk(void)**

Description: Checks data input, in terms of both expected data size and individual data elements.

3.2.5. Components *estimate.h* and *estimate.c*

The header file *estimate.h* consists of

- a. Declarations for structures used in profile likelihood confidence interval calculations, and
- b. Declarations for functions in *check.c*.

The *check.c* file consists of the following functions:

a. **struct s_donlp3_return s_donlp_ml(double *pd_start)**

Description: Maximizes the log-likelihood. Sets starting values in global optimization input structure, and calls (external or global) functions in the following sequence:

1. `reset_eval()`: Reset the evaluation of the log-likelihood.
2. `g_s_fun.scale()`: Set internal scaling values.
3. `donlp3()`: Carry out the optimization.
4. `reset_scale()`: Reset internal scaling values.

Arguments:

double *pd_start: Pointer to array of starting values (free model parameters only).

b. **void mle(void)**

Description: Carries out all MLE calculations. Calls the following functions in sequence:

1. `parm_ml()`: Calculate the MLEs of free parameters.
2. `g_s_fun.n_inv_cov()`: Calculates the $(-2 \times)$ Hessian of the log-likelihood at the MLEs.
3. `INVMAT()`: Inverts the $(-2 \times)$ Hessian to calculate asymptotic covariance matrix.
4. `g_s_fun.start_reset()`: Reset starting values (if `g_s_mle.s_bmd.b_calc = 1`)
5. `bmd()`: Carries out the BMD calculations (if `g_s_mle.s_bmd.b_calc = 1`).

c. **void parm_ml(double *pd_l, double *pd_fixed, size_t un_parm, unsigned int b_init, size_t un_start, double *pd_start, unsigned int b_store)**

Description: Calculates the MLEs of the free parameters. Calls the following functions in sequence:

1. `g_s_fun.set()`: Sets up the model log-likelihood.
2. `g_s_fun.start()`: Calculates automatic starting values (if `b_init = 0`).
3. `donlp_init()`: Sets the optimization input parameters (mostly to default).
4. `s_donlp_ml()`: Maximizes the log-likelihood `un_start` times, using different sets of starting values.

Subsequently, the result with the largest log-likelihood goes through the following additional refinement:

5. `g_s_fun_refine()`: Sets up the refinement, e.g., fixing free parameter values that are very close to their boundaries.
6. `g_s_fun.set()`: Sets up the model log-likelihood.
7. `s_donlp_ml()`: Maximizes the log-likelihood.

Arguments:

- | | |
|------------------------------------|--|
| <code>double *pd_l:</code> | Pointer for returning the maximum of the log-likelihood function. |
| <code>double *pd_fixed:</code> | Pointer to array of fixed model parameters. |
| <code>size_t un_parm:</code> | Number of model parameters. |
| <code>unsigned int b_init:</code> | Starting value option indicator (0 = automatic, 1 = user specified). |
| <code>size_t un_start:</code> | Number of starting value sets. |
| <code>double pd_start:</code> | Pointer to array of starting values (if <code>b_init = 1</code>). |
| <code>unsigned int b_store:</code> | Starting value storage indicator (0 = do not store, 1 = store). |

d. **`void bmd(double *pd_start)`**

Description: Calculates the MLE of the *BMD*. Carries out a root search using the `zeroin()` function on `d_bmd_zi()`, the defining function for *BMD*. Calls `pl_conf()` function to calculate the profile likelihood confidence interval if `g_s_mle.s_bmd.s_conf.b_calc = 1`.

Arguments:

- | | |
|--------------------------------|--|
| <code>double *pd_start:</code> | Pointer to array of starting values (free model parameters only) for profile likelihood confidence interval calculations, i.e., <code>pl_conf()</code> function. |
|--------------------------------|--|

e. **`void pl_conf(struct conf *ps_conf, double *pd_parm, size_t un_parm, double d_mle, double *pd_start, double *pd_lim)`**

Description: Calculates the profile likelihood confidence interval for a parameter MLE (e.g., the *BMD*).

