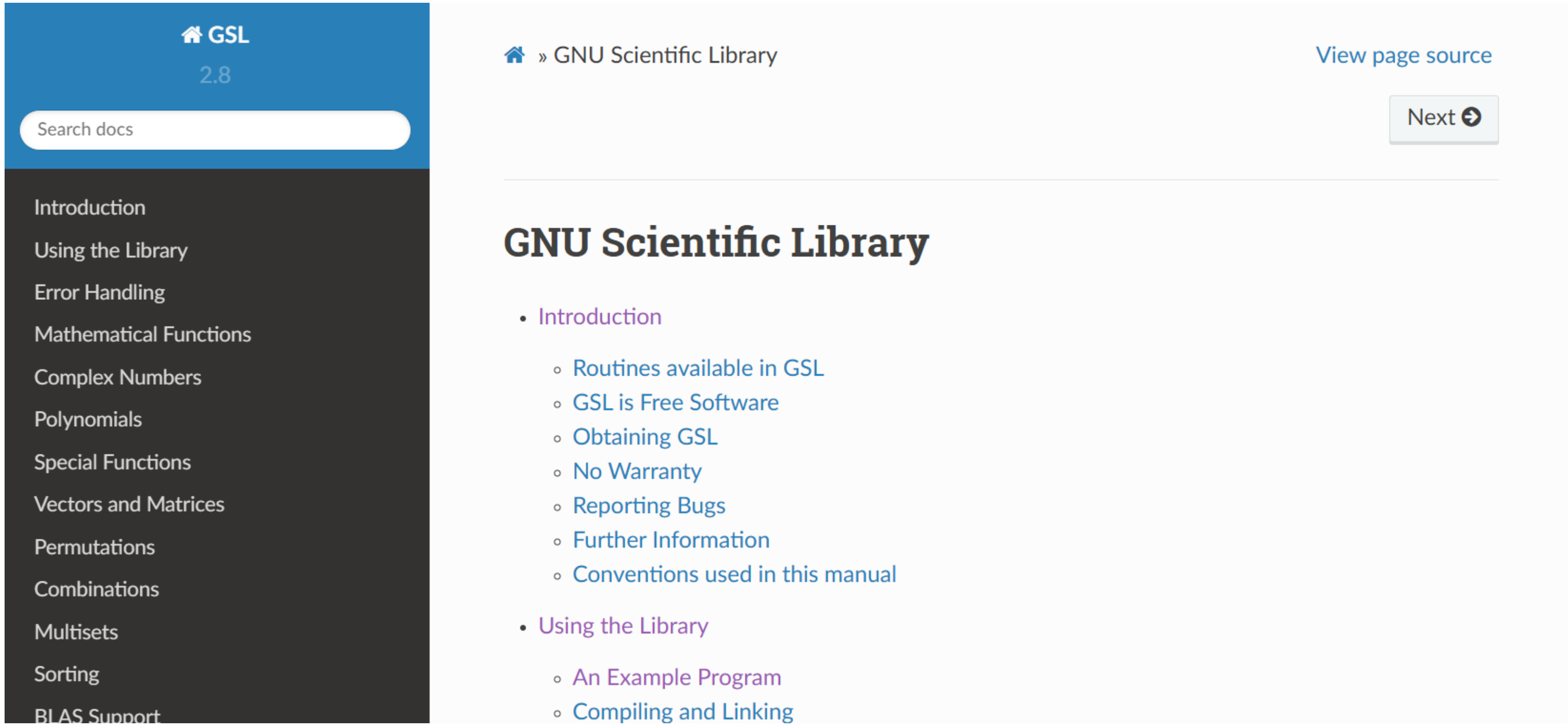
GSL C library mini guide

source

- GNU Scientific Library — GSL 2.8 documentation



Installation

```
qbaliu@Noovera:~$ sudo apt install libgsl-dev
```

```
qbaliu@Noovera:~$ gsl-config --version
2.7.1
qbaliu@Noovera:~$
```

