

GSL

The GNU Scientific Library. Provides a wide range of mathematical routines.

Introduction

The GNU Scientific Library (GSL) provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total. The routines have been written from scratch in C, and present a modern Applications Programming Interface (API) for C programmers, allowing wrappers to be written for very high level languages.

The library covers a wide range of topics in numerical computing. Routines are available for the following areas:

Complex Numbers	Roots of Polynomials
Special Functions	Vectors and Matrices
Permutations	Combinations
Sorting	BLAS Support
Linear Algebra	CBLAS Library
Fast Fourier	Eigensystems
Transforms	
Random Numbers	Quadrature
Random Distributions	Quasi-Random
	Sequences
Histograms	Statistics
Monte Carlo	N-Tuples
Integration	
Differential Equations	Simulated Annealing
Numerical	Interpolation
Differentiation	
Series Acceleration	Chebyshev
	Approximations
Root-Finding	Discrete Hankel
	Transforms
Least-Squares Fitting	Minimization
IEEE Floating-Point	Physical Constants
Basis Splines	Wavelets

Modules

The GSL 1.16 is available on Anselm, compiled for GNU and Intel compiler. These variants are available via modules:

Module	Compiler
gsl/1.16-gcc	gcc 4.8.6
gsl/1.16-icc(default)	icc

```
$ module load gsl
```

The module sets up environment variables, required for linking and running GSL enabled applications. This particular command loads the default module, which is gsl/1.16-icc

Linking

Load an appropriate gsl module. Link using **-lgsl** switch to link your code against GSL. The GSL depends on cblas API to BLAS library, which must be supplied for linking. The BLAS may be provided, for example from the MKL library, as well as from the BLAS GSL library (-lgslcblas). Using the MKL is recommended.

Compiling and linking with Intel compilers

```
$ module load intel
$ module load gsl
$ icc myprog.c -o myprog.x -Wl,-rpath=$LIBRARY_PATH -mkl -lgsl
```

Compiling and linking with GNU compilers

```
$ module load gcc
$ module load mkl
$ module load gsl/1.16-gcc
$ gcc myprog.c -o myprog.x -Wl,-rpath=$LIBRARY_PATH -lmkl_intel_lp64 -lmkl_gnu_thread -lmkl_core -lgsl
```

Example

Following is an example of discrete wavelet transform implemented by GSL:

```
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_sort.h>
```

```

#include <gsl/gsl_wavelet.h>

int
main (int argc, char **argv)
{
    int i, n = 256, nc = 20;
    double *data = malloc (n * sizeof (double));
    double *abscoeff = malloc (n * sizeof (double));
    size_t *p = malloc (n * sizeof (size_t));

    gsl_wavelet *w;
    gsl_wavelet_workspace *work;

    w = gsl_wavelet_alloc (gsl_wavelet_daubechies, 4);
    work = gsl_wavelet_workspace_alloc (n);

    for (i=0; i<n; i++)
        data[i] = sin (3.141592654*(double)i/256.0);

    gsl_wavelet_transform_forward (w, data, 1, n, work);

    for (i = 0; i < n; i++)
    {
        abscoeff[i] = fabs (data[i]);
    }

    gsl_sort_index (p, abscoeff, 1, n);

    for (i = 0; (i + nc) < n; i++)
        data[p[i]] = 0;

    gsl_wavelet_transform_inverse (w, data, 1, n, work);

    for (i = 0; i < n; i++)
    {
        printf ("%gn", data[i]);
    }

    gsl_wavelet_free (w);
    gsl_wavelet_workspace_free (work);

    free (data);
    free (abscoeff);
    free (p);
    return 0;
}

```

Load modules and compile:

```
$ module load intel gsl  
icc dwt.c -o dwt.x -Wl,-rpath=$LIBRARY_PATH -mkl -lgsl
```

In this example, we compile the `dwt.c` code using the Intel compiler and link it to the MKL and GSL library, note the `-mkl` and `-lgsl` options. The library search path is compiled in, so that no modules are necessary to run the code.