Arguments:

<code>struct conf *ps_conf:</code>	Pointer to profile likelihood calculation structure for inputs (e.g., confidence level) and outputs (e.g., confidence bounds).
<code>double *pd_parm:</code>	Pointer to array of input parameters in parameter defining function calculations (e.g., benchmark risk level for <i>BMD</i>).
<code>size_t un_parm:</code>	Number of input parameters in parameter defining function calculations.
<code>double d_mle:</code>	MLE of the parameter.
<code>double *pd_start:</code>	Pointer to array of starting values.
<code>double *pd_lim:</code>	Pointer to 2-dimensional array, specifying the search boundary limits for the profile likelihood confidence bounds.

f. **`struct s_donlp3return s_donlp_pl(double *pd_start, double d_x)`**

Description: Calculates the profile log-likelihood function.

Arguments:

<code>double *pd_start:</code>	Pointer to array of starting values, returning optimized parameters.
<code>double d_x:</code>	Parameter value.

Return: Structure of type `s_donlp3return`, containing the profile log-likelihood value, the optimized free parameters, and returned optimum and diagnostics from `donl p`.

g. **`double d_pl_zi(int nparm, double parm[], double x, double gtol)`**

Description: Defines the function used by the root search function `zeroin()` to find the profile likelihood confidence bounds for a parameter.

Arguments:

<code>int nparm:</code>	Number of parameters, set to the number of free model parameters.
<code>double parm[]:</code>	Array of length $3 * nparm + 1$.
<code>double x:</code>	Parameter value.
<code>double gtol:</code>	Tolerance level for root search convergence (not used).

Note: the array `parm[]` consists of the following components:

<code>[0 to nparm-1]</code>	Optimized free model
and	parameters of the profile log-
<code>[nparm to 2*nparm-1]</code>	likelihood at the search
	interval boundaries for the
	current root search algorithm

[2*nparm to 3*nparm-1]

[3*nparm]

iteration. One of the two components is updated with a function call to `d_pl_zi`, the choice depending on whether the profile log-likelihood at x is above or below the cut-off.

Array used to store the midpoint of the two sets of free model parameters, i.e., the starting values in the constrained optimization to evaluate the profile log-likelihood at x .

Cut-off value for the profile log-likelihood

Return: Value of the profile log-likelihood function at x minus the cut-off value.

3.2.6. Components *output.h* and *output.c*

The header file `output.h` contains declarations of the following function stored in `output.c`:

a. **`void output(void)`**

Description: Opens the output stream to the `.out` file specified in the input batch file, outputs the modeling results into the file, and closes the output stream.

3.2.7. Components *utilities.h* and *utilities.c*

The header file `utilities.h` contains declarations for functions stored in `utilities.c`. This header file is included in all `.c` files, because `utilities.c` contains utility functions that are used throughout all `.c` files.

The `utilities.c` file consists of the following functions:

Functions for messaging

a. **`void error(const char *sz_file, const int n_line, const char *fmt, ...)`**

Description: Prints error messages to `stderr` and terminates the program by calling the `exit()` function.

Arguments:

<code>const char *sz_file:</code>	Pointer to character string with filename where the error occurred.
<code>const int n_line:</code>	Line number where the error occurred.

<code>const char *fmt:</code>	Pointer to character string with the error message.
<code>....:</code>	Arguments printed in the error message.

- b. **`void print(const char *sz_file, const int n_line, const char *fmt, ...)`**

Description: Prints messages to stderr.

Arguments:

<code>const char *sz_file:</code>	Pointer to character string with filename where the error occurred.
<code>const int n_line:</code>	Line number where the error occurred.
<code>const char *fmt:</code>	Pointer to character string with the error message.
<code>....:</code>	Arguments printed in the error message.

Functions for model parameters

- c. **`void parm_set(struct parm *ps_parm, const double *pd_fixed, char **psz_name, size_t un_value)`**

Description: Sets up the model parameter structure addressed by the pointer *ps_parm.

Arguments:

<code>struct parm *ps_parm:</code>	Pointer to address of model parameter structure.
<code>const double *pd_fixed:</code>	Pointer to array of fixed model parameters.
<code>char **psz_name:</code>	Pointer to array of character strings with the model parameter names.
<code>size_t un_value:</code>	Number of model parameters.

- d. **`void parm_ins(struct parm *ps_parm, double *pd_free)`**

Description: Inserts free model parameter values in array pd_free into model parameter structure addressed by the pointer *ps_parm.

Arguments:

<code>struct parm *ps_parm:</code>	Pointer to address of model parameter structure.
<code>double *pd_free:</code>	Pointer to array of free model parameters.

