# The Numerit-GSL Interface Package

## User's Guide



**Introduction**

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free under the GNU General Public License. The library provides a wide range of mathematical routines and includes over 1000 functions in total.
The Numerit-GSL Interface Package includes a Numerit module "gsl-interface.num" that allows you to use the Gnu Scientific Library in Numerit right away (provided you have obtained the GSL Dynamic Link Library (DLL); see below "**Getting the GSL library**"). The module includes more than 1400 Numerit declarations of GSL functions and more than 100 Numerit declarations of GSL variables that give you instant access to GSL from Numerit. The declarations are divided into categories according to the GSL Reference Manual. These categories include:

- Polynomial evaluation and roots
- Special functions
- Linear Algebra
- Eigensystems
- Fast Fourier Transforms
- Quadratures
- Random Numbers
- Quasi-Random Sequences
- Statistics
- Histograms
- Monte-Carlo Integration
- Simulated Annealing
- Differential Equations
- Interpolation
- Numerical Differentiation
- Chebyshev Approximation
- Discrete Hankel Transforms
- Root Finding
- Minimization
- Least-Squares Fitting
- Discrete Wavelet Transforms
- And more...

In addition, the package contains a collection of Numerit commented example programs that cover all the topics in the GSL Reference Manual. These programs are Numerit implementations of the C examples that are found in the reference manual. Besides demonstrating the functions that are described in the manual, the programs can be used as starting points to your own programs.
The package also contains a collection of Dynamic Link Libraries (DLLs) with utilities that allow you to access GSL data objects that are not supported by Numerit. For example: copy GSL vectors and matrices from/to Numerit vectors and matrices, access the GSL integration workspace structure, copy null-terminated strings to Numerit strings, etc. Besides the usefulness of the utilities themselves, they can be used as examples to creating your own DLLs, as they are delivered with full source code and instructions (below) how to compile them to DLLs using the Borland free C++ compiler.
Another useful DLL in the package provides a simple but powerful mechanism to allow handling of GSL errors in Numerit. With this mechanism any error that is issued by a GSL function invokes Numerit's error handler (see "Error handling").

## The Numerit-GSL Interface Package contents

The Numerit-GSL Interface Package - User's Guide.pdf (this file)
gsl-interface.num – the Numerit-GSL interface module
gsl-example-xxx.num – Numerit example programs (35 files)
num_dllutil_xxx.dll – DLLs with various utilities (7 files)
num_dllutil_xxx.cpp – source code of the above utilities (7 files)
num_dllutil_xxx.rsp – response files for compiling the above utilities (7 files)
num_tsp.dll – A DLL for the Simulated Annealing example program
num_tsp.cpp – source code of num_tsp.dll
num_tsp.rsp – response file for compiling num_tsp.cpp
license.txt - The Numerit-GSL Interface Package License Agreement

Note: The source code files and response files are in the folder "Numerit-GSL utilities source".

## Installation

1. Extract the Numerit-GSL Interface Package files into any directory of your choice.
2. Copy the GSL library files (see below): libgsl.dll and libgslcblas.dll to the same directory.

## Getting the GSL library

The GSL library can be downloaded from SourceForge.net:
http://gnuwin32.sourceforge.net/packages/gsl.htm.
The DLLs are found in the **Binaries** component:
http://gnuwin32.sourceforge.net/downlinks/gsl-bin-zip.php.
The manual is in the **Documentation** component:
http://gnuwin32.sourceforge.net/downlinks/gsl-doc-zip.php.

These components come as zip files so you need to open the zip files and extract the files that you need. From the Binaries file (gsl-1.8-bin.zip) you need to extract two DLLs - libgsl.dll and libgslcblas.dll.
From the Documentation file (gsl-1.8-doc.zip) you should extract the Reference Manual either as a PDF file (gsl-ref.pdf), or as a Help file (gsl-ref.chm or gsl-ref.hlp), or as a HTML file (gsl-ref.html).