Installation

- after installation a gsl/ parent directory is created, its standard location is

```
qbaliu@Noovera:/usr/include$ pwd
/usr/include
qbaliu@Noovera:/usr/include$
```

Installation

- this parent directory includes all the header files for the specific mathematical functions available in the GSL library

```
qbaliu@Noovera:/usr/include$ cd gsl/
qbaliu@Noovera:/usr/include/gsl$ pwd
/usr/include/gsl
qbaliu@Noovera:/usr/include/gsl$ ls -lh | head -n 5
total 1.8M
-rw-r--r-- 1 root root  22K Dec  6  2021 gsl_blas.h
-rw-r--r-- 1 root root 1.6K Dec  6  2021 gsl_blas_types.h
-rw-r--r-- 1 root root  580 Dec  6  2021 gsl_block.h
-rw-r--r-- 1 root root 2.3K Dec  6  2021 gsl_block_char.h
qbaliu@Noovera:/usr/include/gsl$
```

Running an example

- example programme which uses the GSL library
- note the header file, we must specify the specific header file, which is in the gsl/ directory
- the programme itself can be in any directory (the #include statement will by default search the /usr/include/ directory)

```
#include <stdio.h>
#include <gsl/gsl_integration.h>

// Define the function to be integrated (e.g., sin(x))
double my_function(double x, void *params) {
    return sin(x);
}

int main() {
    // Declare the integration workspace
    gsl_integration_workspace *workspace = gsl_integration_workspace_alloc(1000);

    // Set up the integration function
    gsl_function F;
    F.function = &my_function;
    F.params = NULL;
```

Running an example

- during compilation, we need to link (-l) the following libraries: gsl, cBLAS (gslcblas), math (m)

```
qbaliu@Noovera:~$ gcc gsl_test.c -o gsl_integration_example -lgsl -lgslcblas -lm
```

```
qbaliu@Noovera:~$ ./gsl_integration_example
```

Basic statistics part I

- example function notice the input and output types

double gsl_stats_mean(const double data[], size_t stride, size_t n)

This function returns the arithmetic mean of `data`, a dataset of length `n` with stride `stride`. The arithmetic mean, or *sample mean*, is denoted by $\hat{\mu}$ and defined as,

$$\hat{\mu} = \frac{1}{N} \sum x_i$$

where x_i are the elements of the dataset `data`. For samples drawn from a gaussian distribution the variance of $\hat{\mu}$ is σ^2/N.

BASIC_STATISTICS/basic_stat_demo1.c/.exe

Basic statistics part I

```
#include<stdio.h>
#include<gsl/gsl_statistics.h>   // load the 'gsl_statistics' header file

#define SIZE 4

int main(){

        double data[SIZE] = {11.00, 12.3, -9.10, 4.00};  // define data array of type double
        double mean, mean_stride2, variance, min, max;     // these variables will hold the output of the function, which return doub
les
        int stride = 1;    // stride parameter
        int bigger_stride = 2;

        mean     = gsl_stats_mean(data, stride, SIZE);
        variance = gsl_stats_variance(data, stride, SIZE);
        max  = gsl_stats_max(data, stride, SIZE);
        min = gsl_stats_min(data, stride, SIZE);

        mean_stride2 = gsl_stats_mean(data, bigger_stride, SIZE);


        printf("mean with stride 1: %g\n", mean);
        printf("mean with stride 2: %g\n", mean_stride2);
        printf("variance: %g\n", variance);
        printf("max: %g\n", max);
        printf("min: %g\n", min);


        return(0);
}
~
~
```

Basic statistics part I

```
gcc basic_stat_demo1.c -o basic_stats_demo1.exe -lgsl -lgslcblas -lm
```

```
qbaliu@Noovera:/mnt/c/Users/Qba Liu/Documents/NAUKA_WLASNA/CODING_GENERAL/GSL_examples/BASIC_STATISTICS$ ./basic_stats_demo1.exe
mean with stride 1: 4.55
mean with stride 2: 0.475
variance: 96.0967
max: 12.3
min: -9.1
qbaliu@Noovera:/mnt/c/Users/Qba Liu/Documents/NAUKA_WLASNA/CODING_GENERAL/GSL_examples/BASIC_STATISTICS$
```