- e. **`void parm_free_ext(double *pd_free, struct parm *ps_parm)`**

Description: Extracts free model parameter values from model parameter structure addressed by the pointer *ps_parm into array pd_free.

Arguments:

<code>double *pd_free:</code>	Pointer to array of free model parameters.
-------------------------------	--

struct parm *ps_parm: Pointer to address of model parameter structure.

f. **void parm_print (struct parm *ps_parm, unsigned int un_flag)**

Description: Prints model parameter values in model parameter structure addressed by the pointer *ps_parm to stdout.

Arguments:

struct parm *ps_parm: Pointer to model parameter structure.
unsigned int un_flag: Flag for print detail options (0 = Basic, 1 = Detailed).

g. **size_t un_free_fixed (double *pd_fixed, size_t un_parm)**

Description: Counts number of free model parameters in fixed model parameter array.

Arguments:

double *pd_fixed: Pointer to array of size un_parm containing fixed model parameter values. Free parameters are set to preprocessor constant DBL_NAN.
size_t un_parm: Number of model parameters.

Return: Number of free model parameters.

Functions for data structure

h. **int n_data_cmp(const struct data *ps_1, const struct data *ps_2)**

Description: Compares data structure elements by dose and response categories. Used by qsort()

Arguments:

const struct data *ps_1: Pointer to data structure element.
const struct data *ps_2: Pointer to data structure element.

Return: If ps_1 has higher dose category, or same dose but higher response category, then return +1. If dose and response categories are the same, return 0. Otherwise, return -1.

i. **void data_set(int n_loc)**

Description: Extracts n_loc th observation from data structure array, and copies each element into global variables d_dose, c_resp, d_time, and un_sub (and d_sub).

Arguments:

int n_loc: Observation number.

Functions for array summation

j. **double d_sum(const double *pd_a, size_t un_a)**

Description: Sums the first `un_a` elements of array `pd_a`, minimizing floating point errors.

Arguments:

`const double *pd_a:` Pointer to array, length at least `pd_a`.
`size_t un_a:` Number of elements to sum.

Return: The sum of the elements.

k. **`int n_sum_cmp(const double *pd_1, const double *pd_2)`**

Description: Compares array elements for sorting in summation. Used by `qsort()`.

Arguments:

`const double *pd_1:` Pointer to array element.
`const double *pd_2:` Pointer to array element.

Return: If `pd_1` has higher value, then return +1. If values are the same, return 0. Otherwise, return -1.

l. **`double d_sum_rec(double *pd_x, size_t un_x)`**

Description: Sums first two elements in array `pd_x` (sorted from smallest to largest), then resorts. Used recursively by `double d_sum()`.

Arguments:

`double *pd_x:` Pointer to array of length `un_x`.
`size_t un_x:` Number of elements in array `pd_x`.

Return: Sum of the two elements.

m. **`double d_sum_pow(double **ppd_a, size_t un_a, double d_x, int n_start, int n_step)`**

Description: Sums elements of array `*ppd_a` in a power sequence, starting at `n_start` and step size of `n_step`, i.e.,

$$\sum_{\substack{i=0,1,\dots \\ n_{start}+n_{step}\times i \leq n_a}} a[n_{start} + n_{step} \times i] x^{n_{start}+n_{step}\times i}$$

Uses `d_sum()` function to minimize floating point errors.

Arguments:

`const double **ppd_a:` Pointer to array pointer, length at least `un_a`.
`size_t un_a:` Largest index of summed array element.
`double d_x:` Power term.
`int n_start:` Starting index.
`int n_step:` Step size.

Return: Power sum of the elements.

Functions for output of optimization information

n. **`void start_print(void)`**

Description: Prints optimization starting values, starting log-likelihood, and internal scaling values to `stdout`.

o. **void opt_print(void)**

Description: Prints optimization results to stdout.

Other functions

p. **void *p_vec_alloc(size_t un_n, size_t un_val)**

Description: Dynamically allocates memory for un_n elements of size un_val.

Arguments:

size_t un_n: Number of elements.

size_t un_val: Size of elements.

Return: Pointer to start of allocated memory.

q. **void reset_eval(void)**

Description: Reset log-likelihood evaluation flags.

r. **void reset_scale(void)**

Description: Reset optimization internal scaling values (to 1.0).

s. **double d_log(double d_x)**

Description: Safe log function.

Arguments:

double d_x: Logarithm.