## Using the Numerit-GSL interface module

The Numerit-GSL interface module is intended to be used with Numerit Pro (version 1.7.106 and higher). However, you can still use it even if you are a user of the standard edition (see below).
**If you are a Numerit Pro user**: after installation just open any of the example programs and test it. Build your own programs similar to the examples. The simplest way is to start with an example and modify it according to your needs. Note that if you start your program from scratch you need to open the "Modules" window and add the module "gsl-interface.num".
**If you are a Numerit (standard edition) user**: Open the module as a normal Numerit program and copy its content to the beginning of your program. Note that you need to do so for each of your programs that call functions from GSL. You can, of course, copy only those declarations that are relevant to your application.

**Note**: each program should start with a call to gsl.main() that initializes the module, and should end with a call to TerminateGSL() that restores GSL's error handler and frees all DLLs (see the examples).

## The utilities

The Numerit built-in DLL interface converts C variables (including arrays) to Numerit variables and back. However, there are some GSL objects that cannot be converted in a simple way since they have no equivalents in Numerit. That's why we have created a set of utilities (in DLLs) that allow you to access these objects from Numerit. The function declarations for all these utilities are included in the Numerit-GSL interface module.

These utilities are:

String utilities (in num_dllutil_str.dll)
**num_str_len**: returns the length of a null-terminated string
**num_str_char**: returns the ASCII value of the i'th character of a null-terminated string
**num_str_copy**: copies a null-terminated string to a Numerit string
**num_strn_copy**: copies n characters from a null-terminated string to a Numerit string

Vector/Matrix utilities (in num_dllutil_vecmat.dll)
**num_vec_copy_g2n**: copies a 'double' vector from GSL to Numerit
**num_vec_int_copy_g2n**: copies a 'int' vector from GSL to Numerit
**num_vec_complex_copy_g2n**: copies a 'complex' vector from GSL to Numerit
**num_vec_copy_n2g**: copies a 'double' vector from Numerit to GSL
**num_vec_int_copy_n2g**: copies a 'int' vector from Numerit to GSL
**num_vec_complex_copy_n2g**: copies a 'complex' vector from Numerit to GSL
**num_mat_copy_g2n**: copies a 'double' matrix from GSL to Numerit
**num_mat_int_copy_g2n**: copies a 'int' matrix from GSL to Numerit
**num_mat_complex_copy_g2n**: copies a 'complex' matrix from GSL to Numerit
**num_mat_copy_n2g**: copies a 'double' matrix from Numerit to GSL
**num_mat_int_copy_n2g**: copies a 'int' matrix from Numerit to GSL
**num_mat_complex_copy_n2g**: copies a 'complex' matrix from Numerit to GSL

Integration utilities (in num_dllutil_integration.dll)
Get members of gsl_integration_workspace:
**num_integration_limit**: returns the value of 'limit'
**num_integration_size**: returns the value of 'size'

Monte-Carlo Integration utilities (in num_dllutil_monte.dll)
Get members of gsl_monte_miser_state:
**num_monte_miser_estimate_frac**: returns the value of 'estimate_frac'
**num_monte_miser_min_calls**: returns the value of 'min_calls'
**num_monte_miser_min_calls_per_bisection**: returns the value of 'min_calls_per_bisection'
**num_monte_miser_alpha**: returns the value of 'alpha'
**num_monte_miser_dither**: returns the value of 'dither'
Set members of gsl_monte_miser_state:
**num_monte_miser_estimate_frac_set**: sets the value of 'estimate_frac'
**num_monte_miser_min_calls_set**: sets the value of 'min_calls'
**num_monte_miser_min_calls_per_bisection_set**: sets the value of 'min_calls_per_bisection'

**num_monte_miser_alpha_set**: sets the value of 'alpha'
**num_monte_miser_dither_set**: sets the value of 'dither'

Get members of gsl_monte_vegas_state:
**num_monte_vegas_result**: returns the value of 'result'
**num_monte_vegas_sigma**: returns the value of 'sigma'
**num_monte_vegas_chisq**: returns the value of 'chisq'
**num_monte_vegas_alpha**: returns the value of 'alpha'
**num_monte_vegas_iterations**: returns the value of 'iterations'
**num_monte_vegas_stage**: returns the value of 'stage'
**num_monte_vegas_mode**: returns the value of 'mode'