Basic statistics part I

- further functions (usage is analog):
- Statistics — GSL 2.8 documentation

Basic statistics part II

Autocorrelation

double gsl_stats_lag1_autocorrelation(const double data[], const size_t stride, const size_t n)

This function computes the lag-1 autocorrelation of the dataset `data`.

$$a_1 = \frac{\sum_{i=2}^{n}(x_i - \hat{\mu})(x_{i-1} - \hat{\mu})}{\sum_{i=1}^{n}(x_i - \hat{\mu})(x_i - \hat{\mu})}$$

Covariance

double gsl_stats_covariance(const double data1[], const size_t stride1, const double data2[], const size_t stride2, const size_t n)

This function computes the covariance of the datasets `data1` and `data2` which must both be of the same length `n`.

$$covar = \frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \hat{x})(y_i - \hat{y})$$

Basic statistics part II

Correlation

double gsl_stats_correlation(const double data1[], const size_t stride1, const double data2[], const size_t stride2, const size_t n)

This function efficiently computes the Pearson correlation coefficient between the datasets `data1` and `data2` which must both be of the same length `n`.

$$r = \frac{cov(x,y)}{\hat{\sigma}_x \hat{\sigma}_y} = \frac{\frac{1}{n-1}\sum(x_i - \hat{x})(y_i - \hat{y})}{\sqrt{\frac{1}{n-1}\sum(x_i - \hat{x})^2}\sqrt{\frac{1}{n-1}\sum(y_i - \hat{y})^2}}$$

Basic statistics part II

BASIC_STATISTICS/basic_stat_demo2.c/.exe

```
#include<stdio.h>
#include<gsl/gsl_statistics.h>

#define SIZE 20

int main(){

    double sample1[SIZE] = {4.236, 7.895, 2.467, 9.032, 5.364, 1.907, 8.123, 6.548, 0.483, 3.792, 7.024, 4.691
7.219, 0.837, 3.908, 6.255, 1.638};
    double sample2[SIZE] = {6.483, 2.917, 8.256, 4.732, 1.398, 7.031, 9.128, 0.573, 3.684, 5.439, 2.741, 8.395
7.567, 1.274, 9.342, 3.018, 5.768};
    double lag1_autocorr, correlation, covariance;
    int stride = 1;

    lag1_autocorr = gsl_stats_lag1_autocorrelation(sample1, stride, SIZE); // calculate lag 1 autocorrelation
    correlation = gsl_stats_correlation(sample1, stride, sample2, stride, SIZE); // calculate correlation
    covariance = gsl_stats_covariance(sample1, stride, sample2, stride, SIZE); // calculate covariance

    printf("lag 1 autocorrelation %g\n", lag1_autocorr);
    printf("correlation: %g\n", correlation);
    printf("covariance: %g\n", covariance);


    return(0);
}
```

```
lag 1 autocorrelation -0.369567
correlation: -0.0190039
covariance: -0.15041
```

Vector operations

- the GSL library has its own *struct* for vectors
- only the non-trivially implementable functions are shown
- all functions:
  - Vectors and Matrices — GSL 2.8 documentation

VECTORS/vector_examples1.c/.exe

Vector operations

```c
#include <stdio.h>
#include <gsl/gsl_vector.h>

int main(){
    int i,a;
    a = 4;
    double max,min,argmax,argmin;

    // allocate space for vectors, these are structs and not standard C arrays
    gsl_vector *u = gsl_vector_alloc(5);

    for(i=0; i<5; i++){
        gsl_vector_set(u, i, i*10); // write the value i*10 to the i-th position of u
    }

    // calculate max,min,argmax,argmin, the return type of these functions is double
    max = gsl_vector_max(u);
    min = gsl_vector_min(u);
    argmax = gsl_vector_max_index(u);
    argmin = gsl_vector_min_index(u);

    printf("max: %g\n", max);
    printf("min: %g\n", min);
    printf("argmax: %g\n", argmax);
    printf("argmin: %g\n", argmin);

    return(0);
```