Return: If $d_x > 1.0e-50$, then ANSI C function **log(d_x)**; otherwise, **log(1.0e-50)**.

t. **double d_exp(double d_x)**

Description: Safe exponential function.

Arguments:

double d_x: Exponent.

Return: If $d_x < 1.0e+02$, then ANSI C function **exp(d_x)**; otherwise, **exp(1.0e+02)**.

u. **d_pow(double d_x, double d_y)**

Description: Safe power function.

Arguments:

double d_x: Base.

double d_y: Exponent.

Return: If $d_x > 0$, then ANSI C function **pow(d_x, d_y)**; otherwise, 0.

v. **double d_intpow(double d_x, int i_n)**

Description: Safe integer power function.

Arguments:

double d_x: Base.

int i_n: Integer exponent.
Return: ANSI C function `pow(d_x, (double) i_n)`.

w. **double computeProb(double dose, double t, unsigned int modelType)**

Description: function to compute probability of response for the MSW model.

Arguments:

double dose	dose value
double t	time t0
int modelType	Incidental or Fatal tumor model

Return:

double	probability of positive response
--------	----------------------------------

x. **compute_response(unsigned int modelType)**

Description: function to compute estimated number of positive responses at each dose level.

Argument:

int modelType	Incidental or Fatal tumor model
---------------	---------------------------------

3.2.8. Components *gev.h* and *gev.c*

The header file *gev.h* contains declarations for functions stored in *gev.c*.

The *gev.c* file consists of the following:

- Definitions of preprocessor constants,
- Definitions and declaration of structures, and
- Definitions and declarations of arrays and variables

All definitions and declarations above are specific to the multistage Weibull or GEV model.

The *gev.c* file also consists of the following functions:

Function for setting up the model

a. **void gev_fun(void)**

Description: Sets up the global functions structure for the Multistage Weibull model.

Functions required for the global functions structure

b. **void gev_init(void)**

Description: Initializes the multistage Weibull and GEV calculations. Calls the following functions in sequence:

1. `msw_gev_fixed_init()`: Sets up fixed model parameter arrays for multistage Weibull and GEV.
2. `parm_set()`: Sets up model parameter structures. Called multiple times to set up both local and global multistage Weibull and GEV model parameters.
3. `msw_gev_free_conv()`: Converts (free model parameter) optimization starting values from multistage Weibull to GEV.

Also carries out additional initializations, including dynamic allocation to the intermediate calculation structure `ps_gev_calc`, setting `b_t_0_0` the flag to indicate whether multistage Weibull location parameter $t_0 = 0$, and calling `donlp_init()` to initialize the optimization input parameters.

c. **`void gev_clr(void)`**

Description: Converts all multistage GEV parameter structure output, including MLEs and automatic starting values, to multistage Weibull, and frees dynamic memory in intermediate calculation structure `ps_gev_calc`.

d. **`void gev_set(unsigned int b_reset, size_t un_parm)`**

Description: Sets up the multistage GEV model for estimations. Resets all the local model parameter structures and global parameter structure `g_s_parm` (if `b_reset = 1`), and sets the appropriate boundaries for the optimization.

Arguments:

<code>unsigned int b_reset:</code>	Flag for resetting (0 = No reset, 1 = Reset).
<code>size_t un_parm:</code>	Number of model parameters.

e. **`double d_gev_l(void)`**

Description: Calculates the multistage GEV log-likelihood function. *Note*: the parameters in the model are set by the global model parameter structure `g_s_parm`.

Return: Multistage GEV log-likelihood function value.

f. **`double d_gev_lgrad(unsigned int un_p)`**

Description: Calculates the gradient of the multistage GEV log-likelihood function for `un_p`th model parameter. *Note*: the parameters in the model are set by the global model parameter structure `g_s_parm`.

Arguments:

<code>unsigned int un_p:</code>	Index of the model parameter for which the gradient is calculated.
---------------------------------	--

Return: Gradient of multistage GEV log-likelihood function value for `un_p`th model parameter.

g. **double d_gev_nl(unsigned int un_loc)**

Description: Calculates the `un_loc`th non-linear constraint function used for the optimization, including model parameter constraints and, if required, *BMD* defining function constraints. *Note:* the constraints are determined by the global model parameter structure `g_s_parm`, and if required, by the global *BMD* calculation parameter array `g_pd_bmd`.