Set members of gsl_monte_vegas_state:
**num_monte_vegas_result_set**: sets the value of 'result'
**num_monte_vegas_sigma_set**: sets the value of 'sigma'
**num_monte_vegas_chisq_set**: sets the value of 'chisq'
**num_monte_vegas_alpha_set**: sets the value of 'alpha'
**num_monte_vegas_iterations_set**: sets the value of 'iterations'
**num_monte_vegas_stage_set**: sets the value of 'stage'
**num_monte_vegas_mode_set**: sets the value of 'mode'

Series Acceleration utilities (in num_dllutil_accel.dll)

Get members of gsl_sum_levin_u_workspace:
**num_accel_terms_used**: returns the value of 'terms_used'
**num_accel_sum_plain**: returns the value of 'sum_plain'

Multifit utilities (in num_dllutil_multifit.dll)

Get members of gsl_multifit_fsolver:
**ptr num_multifit_fsolver_x**: returns 'x' (pointer to gsl_vector)
**ptr num_multifit_fsolver_f**: returns 'f' (pointer to gsl_vector)
**ptr num_multifit_fsolver_dx**: returns 'dx' (pointer to gsl_vector)

Get members of gsl_multifit_fdfsolver:
**ptr num_multifit_fdfsolver_x**: returns 'x' (pointer to gsl_vector)
**ptr num_multifit_fdfsolver_f**: returns 'f' (pointer to gsl_vector)
**ptr num_multifit_fdfsolver_dx**: returns 'dx' (pointer to gsl_vector)
**ptr num_multifit_fdfsolver_J**: returns 'J' (pointer to gsl_matrix)

**Error handling**

You can send all GSL errors to Numerit's internal error handler (recommended). To enable this you need to set the flag 'EnableNumeritErrorHandler' in the Numerit-GSL interface module to TRUE (currently set as the default value). To disable error handling set this flag to FALSE.

When the flag is set to TRUE, the GSL error handling is redirected to the function 'num_dll_error' that is found in 'num_dllutil_err.dll'. This redirection is done by the Numerit-GSL interface module itself, when it is initialized, by calling the GSL function 'gsl_set_error_handler'. In this mode, whenever GSL calls the error handler it actually calls 'num_dll_error' which sends the message to Numerit. Note that GSL's default error handler should never be left in effect since it aborts Numerit when called, so setting the flag to FALSE actually disables GSL error handling (the interface module calls the GSL function 'gsl_set_error_handler_off').

If you don't enable error handling and the GSL function you call returns an error code, you can call the function 'ShowGSLErr' (defined in the Numerit-GSL interface module) to report the error.

Note that calling the Numerit function TerminateGSL() at the end of your program also calls the function restore_gsl_error_handler() which restores the original GSL error handler.

**The examples**

We have made an effort to cover most of the GSL function categories with examples. In most cases we followed the examples that appear in the GSL reference manual.

The example programs are:

| Function Category | Numerit program |
|---|---|
| Elementary Functions | gsl-example-elementary.num |
| Small Integer Powers | gsl-example-small powers.num |
| Complex Numbers | gsl-example-complex.num |
| Polynomials | gsl-example-polynomial.num |
| Special Functions | gsl-example-special functions.num |
| Vectors | gsl-example-vector.num |
| Matrices | gsl-example-matrix.num |
| Permutations | gsl-example-permutations.num |
| Combinations | gsl-example-combinations.num |
| Sorting | gsl-example-sorting.num |
| Linear Algebra | gsl-example-linear algebra.num |
| Eigensystems | gsl-example-eigensys.num |
| Fast Fourier Transforms | gsl-example-fft.num |
| Numerical Integration | gsl-example-integration.num |
| Random Number Generation | gsl-example-rng.num |
| Quasi-Random Sequences | gsl-example-qrng.num |
| Random Number Distributions | gsl-example-ran.num |
| Statistics | gsl-example-stats.num |
| Histograms | gsl-example-histogram.num |
| Monte-Carlo Integration | gsl-example-monte.num |
| Simulated Annealing - one dimensional | gsl-example-siman-1d.num |
| Simulated Annealing - traveling salesman | gsl-example-siman-tsp.num |
| Ordinary Differential Equations | gsl-example-ode.num |
| Interpolation | gsl-example-interpolation.num |
| Numerical Differentiation | gsl-example-diff.num |
| Chebyshev Approximations | gsl-example-cheb.num |
| Series Acceleration | gsl-example-accel.num |
| Wavelet Transforms | gsl-example-wavelet.num |
| Discrete Hankel Transforms | gsl-example-dht.num |
| One dimensional Root-Finding | gsl-example-root.num |
| One dimensional Minimization | gsl-example-min.num |
| Multidimensional Root-Finding | gsl-example-multiroot.num |
| Multidimensional Minimization | gsl-example-multimin.num |
| Least-Squares Fitting | gsl-example-fit.num |
| Nonlinear Least-Squares Fitting | gsl-example-nlfit.num |

## Creating your own Dynamic Link Library (DLL) using the free Borland C++ compiler

*The instructions in this section are provided as a service to the users of the Numerit-GSL package. KEDMI Scientific Computing does not give any guarantee regarding the procedure described here and does not provide support for it.*

*Note: this section assumes that you have basic knowledge in C programming.*

### Borland compiler

In order to create a DLL you first need to download the free Borland C++ compiler. This involves the following steps:

1. Go to the Download page.
2. Click on the **C++ Compiler 5.5** link.
3. Download the compiler (a 8.5 Mb installer file).
4. Run the installer.

### GSL header

If you want to refer to GSL objects in your DLL you'll also need the GSL header files that are found in the **Developer files** component of the GSL package. This can be downloaded from: http://gnuwin32.sourceforge.net/downlinks/gsl-lib-zip.php.

### Building a DLL

To build a DLL you need to create two files: 1. A C source file that contains the functions you want to call from another application (e.g. from Numerit). 2. A response file that instructs the compiler to create the DLL.

For example, lets see one of the DLLs that are included in the package: num_tsp.dll (used in the Simulated Annealing example program). The C source file is "num_tsp.cpp" which contains four functions. It is written in C++ but does not use any object oriented features of C++ so it could easily be written in plain C.

The instruction **extern "C"** at the top of the file "num_tsp.cpp" makes sure that the functions are linked according to the C linkage conventions.

Each function that needs to be called from Numerit (or from any other application) should be exported. This is done by the declaration __declspec(dllexport) that precedes each such function.

The response file for creating the DLL is "num_tsp.rsp". It includes the following instructions to the compiler:

```
-WD
-ff
-O2
-Ic:\bc55\include
-Lc:\bc55\lib
-enum_tsp
num_tsp.cpp
```

The first line tells the compiler to create a DLL. The second line sets the compiler's "floating point" flag. The third line sets the optimization flag. The fourth line specifies the path to the compiler's "include" directory (here we assumed that the Borland compiler was installed in the directory c:\bc55). The fifth line specifies the path to the compiler's "lib" directory. The sixth line specifies the DLL name. The seventh line specifies the name of the source file.

To actually generate the DLL you need to open a command-line window (command prompt) and set the path to the compiler's "bin" directory (e.g., path c:\bc55\bin). Make sure you are in the same directory where the C program and the response file reside and type:
bcc32 @num_tsp.rsp
at the command line, press Enter and you are done (you should, of course, replace "num_tsp" by the name of *your* dynamic link library). The DLL will be created in the same directory. In order to use it with Numerit you need to declare it in your Numerit program (see the Numerit help topic for the instruction 'dll').

---