- space allocation
- writing to a vector
- min,max,argmin,argmax
- don’t forget:

```c
gsl_vector_free (v);
```

Vector operations

- BLAS library support for vector operations
- BLAS Support — GSL 2.8 documentation
- level1 → vector operations
- level2 → vector-matrix operations
- level3 → matrix operations

Level 1	Vector operations, e.g. $y = \alpha x + y$
Level 2	Matrix-vector operations, e.g. $y = \alpha A x + \beta y$
Level 3	Matrix-matrix operations, e.g. $C = \alpha A B + C$

Vector operations

Function nomenclature
(not only for vectors)

DOT	scalar product, $x^T y$
AXPY	vector sum, $\alpha x + y$
MV	matrix-vector product, Ax
SV	matrix-vector solve, $inv(A)x$
MM	matrix-matrix product, AB
SM	matrix-matrix solve, $inv(A)B$

GE	general
GB	general band
SY	symmetric
SB	symmetric band
SP	symmetric packed
HE	hermitian
HB	hermitian band
HP	hermitian packed
TR	triangular
TB	triangular band
TP	triangular packed

S	single real
D	double real
C	single complex
Z	double complex

```
int gsl_blas_zdotu(const gsl_vector_complex *x, const gsl_vector_complex *y, gsl_complex *dotu)
```

Vector operations

VECTORS/vector_examples2.c/.exe

```c
#include <stdio.h>
#include <gsl/gsl_blas.h>
#define SIZE 4

int main(){

    /*
    int gsl_blas_dsdot(const gsl_vector_float *x, const gsl_vector_float *y, double *result)

    input:
        - two vectors of type gsl_vector_float (struct)
        - a scalar value of type float (the result will be stored there)

    output:
        - an integer: 0 --> success, 1 --> failure
    */

    gsl_vector_float *u , *v;        // new type!
    int i, exit_code;
    float j;
    double result;

    u = gsl_vector_float_alloc(SIZE); // designated allocation function for this type
    v = gsl_vector_float_alloc(SIZE);

    j = 0.0;
    for(i=0;i<SIZE;i++){
        gsl_vector_float_set(u, i, j*10); // designated value set function for this type
        gsl_vector_float_set(v, i, j*100);
        j = j + 1.0;
    }

    exit_code = gsl_blas_dsdot(u, v, &result); // t(u)*v, '&result' reffers to the address of the variable 'result'

    printf("Exitcode: %d\n", exit_code);
    printf("t(u)*v = %g\n", result);

    return(0);
}
```

Matrices

- Vectors and Matrices — GSL 2.8 documentation
- matrices are stored in row-major order (as in all of C, Fortran matrices are stored in column-major order)

Matrices

- basic matrix usage

```
#include <stdio.h>
#include <gsl/gsl_matrix.h>
#define N 10 // number of rows
#define P 5  // number of columns

int main(){
    int i,j;
    double val;
    gsl_matrix * M = gsl_matrix_alloc (N, P); // allocate space

    // 'populate' matrix (remember row-major ordering)
    for(i=0; i<N; i++){
        for(j=0; j<P; j++){
            gsl_matrix_set(M, i, j, (i+j));
            //gsl_matrix_set(matrix, row, col, value)
        }
    }

    // get elements of matrix
    for(i=0;i<N;i++){
        for(j=0;j<P;j++){
            val = gsl_matrix_get(M, i, j); // returns a double
            printf("%g\n", val);
        }
    }

     gsl_matrix_free(M); // free allocated space

    return(0);
}
```