Arguments:

<code>unsigned int un_loc:</code>	Index of the nonlinear constraint that is calculated.
-----------------------------------	---

Return: `un_loc`th non-linear constraint function value.

h. **double d_gev_nlgrad(unsigned int un_p, unsigned int un_loc)**

Description: Calculates the `un_p`th model parameter gradient for the `un_loc`th non-linear constraint function used for the optimization, including model parameter constraints and, if required, *BMD* defining function constraints. *Note:* the constraints are determined by the global model parameter structure `g_s_parm`, and if required, by the global *BMD* calculation parameter array `g_pd_bmd`.

Arguments:

<code>unsigned int un_loc:</code>	Index of the nonlinear constraint for which the gradient is calculated.
<code>unsigned int c:</code>	Index of the model parameter for which the gradient is calculated.

Return: `un_p`th model parameter gradient for `un_loc`th non-linear constraint function value.

i. **double d_gev_bmd(double d_bmd)**

Description: Calculates the defining function for the benchmark dose (*BMD*). *Note:* the model parameters in the defining function is determined by the global model parameter structure `g_s_parm`, and the *BMD* type is determined by the global MLE structure `g_s_mle`.

Arguments:

<code>double d_bmd</code>	Benchmark dose.
---------------------------	-----------------

Return: *BMD* defining function value at `d_bmd`.

j. **void gev_start(double **ppd_start, size_t un_parm)**

Description: Calculates sets of automatic starting values for the MLEs of the multistage GEV model (free) parameters. *Note:* the number of automatic starting value sets is determined by `g_s_parm_info.s_search.un_start`. Calls the following functions in sequence:

1. `un_free_fixed()`: Counts the number of free model parameters in multistage GEV model with γ and μ fixed.
2. `gev_start_0()`: Fits separate restricted 0-stage GEV model for each dose level at central point of the search grid over γ and μ .

3. `gev_start_reg()`: Fit a linear regression model of the estimated values from the restricted 0-stage GEV models onto a polynomial over the dose levels.
4. `gev_start_grid()`: Using the estimates for the polynomial regression as starting values for the central point of the search grid, fit restricted multistage GEV models at all the grid points over γ and μ . Only returns grid points and associated parameter estimates with the highest maximized log-likelihood.

Note: minor set-up function calls have been omitted from the above sequence for clarity.

Arguments:

<code>double **ppd_start</code>	Pointer to array pointers (i.e., 2-D matrix), containing returned sets of automatic starting values.
<code>size_t un_parm</code>	Number of model parameters.

k. **`void gev_scale(void)`**

Description: Automatically sets the internal scaling for the optimizations.

l. **`void gev_refine(double *pd_fixed, double *pd_free, unsigned int b_store)`**

Description: Sets up therefinement of the multistage GEV model parameter MLEs. Converts, fixed and free multistage GEV model parameters to multistage Weibull. Checks closeness of the multistage Weibull model parameters to their boundaries, and fixes those that are close at the boundaries. Converts refined fixed and free model multistage Weibull model parameters back to multistage GEV model parameters.

Arguments:

<code>double *pd_fixed</code>	Pointer to array of returned (multistage GEV) fixed model parameters.
<code>double *pd_free</code>	Pointer to array of free (multistage GEV) model parameters.
<code>int b_store</code>	Flag for updating MLE parameter structure (<code>g_s_mle.s_parm</code>) with refined fixed parameters.

m. **`int n_msw_inv_cov(double **ppd_inv_cov, double *pd_fixed)`**

Description: Calculates the inverse asymptotic covariance of the multistage Weibull parameter MLEs (i.e., -0.5 H Hessian of the log-likelihood).

Arguments:

<code>double **ppd_inv_cov:</code>	Pointer to array pointer (i.e., 2-D matrix) returning the inverse asymptotic covariance.
------------------------------------	--

Return: Gradient of multistage GEV log-likelihood function value for un_pth model parameter.

Description: Resets starting values (that may have been changed, e.g., by adjustments made in the refined MLE optimization) for *BMD* calculations.

double *pd_start:	Array pointer returning multistage GEV free model parameters.
-------------------	--

Description: Converts local multistage Weibull fixed model parameter array to local multistage Weibull fixed model parameter array.

Description: Converts free parameters in multistage Weibull model parameter structure to free parameters in multistage GEV model parameter structure.

```

struct parm *ps_gev_parm:Array pointer to multistage GEV model
                parameters (modified by function).
struct parm *ps_msw_parm:Array pointer to multistage Weibull
                model parameters.

```

Description: Converts free parameters in multistage GEV model parameter structure to free parameters in multistage Weibull model parameter structure.

```

struct parm *ps_msw_parm:Array pointer to multistage Weibull
                                model parameters (modified by
                                function).
struct parm *ps_gev_parm:Array pointer to multistage GEV model
                                parameters.

```

Description: Converts **ps_parm** model parameter structure from multistage GEV to multistage Weibull.

Arguments:

<code>struct parm *ps_parm:</code>	Structure pointer to multistage GEV model parameters (modified by function).
<code>double *ps_fixed:</code>	Array pointer to fixed multistage Weibull model parameters.

s. **void msw_gev_fixed_init(void)**

Description: Initializes the local multistage Weibull and multistage GEV fixed parameter arrays.

Support functions for calculation of Multistage GEV log-likelihoods

t. **void gev_calc_set(int n_loc)**

Description: Sets pointers to addresses in multistage GEV log-likelihood intermediate calculation structure array `ps_gev_calc`.

Arguments:

<code>int n_loc:</code>	Index number for structure array <code>ps_gev_calc</code> .
-------------------------	---

u. **unsigned int un_gev_updt(const double d_free[], unsigned int *pb_first)**

Description: Determines update requirements for multistage GEV log-likelihood intermediate calculations, based on the free parameter array `d_free`.

Arguments:

<code>const double d_free[]:</code>	Array of multistage GEV free parameter values.
<code>unsigned int *pb_first:</code>	Pointer to flag (0 = Do not reset all calculations, 1 = First calculation of log-likelihood, i.e., reset every calculation).

Return: Binary valued flag of update requirements, which is the sum of 0 and any of the following constants.

<code>GAM_FLAG = 1:</code>	Require update of γ parameter value components.
<code>MU_FLAG = 2:</code>	Require update of μ parameter value components.
<code>POLY_FLAG = 4:</code>	Require update of polynomial (i.e., (b_i) components).

v. **void gev_updt(void)**

Description: Updates multistage GEV log-likelihood intermediate calculations, depending on the flag returned by `un_gev_updt ()`.

Support functions for automatic starting values

w. **double d_gam(int i)**

Description: Calculates γ search grid value.

Arguments:

int i: Search grid index.

Return: Value of $\gamma[i]$ search grid point.

x. **double d_mu(int i, int j)**

Description: Calculates μ search grid value.

Arguments:

int i: First search grid index.

int j: Second search grid index.

Return: Value of $\mu[i, j]$ search grid point.

y. **void gev_start_0(double *pd_y)**

Description: Fits separate 0-stage GEV models restricted to the search grid center $[0, 0]$ for each dosage level.

Arguments:

double *pd_y: Array pointer to values of the estimated b_0 parameter for each dosage level (returned).

z. **void gev_start_reg(double *pd_start, double *pd_y, size_t un_parm)**

Description: Carry out a linear regression of the b_0 parameter estimates from the restricted 0-stage GEV models on polynomial of dosage levels.

Arguments:

double *pd_start: Estimated polynomial regression parameters.

double *pd_y: Array pointer to values of the estimated b_0 parameter for each dosage level.

size_t un_parm: Number of model parameters.

aa. **void gev_start_grid(double **ppd_start, size_t un_parm, double *pd_start)**

Description: Fits restricted full-stage GEV model at each point on the search grid. Selects estimates from grid points with the highest log-likelihood as potential starting-values.

Arguments:

double **ppd_start: Pointer to array pointer (i.e., 2-D matrix) of returned sets of automatic starting values.

size_t un_parm: Number of parameters in the model.

double *pd_start:	Array pointer to starting values for the central gridpoint [0, 0].
-------------------	--

bb. void gev_start_full(double **ppd_start, double *pd_l, int *pn_gam, int *pn_mu, size_t un_parm, int i, int j, unsigned int b_cold)

Description: Fits restricted full-stage GEV model at grid point [i, j].