MATRIX/matrix_example1.c/.exe

Matrices

- reading a matrix from a file

MATRIX/matrix_example2.c/.exe

```
int main(){
    FILE *input_file;
    gsl_matrix *mat;
    int dim, status_read, i, j;
    dim = 3;
    input_file = fopen("input_file.txt", "r");
    mat = gsl_matrix_alloc(dim,dim);
    status_read = gsl_matrix_fscanf(input_file, mat);  // read from an ASCII file, THE FILE DIMENSIONS MUST MATCH EXACTLY
    printf("reading exitcode: %d\n", status_read);

    printf("File contents: \n");
    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            printf("%g ", gsl_matrix_get(mat, i, j));
        }
        printf("\n");
    }
    return(0);
}
```

Matrices

- writing matrices to a file

MATRIX/matrix_example3.c/.exe

```c
int main(){
    /*
    To ensure that the dimensions of the output file match the dimensions
    of the matrix we must loop through the matrix, and write each element
    manuall to the file.
    */
    int dim,i,j,value;
    double element;
    FILE *output_file;
    gsl_matrix *mat;
    dim = 3;

    mat = gsl_matrix_alloc(dim,dim);
    value = 1;
    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            gsl_matrix_set(mat,i,j,value);
            value = value + 1;
        }
    }
    output_file = fopen("output_file.txt", "w");

    // loop through the matrix and write each single element
    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            element = gsl_matrix_get(mat,i,j);
            fprintf(output_file, "%g ", element);
        }
        fprintf(output_file,"\n"); // add a newline after the line has ended

    }
    return(0);
}
```

Matrices (matrix views)

- create matrices from standard C arrays
- this is another struct *(gsl_matrix_view)*
- Vectors and Matrices — GSL 2.8 documentation

Matrices (matrix views)

```c
#include <stdio.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>  // BLAS support is needed
#define NROW 2
#define NCOL 3

int main(){
    int i,j;
    double val;
    double arr[6] = {1.0,2.0,3.0,4.0,5.0,6.0}; // our array
    gsl_matrix_view MAT_view; // define the matrix VIEW
    gsl_matrix *MAT; // define the MATRIX
    MAT_view = gsl_matrix_view_array(arr, NROW, NCOL); // create the view from the array
    MAT = &MAT_view.matrix; // convert the view to a standard gsl matrix

    // do something with the matrix
    for(i=0;i<NROW;i++){
        for(j=0;j<NCOL;j++){
            val = gsl_matrix_get(MAT,i,j);
            printf("%g ", val);
        }
        printf("\n");
    }
    return(0);
}
```

MATRIX/matrix_example4.c/.exe

Matrix vector operations

- level 2 in BLAS
- BLAS Support — GSL 2.8 documentation

Matrix vector operations

int gsl_blas_dgemv(CBLAS_TRANSPOSE_t TransA, double alpha, const gsl_matrix *A, const gsl_vector *x, double beta, gsl_vector *y)

These functions compute the matrix-vector product and sum $y = \alpha op(A)x + \beta y$, where $op(A) = A, A^T, A^H$ for `TransA` = `CblasNoTrans`, `CblasTrans`, `CblasConjTrans`.

- for me the notation is weird
- **y:** result vector, **x:** vector, **A:** matrix, **alpha&beta:** constants
- **TransA:** specify if the matrix is transposed or not (`TransA` = `CblasNoTrans`, `CblasTrans`, `CblasConjTrans`.), these are GSL constants
- to get a ‘classic’ y = A*x multiplication, set alpha = 1 and beta = 0

Matrix vector operations

MATRIX_VECTOR/matrix_vector_example1.c/.exe

```c
#include <stdio.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_matrix.h>
#define N 2
#define P 3
```

```c
int main(){
    int i,j,status;
    gsl_vector *u = gsl_vector_alloc(P); // vector
    gsl_vector *res = gsl_vector_alloc(N); // result vector
    gsl_matrix *M = gsl_matrix_alloc(N,P); // matrix
    double alpha_param = 1.0;
    double beta_param = 0.0;

    for(i=0;i<P;i++){
        gsl_vector_set(u,i,i);
    }

    for(i=0;i<N;i++){
        for(j=0;j<P;j++){
            gsl_matrix_set(M,i,j,i);
        }
    }

    // perform matrix vector multiplication
    status = gsl_blas_dgemv(CblasNoTrans, alpha_param, M, u, beta_param, res);
    printf("Exitcode: %d\n", status);

    printf("Result: \n");
    for(i=0;i<N;i++){
        printf("%g ", gsl_vector_get(res,i));
    }
    printf("\n");

    return(0);
}
```

Matrix operations

- **transposition**
- could be written by hand, but not easily on the GSL structs
- [Vectors and Matrices — GSL 2.8 documentation]()

`int gsl_matrix_transpose_memcpy(gsl_matrix *dest, const gsl_matrix *src)`

This function makes the matrix `dest` the transpose of the matrix `src` by copying the elements of `src` into `dest`. This function works for all matrices provided that the dimensions of the matrix `dest` match the transposed dimensions of the matrix `src`.

`int gsl_matrix_transpose(gsl_matrix *m)`

This function replaces the matrix `m` by its transpose by copying the elements of the matrix in-place. The matrix must be square for this operation to be possible.

Matrix operations

MATRIX_MATRIX/matrix_matrix_example1.c/.
exe

```c
#include <stdio.h>
#include <gsl/gsl_matrix.h>
#define N 10 // rows
#define P 4 // columns

int main(){
    int i,j,status;
    double val;
    gsl_matrix * M = gsl_matrix_alloc(N, P);
    gsl_matrix * M_T = gsl_matrix_alloc(P,N);  // t(M)

    for(i=0;i<N;i++){
        for(j=0;j<P;j++){
            gsl_matrix_set(M,i,j,i+j);
        }
    }

    //transpose
    status = gsl_matrix_transpose_memcpy(M_T, M);
    printf("Exitcode: %d\n", status);
    printf("_____________________________________\n");

    printf("Transposed matrix:\n");
    for(i=0;i<P;i++){
        for(j=0;j<N;j++){
            val = gsl_matrix_get(M_T, i, j);
            printf("%g ", val);
        }
        printf("\n");
    }
    return(0);
}
```

Matrix operations

- **matrix inversion**
- this function performs matrix transposition based on its LU decomposition
- **LU:** the LU decomposition of our original matrix, **p:** permutation vector, **inverse:** the inverse of our matrix

```
int gsl_linalg_LU_invert(const gsl_matrix *LU, const gsl_permutation *p, gsl_matrix *inverse)
```

Matrix operations

MATRIX_MATRIX/matrix_matrix_example2.c/.exe

```c
#include <stdio.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>
#define DIM 3

int main(){
    double arr[DIM*DIM] = {4.0,7.0,2.0,3.0,5.0,1.0,1.0,2.0,3.0};
    double val;
    int i,j;
    gsl_matrix_view MAT_view;
    gsl_matrix *MAT;
    MAT_view = gsl_matrix_view_array(arr, DIM, DIM);
    MAT = &MAT_view.matrix;

    // LU decomposition specific
    gsl_matrix *LU_decomp_MAT = gsl_matrix_alloc(DIM,DIM); // matrix to hold the LU decomposition
    gsl_matrix_memcpy(LU_decomp_MAT, MAT); // copy the contents of MAT to LU_decomp_MAT
    int sign;
    gsl_permutation * perm_vec = gsl_permutation_alloc(DIM);
    gsl_linalg_LU_decomp(LU_decomp_MAT, perm_vec, &sign); // this function works inplace

    gsl_matrix *MAT_inv = gsl_matrix_alloc(DIM, DIM);
    gsl_linalg_LU_invert(LU_decomp_MAT, perm_vec, MAT_inv); // matrix inversion

    for(i=0;i<DIM;i++){
        for(j=0;j<DIM;j++){
            val = gsl_matrix_get(MAT_inv, i, j);
            printf("%g ", val);
        }
        printf("\n");
    }

    return(0);
}
```