Arguments:

double **ppd_start:	Pointer to array pointer (i.e., 2-D matrix) of returned sets of automatic starting values.
double *pd_l:	Array pointer to stored values of the maximized log-likelihood (modified).
int *pn_gam:	Array pointer to stored index of the γ grid point, i.e., index i (modified).
int *pn_mu:	Array pointer to stored index of the μ grid point, i.e., index j (modified).
size_t un_parm:	Number of parameters in the model.
int i:	First grid point index.
int j:	Second grid point index.
unsigned int b_cold:	Flag for “cold” restart of donlp optimization.

cc. void gev_start_set(double d_gam, double d_mu, double *pd_b, size_t un_b)

Description: Sets the (local) fixed parameter arrays for restricted multistage GEV and Weibull models.

Arguments:

double d_gam:	Value of the fixed γ value.
double d_mu:	Value of the fixed μ value.
double *pb_b:	Array pointer to the fixed b_i values.
size_t un_b:	Size of the array pb_b.

dd. void as274c(int np, int nob, double y[], double **X, double beta[])

Description: Interfaces to the linear regression module as274.

Arguments:

int np:	Number of parameters.
int nob:	Number of observations.
double y[]:	Array of response values, size nob.
double **X:	Design matrix, size nob by np.
double beta[]:	Least squares parameter estimates (returned), size np.

Support functions for asymptotic standard error

ee. **double d_msw_lhess(unsigned int un_p1, unsigned int un_p2)**

Description: Calculates the Hessian of the multistage Weibull log-likelihood.

Arguments:

unsigned int un_p1: First parameter index.

unsigned int un_p2: Second parameter index.

Return: Value of the Hessian for the parameters with indices un_p1 and un_p2.

Support functions for gradients of non-linear constraints

ff. **double d_gev_bmdgrad(double d_bmd, unsigned int un_p)**

Description: Calculates the gradient of the *BMD* defining function.

Arguments:

double d_bmd: Value of the benchmark dose.

unsigned int un_p: Parameter index.

Return: Value of the gradient for the parameter with index un_p.

3.2.9. Components donlpfun.h and donlpfun.c

The header file donlpfun.h contains declarations for functions stored in donlpfun.c that were not declared by the donlp header file o8para.h.

The donlpfun.c file consists of the following functions:

a. **void scale(double *pd_vec, int n_p, unsigned int b_dir)**

Description: Carries out internal scaling and unscaling for the optimization.

Arguments:

double *pd_vec: Pointer to vector of n_p values to be scaled / unscaled (modified).

int n_p: Number of parameter values.

unsigned int b_dir: Flag for scaling direction (0 = unscale, 1 = scale).

b. **void donlp_init(double d_tau0, double d_del0, int n_nreset, int n_nsilent, int n_cold, double d_big)**

Description: Sets the optimization parameters in the global optimization parameter structure gs_opt.

Arguments:

double d_tau0: Tuning parameter limiting level of boundary violations by (non-bound) constraints during optimization.

double d_del0: Tuning parameter determining the initial binding of (non-bound) constraints.

<code>int n_reset:</code>	Maximum number of resets.
<code>int n_silent:</code>	Flag for diagnostic output.
<code>int n_cold:</code>	Flag for “cold” start.
<code>double d_big:</code>	Value of the largest floating point value allowed in the optimization search.

The `donl_pfun.c` file also consists of functions required by `donlp`. Please read the `donlp` user document for more details on these functions. The functions include:

- c. **`void user_init_size(void)`**
Description: Sets the dimensions and other sizes of the optimization.
- d. **`void setup(void)`**
Description: Currently inactive.
- e. **`void solchk(void)`**
Description: Sets the number and prints the message of the `donlp` diagnostics. Sets and unscales the solution.
- f. **`void user_init(void)`**
Description: Sets the optimization parameters. Sets the model parameter search boundary values.
- g. **`void ef(DOUBLE x[], DOUBLE *fx)`**
Description: Calculates the objective function, i.e., $-\log$ -likelihood.
- h. **`void egradf(DOUBLE x[], DOUBLE gradf[])`**
Description: Calculates the gradient of the objective function, i.e. gradient of $-\log$ -likelihood.
- i. **`void econ(INTEGER type, INTEGER liste[], DOUBLE x[], DOUBLE con[], LOGICAL err[])`**
Description: Calculates the non-linear constraint functions.
- j. **`void econgrad(INTEGER liste[], INTEGER shift, DOUBLE x[], DOUBLE **grad)`**
Description: Calculates the gradients of the non-linear constraint functions.
- k. **`void eval_extern(INTEGER mode)`**
Description: Currently inactive.